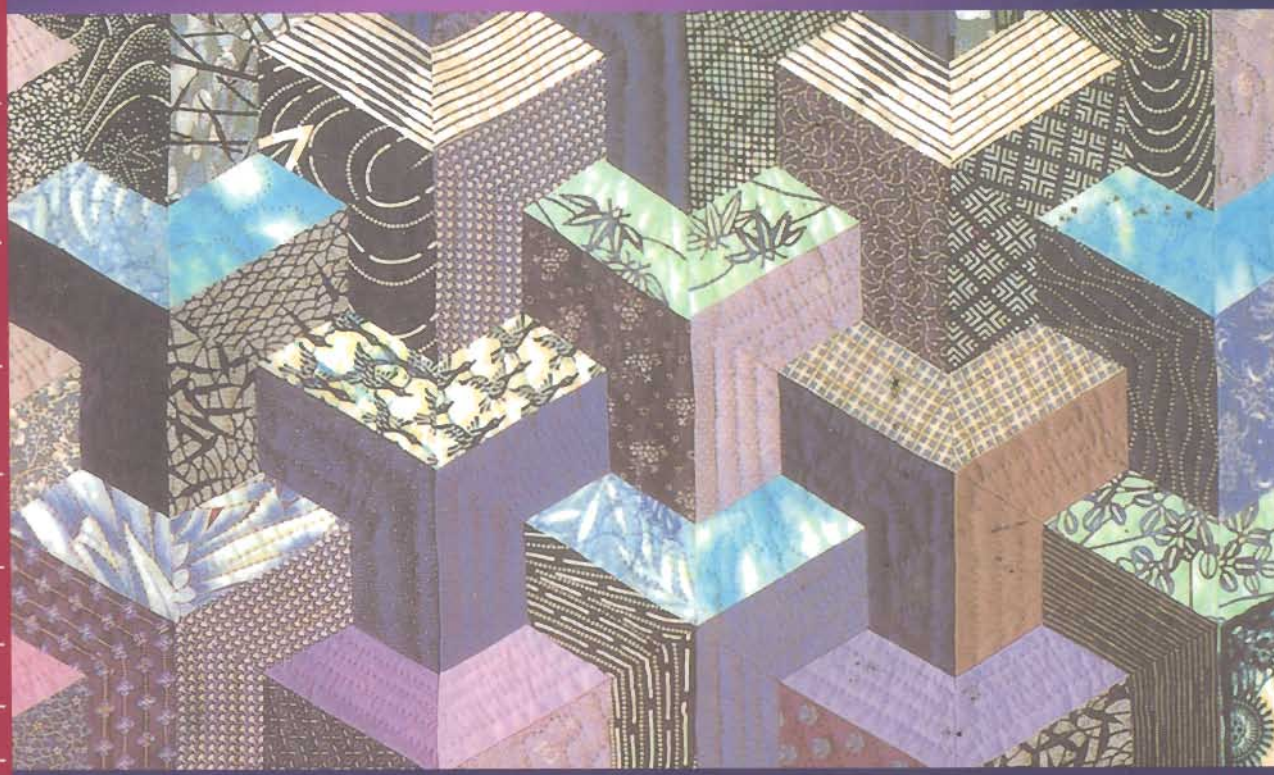
Diseño y Administración de Bases de Datos

Gary W. Hansen
James V. Hansen



2ª edición

Prentice Hall

CONTENIDO

Prólogo XV

Prólogo a la edición en español XIX

I LAS BASES DE DATOS Y SUS CONTEXTOS

1 Los sistemas de bases de datos y la evolución de la tecnología de bases de datos 3
- La evolución de la tecnología de bases de datos 4
- Un caso: La compañía International Product Distribution 5
- Sistemas orientados a los archivos 6
  - Necesidad del proceso de acceso directo a datos 8
  - La información como un recurso 11
  - Otras limitaciones de los sistemas de archivo tradicionales 12
- Los sistemas de bases de datos 14
  - Sistemas de los modelos jerárquico y en red 15
  - Sistemas de bases de datos relacionales 17
- Direcciones actuales-plataformas cliente/servidor 21
- Los sistemas de base de datos: el hardware, el software, los datos, las personas 23
  - El hardware 23
  - El software 24
  - Los datos 27
  - Las personas 27
  - Interrelación (relación) entre los cuatro componentes del sistema . 28
- Resumen 29
- Preguntas de comprobación 30
- Problemas y ejercicios 31
- Proyectos y cuestiones profesionales 32

2 Servicios de celebraciones: un ejemplo de utilización de base de datos 33
- El caso de Servicios para Celebraciones (Catering) 34
  - Antecedentes 34

Operación de los negocios 35
Servicios de empleados 35
Adquisición 35
Planificación de tareas 35
Promoción 36
Estado financiero 36
• Diseño e implementación de la base de datos 36
Descripción del ciclo de recepción venta/efectivo 36
De las entidades a las tablas 38
Creación de una tabla 38
Diagrama revisado del modelo de datos 38
Establecer las comprobaciones de validación 41
• Características adicionales de los SGBD 43
Introducción de datos a través de formularios simples 43
Un formulario multi-tabla: el formulario solicitud 44
Obtener información para la gestión a partir de la base de datos 46
Resumen 49
Preguntas de comprobación 49
Problemas y ejercicios 49
Proyectos y cuestiones profesionales 49

3 Sistemas de bases de datos en las organizaciones 50
Compartir datos y bases de datos 52
Compartir datos entre unidades funcionales 52
Compartir datos entre diferentes niveles de usuarios 53
Compartir datos entre diferentes localidades 55
• El papel de la base de datos 57
Planificación estratégica de bases de datos 58
La necesidad de planificar la base de datos 58
El proyecto de planificación de la base de datos 58
El ciclo de vida del desarrollo de la base de datos (CVDBD) 60
Bases de datos y gestión de control 61
• Diseño de la base de datos 61
Formación del usuario 63
Seguridad e integridad de los datos 63
Rendimiento del sistema de base de datos 63
Riesgos y costos de las bases de datos 64
Conflictos en las organizaciones 64
Fracasos en el desarrollo de proyectos 64
Malfuncionamiento del sistema 64
Costes imprevistos 65
Necesidad de personal cualificado 65
Separar la representación lógica y física de los datos 65
• Arquitectura de tres-niveles de una base de datos 66
Uso de la arquitectura tres-niveles en este libro 66
Desarrollo de la base de datos 67
Diseño de base de datos y el CVDS tradicional 68
El ciclo de vida del desarrollo de la base de datos (CVDBD) 69
El caso de la Corporación Zeus 70
Planificación preliminar 71
Estudio de viabilidad 71
Definición de requisitos 72
Diseño conceptual 74
Implementación 75
Evaluación y perfeccionamiento del esquema de base de datos 76

Construir capacidades en el desarrollo de bases de datos 76
Resumen 77
Preguntas de repaso 78
Problemas y ejercicios 79
Proyectos y cuestiones profesionales 82

II Diseño de base de datos

4 Principios del diseño conceptual de base de datos 84
Realidad, definición de requisitos y modelado conceptual de datos ... 85
Realidad y modelos 85
Modelos conceptuales de datos 88
Fundamentos 89
Objetos 89
Especialización y generalización 90
Interrelaciones 91
Cardinalidad 93
Atributos 95
Ejemplos 99
Ejemplo 1: El modelo de datos de un banco 99
Ejemplo 2: Huerto frutícola de Stratton 102
Ejemplo 3: Un problema de lógica 103
Construir modelos conceptuales de datos a partir de los informes existentes 105
Caso: Servicios de Consultoría Manwaring 105
Un modelo de datos para las compras 105
Un modelo de datos para la facturación de proyectos 107
Agregación 108
Ejemplo 4: Compañía Constructora Premier 111
Ejemplo 5: Huerto frutícola de Stratton (continuación) 113
Ejemplo 6: Servicios de Consultoría Manwaring (continuación) 116
Modelado conceptual de objetos contra objetos físicos 120
El problema de una biblioteca 121
Crear el modelo de datos de la biblioteca 122
Fabricación de piezas 125
Objetos conceptuales para los servicios de consulta Manwaring 127
Integración de vistas: Un ejemplo 129
Resumen 130
Preguntas de comprobación 131
Problemas y ejercicios 132
Proyectos y cuestiones profesionales 136

5 El modelo de datos relacional 137
El modelo de datos relacional y el desarrollo de sistemas 138
El modelo de datos relacional: Conceptos fundamentales 139
Relaciones 139
Valores Nulos 141
Claves 141
Claves externas (ajenas, *foreing*) 143
Restricciones de integridad 144
El proceso de normalización 145
Primera forma normal 146
Dependencias funcionales 147

Segunda forma normal 147
Tercera forma normal 149
Cuarta forma normal 151
Otras formas normales 152
Transformando el modelo conceptual en un modelo relacional 153
Transformar conjuntos de objetos y atributos 153
Transformar modelos sin claves externas 154
Transformar la especialización y la generalización de conjuntos de objetos 154
Transformar interrelaciones 155
Transformar conjuntos de objetos agregados 158
Transformar interrelaciones recursivas 160
Ejemplos de transformaciones: Servicios de Consultoría Manwaring . 161
Comparación del modelado de datos conceptual y relacional 163
Resumen 165
Preguntas de repaso 166
Problemas y ejercicios 167
Proyectos y cuestiones profesionales 171

III Implementación de base de datos relacionales

6 El álgebra y el cálculo relacional 174
Una ventaja revolucionaria en la manipulación de datos 175
Álgebra relacional 176
Unión 178
Intersección 179
Diferencia 180
Producto 180
Selección 182
Protección 184
Reunión 186
División 193
Asignación 195
Un Ejemplo Adicional 195
Cálculo relacional 196
La lista resultado y sentencia de calificación. 197
El cuantificador existencial 198
El cuantificador universal 200
La dificultad relativa del álgebra relacional y el cálculo relacional. ... 202
Resumen 203
Preguntas de repaso 203
Problemas y ejercicios 204
Proyectos y cuestiones profesionales 206

7 Implementación relacional con SQL 207
Implementación relacional: introducción 208
Definición de tablas y esquemas 210
Definición de esquemas 211
Tipos de datos y dominios 211
Definiendo tablas 213
Manipulación de datos 217
Consultas simples 218
Consultas multi-tablas 221
Subconsultas 224

EXISTS y NOT EXISTS 227
Funciones integradas (Built-in functions) 229
GROUP BY y HAVING 231
Funciones integradas con subconsultas 233
Operaciones del álgebra relacional 234
Operaciones de modificación de la base de datos 240
Usar SQL con lenguajes de procesamiento de datos 241
Definición de vistas 242
Restricciones sobre consultas y actualizaciones sobre vistas 243
El esquema de información 244
Resumen 246
Preguntas de repaso 246
Problemas y ejercicios 248
Proyectos y cuestiones profesionales 251

8 Implementación relacional con lenguajes de consulta gráficos 253
Introducción 254
Manipulación de datos 255
Consultas simples 255
Consultas de múltiples tablas 259
Funciones integradas (Built-In Functions) 261
GROUP BY 263
Operaciones que modifican la base de datos 265
Paradox para Windows 266
Definición y entrada de datos 267
Manipulación de datos 270
Seleccionar atributos y filas 270
Consultas conjuntivas y disyuntivas 270
Lenguaje de aplicación 274
Resumen 275
Preguntas de repaso 275
Problemas y ejercicios 276
Proyectos y cuestiones profesionales 277

9 Sistemas de bases de datos cliente/servidor 278
Introducción: un resumen de los conceptos cliente/servidor 279
Definición de las tablas de la base de datos en el sistema de base de datos servidor 281
Creación de tipos de datos definidos por el usuario 281
Definiendo tablas individuales 284
Lenguaje de definición de datos: Sumario y ejemplo final 290
Manipulación y programación del servidor de datos 292
Lenguaje flujo-de-control 292
Procedimientos almacenados 297
Una introducción a los disparadores 300
Desarrollando aplicaciones cliente 302
El enfoque de PowerBuilder 302
Usando PowerBuilder 303
Construir una aplicación 305
Crear Data Windows 308
Crear Windows 312
Algunos comentarios finales 316
Resumen 317
Preguntas de repaso 317

Problemas y ejercicios 319
Proyectos y cuestiones profesionales 321

10 Organizacion física de los sistemas de base de datos 322
Introducción 323
Acceso físico a la base de datos 324
Formas de almacenamiento físico 324
Almacenamiento secundario 325
Bloques de almacenamiento físico 326
Factores de rendimiento del disco 327
Tiempo de posicionamiento 327
Tiempo de activación de la cabeza 328
Retraso de rotación 328
Velocidad de transferencia de datos 328
Tiempo de transferencia de datos 328
Formatos de almaceuamieuto de datos en disco 330
Formatos de pistas 330
Formatos de registros 331
Gestión de entrada/salida 332
Organización de archivos y métodos de direccionamiento 333
Organización secuencial de un archivo 333
Organización secuencial-indexada de un archivo 333
Organización directa de un archivo 334
Funciones Hash estática 334
Funciones Hash dinámicas 337
Implementación de interrelaciones lógicas 339
Listas enlazadas 339
Listas invertidas 341
Árbol-equilibrado indexado (B^+-árbol) 341
Implementación cliente/servidor de la indexación árbol-balanceado (árbol-equilibrado) 347
Correspondencia entre estructuras de datos lógicas y estructuras de datos físicas 348
Correspondencia en las bases de datos relacionales 348
Correspondencia en las bases de datos en red 349
Correspondencia en las bases de datos jerárquicas 351
Acceso por la clave-secundaria 352
Optimización de las consultas 354
Combinar las operaciones de selección y reunión 354
Combinar las operaciones proyección, selección y reunión 355
Resumen 355
Preguntas de comprobación 356
Ejercicios y problemas 357
Proyectos y cuestiones profesionales 359

IV ADMINISTRACIÓN DEL ENTORNO DE BASES DE DATOS

11 La administración y el control de la base de datos 362
Gestión de la base de datos: Una visión panorámica 363
Funciones del ABD 364
Comunicación con los usuarios 364
Establecimiento de normas y procedimientos 366
Objetivos del ABD 367

Integridad de la base de datos 368
Restricciones de integridad en SQL 368
Restricciones de integridad en Query-by-Example 368
Procesamiento de transacciones 369
Control de la concurrencia 370
Seguridad de la base de datos 375
Autentificación (Autenticación) 375
Autorización y vistas 377
Encriptación 381
Recuperación de la base de datos 382
Fuentes de fallos 382
Procedimientos de recuperación 382
Resumen 387
Preguntas de comprobación 387
Problemas y ejercicios 388
Proyectos y cuestiones profesionales 390

12 Sistemas de bases de datos distribuidas 391
¿Por qué sistema de bases de datos distribuidas? 392
Un modelo general de SBDD *(DDS)* 394
Diseño de sistemas de bases de datos distribuida 397
Estrategias y objetivos 397
Distribución de archivos no fragmentados 401
Procesamiento distribuido de las consultas 404
Semirreuniones (*semijoins*) 404
Integridad de los datos en los sistemas de bases de datos distribuidas . 406
Protocolo de cierre de dos-fases 406
Bloqueo distribuido 407
Bloqueo distribuido de dos-fases 408
Marcas de tiempo 409
Recuperación de la base de datos 411
Sistemas cliente/servidor 411
Conceptos generales 411
Aplicaciones de bases de datos 412
SQL Server para Windows NT 412
Resumen 413
Preguntas de comprobación 414
Ejercicios y problemas 415
Proyectos y cuestiones profesionales 417

13 Selección e implementación de un SGBD 418
Análisis de las necesidades de información para la gestión 420
Determinar los requisitos de la aplicación 421
Mantener la consistencia de los datos 421
Requisitos de los tiempos de respuesta 422
Funciones y capacidades de los SGBD 422
El diccionario/directorio de datos 422
Seguridad e integridad de los datos 423
Capacidades de consulta, de manipulación de datos y de informes . 424
Soporte a los requisitos de programación especializada 424
Opciones de organización física de los datos 424
Clasificación de los requisitos de las propiedades de un SGBD 424
Reunir datos sobre la disponibilidad de propiedades y sobre el rendimiento 425
Adquirir datos de los vendedores 425

Pruebas de evaluación *(Benchmark)* 426
Modelos de evaluación 427
Modelo de puntuación 427
Análisis de la envoltura de datos 428
Cuestiones de implementación 432
Administración de la base de datos 432
Probar la base de datos 433
Preparar a los usuarios para el cambio 434
Cargar la base de datos 435
Mantenimiento de la base de datos 435
Resumen 436
Preguntas de comprobación 437
Problemas y ejercicios 438
Proyectos y cuestiones profesionales 438

V TEMAS AVANZADOS

14 Sistemas avanzados: sistemas de bases de datos orientados a objetos y sistemas basados en el conocimientos 441
Una evolución hacia los sistemas de bases de datos orientados a objeto 443
Intentos de proporcionar una representación de los datos más poderosa 443
La contribución de la programación orientada a objetos (OOP) 443
Abstracción de clase 443
Clases derivadas y herencia simple 445
Agregación 445
Un ejemplo extendido que demuestra la herencia múltiple 446
Desarrollos de los sistemas de bases de datos orientados a objeto 449
Gemstone 449
Vbase 449
Orion 449
PDM 449
IRIS 450
O_2 450
Un lenguaje genérico de base de datos de objetos 450
El modelo de datos del Lerner College 450
Formular consultas con TextQuery 455
El formato básico de las soluciones de consulta en TextQuery 455
Definir las direcciones de las interrelaciones 458
Navegar sobre atributos e interrelaciones 460
Asignar nombres a los conjuntos derivados 464
Expresiones calificadas de comparación de conjuntos 464
Conectores booleanos 466
Introducción a los sistemas basados en el conocimiento 468
Conocimiento y bases de datos 469
Representación del conocimiento con reglas 471
Formación de reglas 471
Reglas en PROLOG 472
Una aplicación simple de base de datos en PROLOG 476
Aspectos fundamentales de PROLOG 476
La estructura de una aplicación en PROLOG 477
Aplicación de base de datos 479
Datalog 481

Lenguaje de datos lógicos (LDL) 481
Resumen 481
Preguntas de repaso 482
Problemas y ejercicios 482
Proyectos y cuestiones profesionales 485

VI EL LEGADO DE LOS SISTEMAS DE BASE DE DATOS

15 El modelo de datos en red 489
Antecedentes históricos 491
Conceptos básicos y definiciones 492
Estructnras tres-niveles 492
Registros y conjnntos 492
Interrelación del modeloen red con la semántica del modelado conceptual 495
Transformación de conjnntos de objetos e interrelaciones nno-muchos 496
Transformación de interrelaciones *n*-arias 497
Transformación de interrelaciones muchos-muchos 498
Lenguaje de definición de los datos (DDL) de DBTG 499
Del modelo de datos al esquema 499
Del esquema al subesquema 502
Lenguaje de manipulación de los datos (DML) del DBTG 503
Facilidades de recuperación y actualización del DBTG 505
Facilidades del DBTG para el procesamiento de conjuntos 507
Operaciones de conjuntos 507
Clasificación de la asociación de conjuntos 508
IDMS/R - Un SGBD del DBTG 510
Evaluación de CODASYL del DBTG 510
Representación de los datos 510
Lengnaje de manipulación de los datos 511
Restricciones de integridad 511
Implementación 511
Resumen 512
Preguntas de repaso 512
Problemas y ejercicios 513
Proyectos y cuestiones profesionales 515

16 El modelo de datos jerárquico 516
Introducción 517
Conceptos básicos y definiciones 518
Las interrelaciones del modelo jerárquico para la semántica del modelado conceptual 523
Transformar interrelaciones uno-muchos 524
Transformar interrelaciones muchos-muchos 525
La arquitectura IMS [1] 527
Definir la base de datos física - El DBD 528
Definir la base de datos lógica - El BEP 529
Métodos de acceso de IMS 530
HSAM 530
HISAM 530

[1] Siglas del término en inglés *IBM's Information Management System.*

HDAM 531
HIDAM 532
El lenguaje de la manipulación de datos del IMS 532
El área de trabajo del programa 532
DL/1 : Una panorámica 533
Get Unique (GU) 533
Get Next (GN) 534
Get Next Within Parent (GNP) 534
Get Hold 535
Replace (REPL) 535
Delete (DLET) 536
Insert (INSRT) 536
La evaluación del modelo de datos jerárquico 536
La representación de los datos 536
El lenguaje de manipulación de datos 537
Resumen 538
Preguntas de repaso 538
Ejercicios y problemas 539
Proyectos y cuestiones profesionales 541

Glosario 543

Bibliografía 553

Índice analítico 567

PRÓLOGO

Este libro está concebido para ser utilizado como texto en cursos iniciales dirigidos a estudiantes de sistemas de información, tanto de nivel de pregrado como de posgrado. Como tal puede integrarse fácilmente dentro de un amplio currículum de sistemas de información que incluye programación, análisis de sistema, diseño de sistemas, comunicación de datos, etc.

El libro brinda una comprensiva cobertura de los sistemas de bases de datos en los negocios. Está organizado alrededor del ciclo de vida del desarrollo de la base de datos, lo que nos ofrece un marco adecuado para la discusión del diseño conceptual, la implementación de la base de datos y la gestión del entorno. A lo largo del libro se usará el modelado conceptual de datos, primero como base para el diseño conceptual y luego, posteriormente, como medio para la implementación, usando los modelos relacional, jerárquico y en red. También se cubren los lenguajes de todos los modelos, así como las implementaciones de bases de datos cliente/servidor y los sistemas basados en el conocimiento. Todos estos temas se enmarcan dentro de características tácticas y estratégicas de gestión que incluyen la planificación de la base de datos, la selección del SGBD, la administración de la base de datos, la seguridad y la integridad y las bases de datos distribuidas.

Características principales

- El libro tiene una organización simple y directa de los temas basada en el ciclo de vida del desarrollo de la base de datos. Esto facilita que los estudiantes puedan ver cómo los conceptos se relacionan entre sí y dónde el material que está siendo estudiado encaja dentro de las actividades generales del desarrollo del sistema de una organización. Los capítulos iniciales presentan los conceptos de planificación estratégica de datos, cómo compartir los datos y la arquitectura de tres niveles, haciendo énfasis en el concepto de que los aspectos técnicos de los sistemas de bases de datos se determinan por las necesidades de información genuinas de los negocios.
- En el Capítulo 4 se proporciona una amplia cobertura del diseño conceptual. Ésta incluye la cobertura tradicional básica de objetos, interrelaciones y atribu-

tos, así como también un estudio en profundidad de temas más avanzados, tales como la agregación.

- El diseño conceptual de datos crea las bases de nuestro enfoque para el diseño de base de datos en los modelos relacional (Capítulos 5 y 6), en red (Capítulo 15) y jerárquico (Capítulo 16). Debido a la estructura modular de los capítulos, el diseño de estos modelos puede enseñarse sin hacer referencias al modelado orientado a objetos.
- A lo largo del libro se estudian cuatro casos de negocios que caracterizan a una compañía distribuidora, una manufacturera, una compañía constructora y una firma de consultoría. Se incluyen diálogos que se relacionan directamente con problemas reales de bases de datos en los negocios.
- Se proporciona flexibilidad para los diferentes enfoques que puedan usar diferentes profesores o instructores. Los capítulos son antocontenidos, lo que permite que puedan seleccionarse según se desee.
- El libro incluye un tratamiento completo de los lenguajes relacionales: SQL (Capítulo 7), Query-by-Example (Capítulo 8) y el álgebra y el cálculo relacional (Capítulo 6), así como los lenguajes estándares para las bases de datos jerárquicas y en red. También se incluyen nuevos aspectos, como los relativos a los SGBDs cliente/servidor (Capítulo 9). El capítulo de SQL incluye una extensa cobertura del SQL-92 ANSI estándar y el capítulo sobre QBE discute los aspectos generales del lenguaje, así como su implementación en un sistema comercial en particular. A lo largo de estos capítulos sobre los lenguajes, se presentan de forma tutorial muchos ejemplos y se dan explicaciones detalladas de cada aspecto del lenguaje.

▼ Nuevo material en esta edición

- Esta edición incluye un tratamiento sustancial de los sistemas cliente/servidor, con un capítulo completo dedicado a SGBDs cliente/servidor y a entornos de desarrollo de aplicaciones. Específicamente se examinan los sistemas de gestión de bases de datos Oracle y SQL Server y el entorno de desarrollo de aplicaciones PowerBuilder. Se exploran los principios que introducen estos sistemas, lo que brinda un complemento poderoso a las bases teóricas que se encuentran a lo largo del libro.
- Se ha mejorado el capítulo sobre SQL, que ahora incluye una cobertura significativa del estándar SQL-92. Los aspectos relativos al lenguaje de definición de datos, claves foráneas y otras definiciones de restricciones, nuevos tipos de datos y las operaciones del álgebra relacional se exploran con algún detalle.
- El Capítulo 2 ofrece el caso de Servicios para Celebraciones, que fue previamente incluido como un suplemento. Esto permite al estudiante de forma inmediata poner "las manos encima" de un sistema de base de datos, a la vez que recibe una visión del proceso de desarrollo del sistema.
- Los capítulos existentes se han reelaborado y mejorado. Esto ha dado como resultado, por ejemplo, que se haya incorporado nuevo material sobre la implementación física (*hashing* dinámico), sistemas de bases de datos distribuidas (asignación óptima de archivos) y una nueva sección completa dedicada a los sistemas de bases de datos orientadas a objeto.

▼ Enseñar y aprender con el texto

- Cada capítulo comienza con un escenario que proviene de uno de los cuatro casos a los que se les va dando continuidad. Cada escenario presenta una situación realista que tiene que ver con el tema del capítulo y destaca los temas claves del capítulo. Este escenario es seguido inmediatamente por una lista de los objetivos del

capítulo, que establece los conocimientos que debe alcanzar el estudiante. El balance de cada capítulo incluye otros numerosos ejemplos basados en la continuidad de los cuatro casos, así como otros varios casos.
- Se incluye un gran número de figuras para ilustrar el modelado de las bases de datos, soluciones de las preguntas y otros tópicos. El capítulo sobre cliente/servidor incluye ilustraciones tomadas directamente de las pantallas de los computadores que se usaron para desarrollar los ejemplos.
- El libro tiene una amplia bibliografía actualizada de artículos de investigación, de modo que los estudiantes interesados pueden explorar los temas en profundidad.
- Los materiales, al final del capítulo, se han estandarizado. Cada uno incluye un resumen de lo tratado en el mismo. Además incluye Preguntas de repaso, que se pueden responder directamente a partir del texto; problemas y ejercicios que estimulan y desarrollan una comprensión en profundidad, y proyectos y cuestiones profesionales para un trabajo más avanzado.
- Un glosario de los términos claves se va disponiendo en los márgenes del texto, de modo que esté disponible para una fácil referencia. Al final del texto se brinda también un extenso glosario general.

▼ Aclaraciones sobre la notación

A lo largo del libro se presentan muchos ejemplos para ilustrar conceptos en el modelado de datos, los lenguajes, etc. A menudo en estos ejemplos se usan nombres de datos compuestos. Algunas veces aparecen conectados con guiones y otras veces con subrayados. Esta selección está condicionada por la industria. Por ejemplo, los nombres de datos en el capítulo del modelo en red usan guiones porque eso es lo que emplea el modelo CODASYL estándar, mientras que los capítulos sobre SQL y QBE usan subrayado.

El estudiante que desarrolle un proyecto o el caso particular de un sistema real deberá determinar qué convenio requiere el sistema. Una implementación con éxito de alguno de nuestros ejemplos pudiera requerir convertir guiones a subrayados, o al contrario. En cualquier caso esta conversión no es difícil y cualquier comparación de nuestros resultados con los del estudiante no debiera causar confnsión.

▼ Al instructor

Nuestro objetivo a lo largo del libro es ofrecer nn material que sirva lo mismo al estudiante que se inicia como al estudiante avanzado, presentándolo en una forma clara. También se han destacado diferentes puntos de vista. Por tanto, si desea presentar el tema desde el punto de vista de la gestión o desde el punto de vista técnico, puede seleccionar los capítulos según le convenga para su enfoque. Si su interés es enfatizar el diseño de bases de datos o los lenguajes de manipulación de datos, el libro le brinda un fuerte soporte en ambas áreas. Finalmente se pneden también seleccionar los capítulos para seguir un enfoque tradicional al tema, o puede tomar temas más avanzados, haciendo énfasis en las líneas actuales de investigación. Puede usarse la siguiente lista de capítulos por áreas diferentes para orientar el curso según sus requisitos.

Capítulos de Gestión
1, 2, 3, 11, 12, 13

Capítulos Técnicos
2, 4, 5, 6, 7, 8, 9, 10, 14, 15, 16

Capítulos de Diseño de Bases de Datos
4, 6, 10, 15, 16

Capítulos de Lenguajes
6, 7, 8, 9, 14, 15, 16

Capítulos Avanzados...
4, 7, 9, 14

Este libro se ha usado en una secuencia de dos semestres en la universidad de *Brigham Young*. El primer semestre se dedica al diseño y gestión de bases de datos, cubriendo los Capítulos 1-6, 10 y selecciones del 11-16. El segundo semestre se centra en los lenguajes de bases de datos y el soporte de decisiones y cubre los Capítulos 7, 8, 9 y partes del 11-16.

▼ Suplementos

Con vistas a ayudar a los profesores que usen el Diseño y Gestión de Base de Datos, los autores han preparado un *Manual del Instructor (Profesor)*. Este manual del instructor le ofrece al mismo sugerencias para la enseñanza, respuestas sugeridas para todos los problemas y ejercicios, del libro preguntas de verificación y transparencias de figuras seleccionadas del texto. También se proveen casos adicionales, escritos por Jim Hansen, que ofrecen tutoriales en Microsoft *Access y Paradox para Windows*.

▼ Agradecimientos

Un grupo de revisores nos ha brindado ideas y sugerencias muy útiles que han ayudado a preparar la versión final de este libro. Ellos son: Kirk P. Arnett, Mississipi State University; David J. Auer, Western Washington University; Elin B. Cohen, Wichita State University; Orlando E. Katter, Jr., Wingate Collage; Rajeev Kaula, Southwest Missouri State University; Richard Kerns, East Carolina University; Constance A. Knapp, Pace University; Paul E. Laski, Nothern Virginia Community College; Ronald Maestas, New Mexico Highlands University; Scott McIntyre, University of Colorado; Victor Meyers, Hawaii Pacific University; Stevan Mrdalj, Eastern Michigan University; William P. Wagner, Villanova University y Ahmed Zaki, College of William and Mary.

PRÓLOGO A LA EDICIÓN ESPAÑOLA

El estado del arte de los SGBD

En Enero de 1997, la prestigiosa revista americana de bases de datos DBMS publicaba un editorial recordando su décimo aniversario y el modo en que el actual director accedió al ejemplar número uno de dicha revista: "*De modo casual*", como suele suceder con los grandes acontecimientos relacionados con el mundo de la cultura informática. En ese primer número de otoño de 1988 el editorial examinaba los servidores de bases de datos y su rol en el modelo emergente cliente/servidor. La mayoría de los restantes artículos se centraban en el desarrollo de aplicaciones multiusuario Xbase.

En el número citado de 1997, que iniciaba el 10.º año de su publicación, aventuraba su director que era difícil predecir qué habría en el editorial del 2007. Cliente/servidor fue una nueva idea de hace diez años. Hoy existen numerosas voces que señalan o preconizan "su muerte" y la sustitución por el nuevo Cliente/Servidor Universal constituido por las redes públicas Internet y privadas (corporativas) Intranet.

Durante el año 1997 la citada revista DBMS está examinando las tendencias actuales más innovadoras, tales como desarrollo de aplicaciones de bases de datos para Internet e Intranet, DBMSs (SGBD) relacionales/objetos, sistemas distribuidos, componentes, etc.

En esencia, se podrá decir que los navegadores de la Web y las demás tecnologías Web para uso en Internet e Intranet están cambiando radicalmente el modo de desarrollo de aplicaciones de bases de datos de los próximos años y del tercer milenio. Si a ello se une los nuevos proveedores y también el advenimiento de las nuevas computadoras NetPC y NC, será preciso tener presente que cada día será necesario poseer los conocimientos teóricos y prácticos de diseño y construcción de bases de datos. Si dichos conocimientos siempre han sido necesarios para el desarrollo profesional de software, hoy día es una necesidad vital, ya que si bien el impacto de la Web en servidores DBMS es muy acusado, las bases de selección y gestión de un SGBD no han cambiado drásticamente.

El último número de Junio de 1997 anuncia la implantación de los servidores universales, pero a través de un proceso de migración lento. Acaba de anunciarse Oracle 8, que su fabricante presenta como la base de datos para la informática de red (Network Computer) que permitirá gestionar grandes cantidades de datos sobre redes de computadoras con seguridad, fiabilidad y economía. Además de las características anteriores, y siguiendo las nuevas tendencias, se incorpora el modelo objeto, de modo que se puede considerar que Oracle 8 es un ORDBMS (Sistema de gestión —administración— de bases de datos relacional/objetos), como ha confirmado recientemente la propia revista oficial de Oracle. Es decir, se construyen tecnologías de objetos sobre un fundamento relacional. Otra de las características sobresalientes de Oracle 8 es incluir la capacidad para que los programadores de Java puedan acceder a bases de datos. El estado actual de las tecnologías de bases de datos está soportado por sistemas y objetos distribuidos, ORBs, tecnología Web, C/S, etc. Esto implica que los nuevos usuarios de bases de datos deberán contemplar una formación en estas tecnologías debido esencialmente a que todas las versiones del futuro soportarán estas características. Por ello se necesitan libros innovadores en teoría de bases de datos como es el que tenemos en nuestras manos.

▼ Fundamentos teóricos del libro

La materia *Bases de datos* es una asignatura contemplada en todos los curriculum de las carreras de Ingeniería Informática y de Sistemas Computacionales o Ciencias de la Computación; eso requiere del estudiante o lector avanzado disponer de buenos libros que proporcionen fundamentos teóricos rigurosos y en profundidad, contemplando todas las tecnologías más recientes.

Así un buen libro de base de datos deberá contemplar como mínimo: *Fundamentos teóricos, el modelo de datos, diseño de las bases de datos, los sistemas de administración, sistemas cliente/servidor, sistemas orientados a objetos, sistemas distribuidos y sistemas basados en el conocimiento.*

Por otra parte, uno de los grandes problemas que existen en la actualidad son los sistemas heredados *(legacy database systems)*; es decir, aquellos que representan a los primeros sistemas de bases de datos (modelo de datos en red, modelo de datos jerárquico y los primeros modelos relacionales). Aunque estos sistemas no son tan elegantes como los actuales sistemas relacionales de objetos y basados en el conocimiento, siguen siendo de gran uso y tienen todavía una larga vida. Es muy importante que un profesional del software, ingeniero, analista o programador, se acostumbre a convivir con ellos mientras se produce la transición a los modernos sistemas de bases de datos. Por esta circunstancia es muy interesante que un libro universitario de bases de datos contenga una parte dedicada a esos sistemas antiguos o heredados, soportes todavía de la informática de gran cantidad de empresas.

Administración y diseño de bases de datos de G.W. Hansen y J.V. Hansen contempla con rigor y profundidad todos los temas fundamentales de bases de datos citados anteriormente: fundamentos teóricos y organización de la base de datos; diseño de bases de datos (modelo de dato conceptual, modelo de dato relacional); implementación de la base de datos relacionales (álgebra relacional, SQL, sistemas de base de datos cliente/servidor); administración y gestión del entorno de bases de datos y sistemas de bases de datos distribuidos y basados en el conocimiento. Por último, contempla los sistemas heredados de bases de datos (modelo de datos en red y jerárquico), considerando los métodos y arquitecturas tradicionales IDMS, IMS, etc.

▼ El libro como elemento de ayuda al estudiante universitario y profesional

El contenido de la obra de Hansen y Hansen y su estructuración le permiten ser empleada como libro de texto para cursos universitarios y profesionales de nivel de iniciación y medio de bases de datos, así como para introducciones a cursos avanzados en temas tales como bases de datos orientadas a objetos y de sistemas distribuidos. Por su amenidad y organización, también resulta una obra útil para profesionales autodidactas que deseen iniciarse, reciclarse o perfeccionarse en el estudio de las bases de datos.

Es un libro bien estructurado que puede adaptarse muy bien y con facilidad a un curso anual en Facultades y Escuelas de Ingeniería Informática o de Sistemas Computacionales. La gran cantidad de ejercicios y problemas que se incluyen en todos los capítulos facilita al alumno y al lector autodidacta el aprendizaje gradual de la materia. Con la ayuda de unas prácticas de laboratorio mediante un buen sistema SGBD tal como Oracle, Sybase, Informix, dB2, etc., un alumno al terminar el curso anual dominaría la gestión, diseño e implementación de las bases de datos y estaría preparado para iniciarse en temas avanzados en sistemas de bases de datos distribuidas, de objetos y basadas en el conocimiento.

Consideramos que el tratamiento dado por los autores al apasionante mundo de las bases de datos hacen que sea el libro idóneo para cursos en carreras universitaria de Informática, Telecomunicaciones o Sistemas Computacionales a partir del 2.º año en los centros que siguen cursos anuales, o a partir del tercero o cuarto cuatrimestre/semestre en aquellos centros universitarios que siguen el sistema de cuatrimestre/semestre.

▼ Epílogo

La obra de Gary W. Hansen y James V. Hansen viene a incorporarse a la amplia bibliografía en inglés existente de bases de datos y a la no muy numerosa, pero excelente —en español— publicada por diferentes editoriales españolas o latinoamericanas, por lo que se complementará perfectamente con las ya existentes y además ocupará el espacio significativo notable que supone tratar los temas innovadores ya citados.

Creemos que su temario actualizado permitirá la difusión de la teoría, diseño y construcción de las bases de datos tanto en España como en Latinoamérica y constituirá una magnífica noticia en el ámbito editorial hispano, siempre necesitado de buenas obras de nivel universitario y profesional.

Luis Joyanes Aguilar
Director del Departamento de Lenguajes
y Sistemas Informáticos e Ingeniería de Software
Facultad de Informática. Escuela Universitaria de Informática
Universidad Pontificia de Salamanca. *Campus Madrid*

PARTE

UNO

LAS BASES DE DATOS Y SUS CONTEXTOS

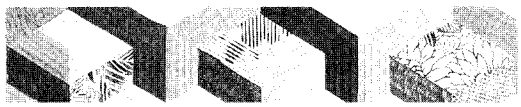
En la parte I se presentan las bases de datos y los sistemas de bases de datos. Como en esta parte se esboza el contexto orgánico e histórico de las bases de datos, el estudio se centrará en las respuestas a las preguntas siguientes:

¿Qué es una base de datos y qué es un sistema de base de datos?
¿Cómo se originaron los sistemas de bases de datos?
¿Cómo se desarrollan los sistemas de bases de datos?
¿Cómo las organizaciones utilizan y controlan las bases de datos?

El capítulo 1 se refiere a las dos primeras preguntas anteriores. En este capítulo se revisa el desarrollo histórico de los sistemas de bases de datos. Se profundiza en la manera en que las necesidades de los negocios han conformado el desarrollo de la tecnología y cómo la información implícita en los datos ha llegado a convertirse en un recurso corporativo valioso. El estudio concluirá con una descripción de los cuatro componentes principales de un sistema de base de datos: el hardware, el software, los datos y las personas.

El capítulo 2 se refiere a la tercera pregunta. Aquí se examina una aplicación típica y se presenta el proceso de desarrollo de un sistema de base de datos paso a paso. Se podrá observar cómo puede desarrollarse un sistema de base de datos real sobre un sistema típico. Si el lector entra en un sistema de gestión de bases de datos, podrá desarrollar pantallas y definiciones de tablas para desarrollar este ejemplo de base de datos.

El capítulo 3 responde a la última pregunta, analizando los sistemas de bases de datos en su contexto orgánico e introduciendo y estudiando el ciclo de vida del desarrollo de una base de datos. Se discute sobre la necesidad de compartir los datos a todos los niveles de una organización, se describe el proceso de planificación de los datos estratégicos y se esboza el papel que juega el personal que administra la base de datos en el control y la protección de la misma. También se muestra que el diseño de una base de datos, aunque ejecutado al mismo tiempo que el diseño del sistema, debería ser considerado con un alcance superior al del diseño de cualquier otro sistema. Una base de datos bien concebida se convertirá entonces en el fundamento para muchos sistemas de aplicación.



CAPÍTULO

1

LOS SISTEMAS DE BASES DE DATOS Y LA EVOLUCIÓN DE LA TECNOLOGÍA DE BASES DE DATOS

Evolución de la tecnología de las bases de datos
Un caso: La compañía International Product Distribution
Sistemas orientados a archivos
- Descubrimiento de la necesidad del procesamiento con acceso directo
- La información como un recurso
- Otras limitaciones de los sistemas de archivos tradicionales
  - Redundancia de los datos
  - Pobre control de los datos
  - Capacidades inadecuadas de manipulación de los datos
  - Esfuerzo excesivo de programación

Sistemas de bases de datos
- Sistemas sobre el modelo jerárquico y sobre el modelo en red
- Sistemas de bases de datos relacionales

Direcciones actuales-plataformas cliente/servidor
Sistemas de bases de datos: el hardware, el software, los datos, las personas
- El hardware
- El software
  - Diccionario/directorio de datos
  - Mecanismos de seguridad e integridad de los datos
  - Acceso concurrente a los datos para múltiples usuarios
  - Consultas e informes de los datos orientados al usuario
  - Facilidades para el desarrollo de las aplicaciones
- Los datos
- Las personas
- Interrelación entre los cuatro componentes del sistema

Resumen
Preguntas de comprobación
Problemas y ejercicios
Proyectos y cuestiones profesionales

Susan Broadbent, CEO, y Sanford Mallon, CIO (director de sistemas de información), de la compañía International Product Distribution, discuten animadamente sobre la tecnología de los sistemas. Susan, viendo la oportunidad de hacer una broma, le provocó: "¿Sandy, quieres que pasemos a un sistema de base de datos cliente/servidor*?" "¿Es ésta una más de tus ideas atolondradas?" Sanford le respondió: "¿Atolondrada? ¿Alguna vez he hecho yo una propuesta que no hubiera sido concebida con brillantez y ejecutada con absoluta precisión?"*

"Bien, veamos. Cuando tú llegaste aquí, nos llevaste de nuestro sistema manual a un sistema orientado a archivos. Luego vino el sistema de bases de datos en red, y después las relacionales. Ahora quieres ir hacia una plataforma cliente/servidor. Si estos esquemas fueron concebidos brillantemente, ¿por qué hay que cambiarlos al cabo de pocos años?"

Sanford se rió. La sonrisa de Susan le decía que ella era consciente de las razones de cada cambio y de los beneficios significativos cosechados por la compañía en cada ocasión. Él contestó: "Ha sido un largo camino mantenerse avanzando a la par que la tecnología, ¿cierto o falso?"

"Sí. Pero tú has sido excepcional al permanecer al tanto de los desarrollos y movernos hacia ellos cuando más contribuirían a nuestro negocio. Y pensar que todo empezó de un modo tan simple..."

Susan Broadbent y Sanford Mallon reflejan su satisfacción sobre varias décadas de crecimiento del negocio basado en la tecnología de acceso a los datos. En este capítulo se analizan: (1) el desarrollo de esta tecnología en su impacto sobre los negocios y cómo ha estado afectada ella misma por las necesidades comerciales, y (2) los cuatro componentes principales de un sistema de base de datos moderno —el *hardware*, el *software*, los datos y las personas—. Después de leer este capítulo, el usuario debería ser capaz de:

- Conocer los puntos fuertes y débiles de los sistemas de archivo de acceso secuencial y de acceso directo.
- Explicar cómo la información ha llegado a considerarse como un recurso valioso en las organizaciones modernas.
- Describir la evolución histórica de los sistemas de bases de datos jerárquicos, en red y relacionales y las necesidades de los negocios que condujeron a su desarrollo.
- Explicar cómo los cuatro componentes —el hardware, el software, los datos y las personas— se integran para constituir los sistemas de bases de datos actuales.

▼ La evolución de la tecnología de bases de datos

La sofisticación de la tecnología moderna de las bases de datos es el resultado de la evolución que a lo largo de varias décadas ha tenido lugar en el procesamiento de los datos y en la gestión de la información. La tecnología de acceso a los datos se ha desarrollado desde los métodos primitivos de los años cincuenta hasta los potentes e integrados sistemas de hoy en día, arrastrados de un lado por las necesidades y las demandas de la administración y, de otro, restringida por las limitaciones de la tecnología.

Las expectativas de la administración han crecido paralelamente a la evolución de la tecnología. Los primeros sistemas de procesamiento de datos ejecutaron las tareas administrativas para reducir el papeleo. Más recientemente, los sistemas se han expandido hacia la producción y la gestión de la información, la que se ha convertido en un recurso vital para las compañías. Actualmente, la función más importante de los sistemas de bases de datos consiste en proporcionar el fundamento a los sistemas de información para la gestión corporativa.

La implantación de los cambios tecnológicos ha estado guiada por las necesidades genuinas de los negocios. La administración únicamente autorizará un nuevo sistema informático cuando vea un beneficio claro al modificar el costo del sistema. Y a pesar de las dificultades y los riesgos, en muchos casos se han experimentado ventajas. Además, el final no está aún a la vista y tardará tiempo en llegar. Las nuevas tecnologías, tales como las bases de datos orientadas a objetos y la plataforma cliente/servidor, plantean nuevos problemas y darán lugar a sistemas más potentes para el futuro.

La estrecha interrelación que existe entre la tecnología de las bases de datos y las necesidades empresariales puede ser más fácil de comprender si nos acercamos a la experiencia de la compañía International Product Distribution.

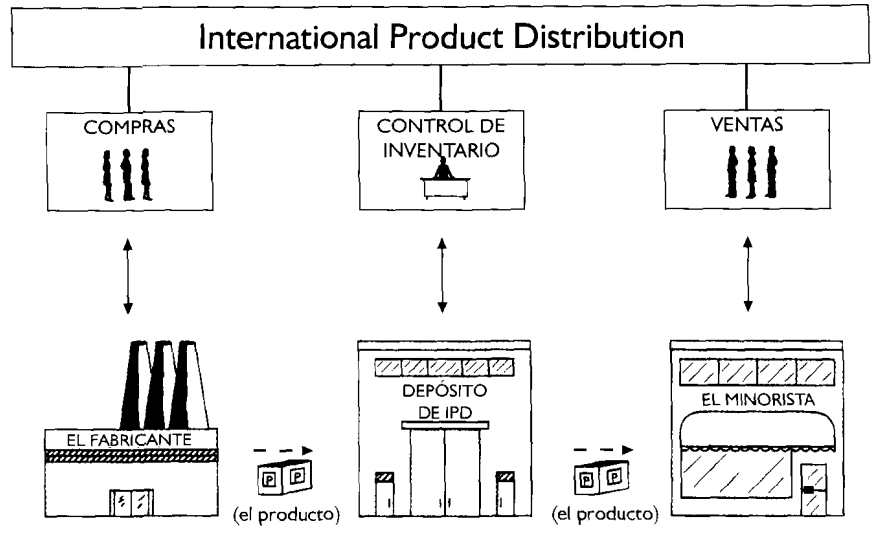
▼ Un caso: La compañía International Product Distribution

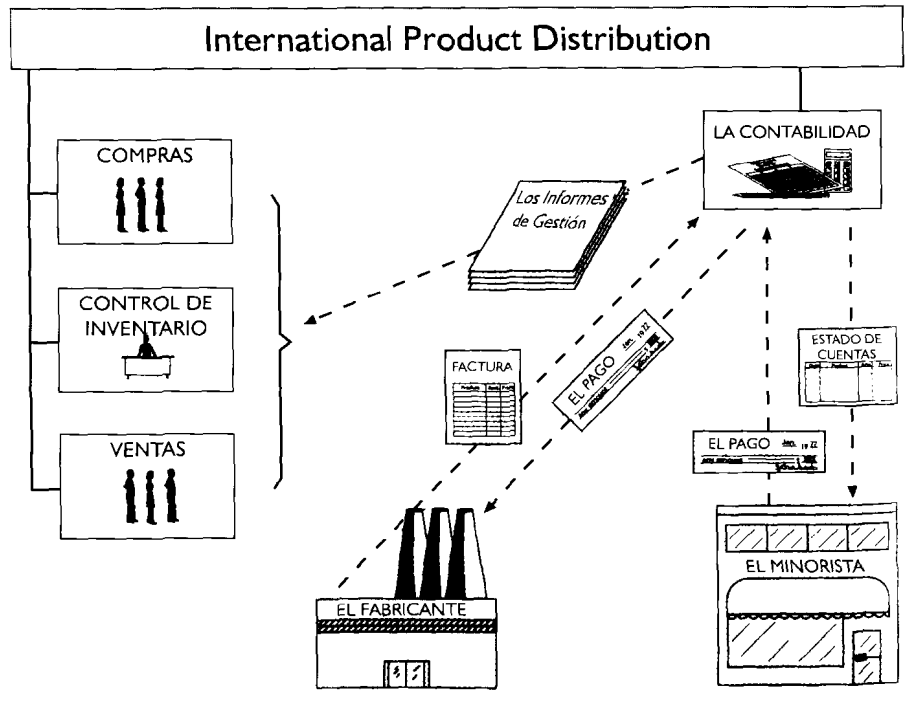
Susan Broadbent es la fundadora, propietaria y presidenta de International Product Distribution (IPD), que vende por encima de 3.500 productos de más de 300 fabricantes en países de todo el mundo. La IPD tiene su sede en Chicago, con oficinas internacionales en Bruselas, Buenos Aires, Lagos, Nueva Delhi, Tokio y Sydney. Alrededor de 2.700 representantes de ventas trabajan localmente en más de 100 países, cada uno de los cuales informa a una oficina regional. La compañía tiene unas ventas anuales de alrededor de 500 millones de dólares y unos beneficios en torno a 50 millones de dólares.

Después de vender ropa de niños durante un cierto número de años producida por un mismo fabricante a los grandes almacenes de Chicago, Susan decidió que podría aumentar significativamente sus ingresos si representaba a varios fabricantes. Así, fundó International Product Distribution. Su concepto era sencillo. (1) Identificar los fabricantes en diversos países cuyos productos mostraran constantemente una calidad reconocida. (2) Identificar a los minoristas que apoyaran la venta de tales productos. (3) Establecer fuertes relaciones comerciales, tanto con los fabricantes como con los minoristas, proporcionando a estos últimos los productos de los fabricantes apropiados.

Inicialmente, contaba con muy poco personal y se relacionaba únicamente con minoristas de Chicago y fabricantes del medio oeste. Sin embargo, en poco tiempo estuvo abasteciendo productos a los comerciantes desde St. Louis hasta Cleveland. Sus primeras ventas internacionales las hizo a unos almacenes en Toronto. Después de tres años, la compañía IPD tuvo representantes en Europa y dos años más tarde en Tokio. Seguidamente se abrieron oficinas en Buenos Aires, Sydney, Lagos y Nueva Delhi. Cada oficina empleaba compradores y representantes de ventas. Los productos comprados se vendían en el país de fabricación o se exportaban para su venta en otro país. La Figura 1.1 ilustra la interrelación existente entre la compañía IPD y sus clientes, tanto los minoristas como los suministradores. Como se puede ver, los productos circulan desde el fabricante a los almacenes de IPD y desde ahí al minorista.

Al principio, los registros de las ventas, de las compras de los productos y el inventario de la compañía se efectuaban a mano. No obstante, al finalizar el segundo año, el volumen del negocio se había extendido tanto que fue necesario comprar un minicomputador para almacenar esta información y confeccionar informes, facturas y pagos, tal y como se muestra en la Figura 1.2. Sanford Mallon fue contratado para desarrollar este sistema orientado a archivos y para dirigir un grupo formado por programadores, operadores para la entrada de datos y personal de operaciones.





▼ Sistemas orientados a los archivos

Los sistemas computacionales se utilizaron inicialmente en los negocios para funciones de contabilidad: las cuentas a cobrar, las cuentas a pagar, la nómina y otras. Estas funciones eran imprescindibles para que el negocio funcionara. Por consiguiente, el costo de los computadores que podían ejecutar estas funciones era fácil de justificar. Por ejemplo, el

esfuerzo manual requerido para la nómina o las cuentas a cobrar era tan grande que, si un sistema automatizado pudiera reemplazar al sistema manual, pagaría por sí solo la inversión en corto tiempo.

sistema de procesamiento de datos. Un sistema automatizado para procesar los datos de los registros de una organización.

Estos sistemas se llamaron **sistemas de procesamiento de datos** debido a que ejecutaban las funciones habituales de tratamiento de los registros. Sin que esto pueda sorprender a los programadores y a los analistas que diseñaron estos sistemas, se dejaron influir durante su programación por la inclinación natural de imitar los procedimientos manuales existentes. Así, los archivos en el computador se correspondían con los archivos de papel y los registros en los archivos del computador contenían la información que podía almacenar una carpeta individual de un archivo en un sistema manual.

La Figura 1.3 muestra algunos archivos y datos de prueba del sistema original orientado a archivos de la compañía International Product Distribution. Cada tabla representa un archivo del sistema. Es decir, se tiene un archivo CLIENTE, un archivo REPRESENTANTE_DE_VENTAS, un archivo PRODUCTO y así sucesivamente. Cada fila representa un registro en el archivo. De esta manera, el archivo PRODUCTO contiene tres registros. Cada uno de estos registros contiene los datos sobre un producto diferente. Los conceptos o campos de los datos individuales en el archivo PRODUCTO son ID_PRODUCTO, DESC_PRODUCTO, ID_FABRICANTE, COSTO y PRECIO [1].

CLIENTE

ID_CLIENTE	NOMB_CLIENTE	DIRECCIÓN	PAÍS	BALANCE INICIAL	PAGOS HASTA EL MES
100	Hnos. Watabe	Apdo. 241, Tokio	Japón	45.551	40.113
101	Matlzl	Salzburg	Austria	75.314	65.200
105	Jefferson	B 918, Chicago	EUA	49.333	49.811
110	Gómez	Santiago	Chile	27.400	28.414

REPRESENTANTE DE VENTAS

ID_REP	NOMB_REP	ID_DIRECTOR	OFICINA	COMM_%
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokio	11
23	François Moire	35	Bruselas	19
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokio	10

PRODUCTO

ID_PRODUCTO	DESC_PRODUCTO	ID_FABRICANTE	COSTO	PRECIO
1035	Suéter	210	11.25	22.00
2241	Lámpara de Mesa	317	22.25	33.25
2518	Escultura de Bronce	253	13.60	21.20

VENTA

FECHA	ID_CLIENTE	ID_REP	ID_PRODUCTO	CANTIDAD	PRECIO TOTAL
02/08	100	14	2241	200	6650,00
02/12	101	23	2518	300	6360,00
02/12	101	23	1035	150	3300,00
02/19	100	39	2518	200	4240,00
02/22	101	23	1035	200	4400,00
02/25	105	10	2241	100	3325,00
02/25	110	37	2518	150	3180,00

FABRICANTE

ID_FABRICANTE	NOMB_FABRICANTE	DIRECCIÓN	PAÍS
210	Confecciones Kiwi	Aukland	Nueva Zelanda
253	Obras de Bronce	Lagos	Nigeria
317	Lámparas Llana	Lima	Perú

[1] Para respetar el sentido del humor del autor, los nombres de personas, organizaciones, lugares y direcciones, utilizados en los ejemplos a lo largo de todo el libro, se han mantenido igual que en el original (N. del T.).

Por el momento, supongamos que se tiene acceso a todos estos archivos secuencialmente. Es decir, que cada registro puede leerse y procesarse únicamente después que todos los registros que lo preceden en el archivo hubieran sido leídos. Eso era lo que ocurría en la compañía IPD en los años sesenta, cuando el almacenamiento en disco era todavía relativamente caro. La mayoría de los archivos se almacenaban en cinta magnética y se tenía acceso y se procesaban los registros en secuencia. Por lo general, estos archivos se procesaban en lotes, lo que significaba que todos los registros de un archivo se procesaban al mismo tiempo, normalmente cada noche después del cierre del negocio.

Los archivos se utilizaban para varias aplicaciones diferentes. Por ejemplo, el programa de las cuentas a cobrar generaba los estados de cuentas para los clientes. Utilizaba los archivos CLIENTE y VENTA. Ambos archivos se ordenaban según el campo ID_CLIENTE y se fusionaban para crear un estado impreso, como se muestra en la Figura 1.4. El campo SALDO_INICIAL en el archivo CLIENTE se actualizaba para reflejar los nuevos cargos. Los pagos, que se recibían y procesaban previamente por otro programa, teniendo en cuenta el archivo CLIENTE, se registraban en el campo PAGOS_HASTA_MES_ACTUAL del archivo CLIENTE y se mostraban en el estado de cuentas.

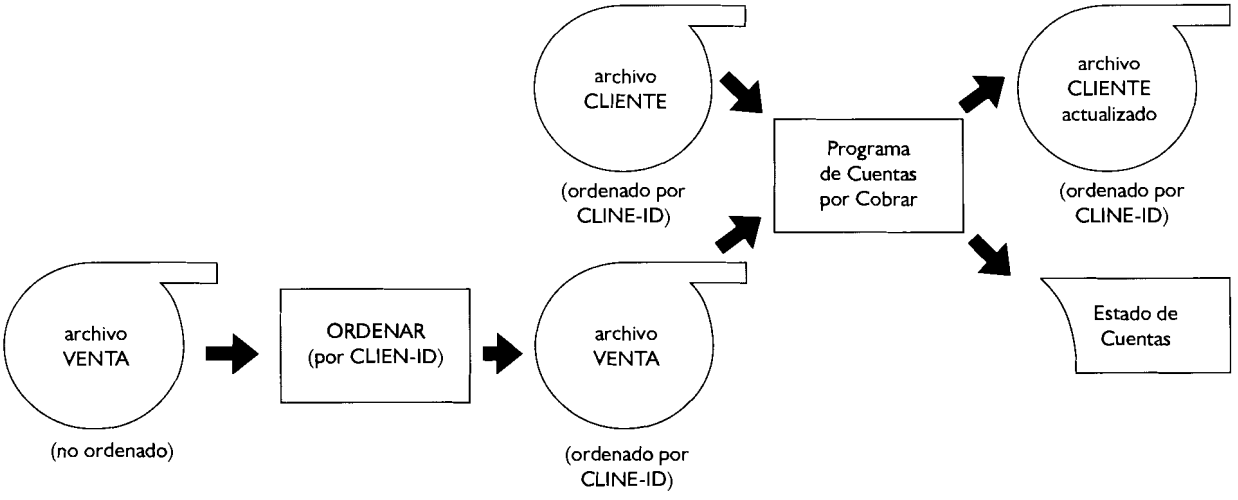
programa de aplicación

Un programa como éste que realiza una tarea específica de valor práctico en una empresa es un **programa de aplicación** o un elemento de un *software de aplicación*. El conjunto de programas que trabajan colectivamente para realizar un grupo de tareas relacionadas entre sí es un *sistema de aplicación*.

Necesidad del proceso de acceso directo a datos

Las limitaciones de los sistemas orientados a archivos puramente secuenciales no los privaron de ser herramientas eficaces para producir pagos, facturas y otros informes una o dos veces al mes. Sin embargo, para ejecutar muchas tareas rutinarias en los negocios se necesita el **acceso directo a los datos**—la capacidad de tener acceso y procesar directamente un registro dado sin ordenar primero el archivo o leer los registros en secuencia. Para comprender algunos de los problemas que ocurren cuando *no* existe el acceso directo a los datos, se verán dos ejemplos del pasado en IPD.

acceso directo a los datos



"Sandy, no comprendo por qué nuestros operadores de entrada de datos tienen que introducir tanto la cantidad como el precio total siempre que introducen una transacción de venta. Nosotros tenemos el precio del producto en el archivo PRODUCTO. ¿Por qué nuestra gente no puede introducir solamente la cantidad y el ID del producto, que en cualquier caso introducen siempre, y que el sistema calcule el precio total?"

Estamos en la época inicial del procesamiento de datos en la compañía IPD y Susan Broadbent está preocupada por la cantidad de trabajo innecesario que hacen los operadores. Cuantos más datos tienen que introducir, más caro resulta. Su pregunta es natural, teniendo en cuenta que la compañía IPD ha invertido ya una considerable suma de dinero para comprar una computadora, desarrollar un paquete de programas y pagar al personal que lo realiza.

"Susan, nosotros tenemos un sistema de archivos secuenciales. Esto significa que el acceso a todos los registros en un archivo se tiene que realizar en orden. Cuando ejecutamos el programa de cuentas a cobrar, trabajamos con los archivos CLIENTE y VENTA, por tanto, no tenemos acceso al registro del producto."

"Me temo que no comprendo lo que estás diciendo."

"Bien, aquí hay algunos datos de prueba de los archivos. (Véase Figura 1.3.) Observa cómo el archivo CLIENTE está ordenado según el ID del Cliente. Antes de ejecutar el programa de las cuentas a cobrar, también ordenamos el archivo VENTA según el ID del Cliente. Entonces, cuando ejecutamos las cuentas a cobrar, leemos ambos archivos. Cuando los IDs del Cliente coinciden en ambos archivos, actualizamos el saldo inicial en el archivo CLIENTE e imprimimos una factura. En la factura listamos los pagos hasta el mes actual y detallamos todos los encargos de ese cliente, que se muestran en el archivo VENTA. Hacemos eso manteniendo el mismo registro de CLIENTE mientras leemos, sucesivamente, cada uno de los registros de VENTA que le corresponden. Observa cómo, en este ejemplo, el archivo VENTA muestra dos encargos del cliente 100, tres encargos del cliente 101 y un encargo de cada uno de los clientes 105 y 110. Como resultado, hemos minimizado la cantidad de tiempo empleada en la lectura de los datos de la cinta magnética, que es la parte más lenta de la ejecución del programa."

"Ahora mira qué sucedería si el programa tuviera que calcular el precio total. La primera venta en el archivo tiene el ID de Producto 2241. Fíjate que éste se encuentra en la mitad del archivo PRODUCTO. Por tanto, para obtener el precio de ese producto, tendríamos que leer hasta la mitad del archivo PRODUCTO. El registro de venta siguiente corresponde al producto 2518, que es el próximo registro en el archivo PRODUCTO. Pero después, el registro de VENTA que sigue corresponde al producto que está al comienzo del archivo PRODUCTO. Para obtener el precio de ese producto, estaríamos obligados a regresar al comienzo del archivo PRODUCTO. ¿ Ves cómo tendríamos que estar saltando continuamente a través del archivo PRODUCTO, leyendo hacia adelante un gran número de registros, rebobinando la cinta después y así sucesivamente?"

"Sí. Un gran consumo de tiempo," contestó Susan.

"Correcto. A largo plazo es más barato tener a nuestros operadores introduciendo los datos del precio total."

"Yo no estoy tan segura de que sea más barato, Sandy. Los representantes de ventas frecuentemente calculan mal el precio, entonces tenemos importes incorrectos en las facturas, clientes descontentos y una disminución de los beneficios. No podemos vivir con este problema. Necesitamos una solución."

"La mejor solución sería cambiar los archivos CLIENTE, PRODUCTO y otros más, a archivos de acceso directo (secuencial indexado). Por ejemplo, utilizaríamos el ID del Producto como la clave del archivo de los productos. Entonces, podríamos cambiar nuestro programa de cuentas a cobrar, de modo que tenga acceso a cualquier registro de ese archivo en el momento que queramos. Lo único que necesitamos saber es el ID del Producto correspondiente al registro que deseamos."

"¿Esto nos daría alguna otra ventaja?"

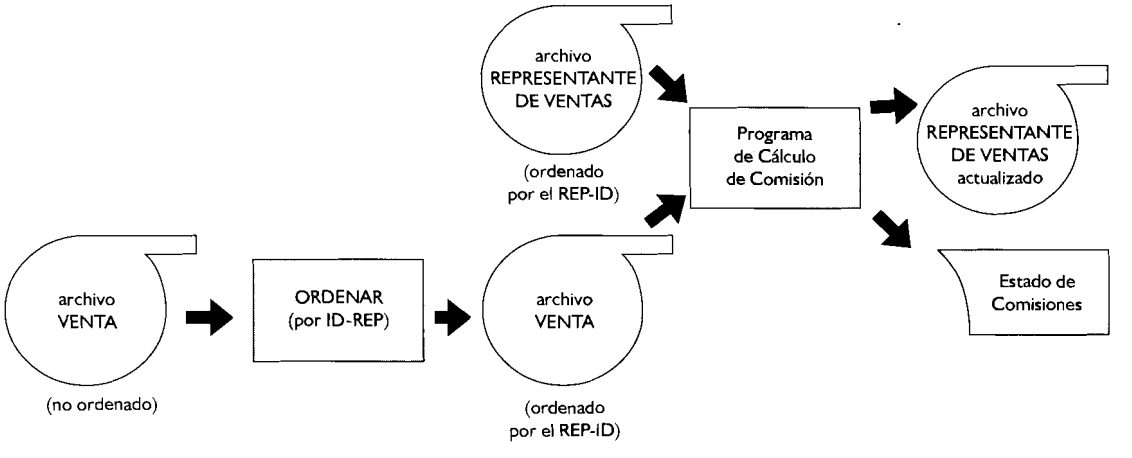
"Por supuesto. Por ejemplo, puedes obtener el saldo actualizado de los clientes siempre que lo necesites. También podemos utilizar los archivos de acceso directo para validar los ID de los Productos, los ID de los Fabricantes, los ID de los Representantes de Ventas y otros. Eso nos ayudará a eliminar los errores en las facturas de los clientes, que están provocando sus quejas."

"Eso suena fabuloso. ¿Cuándo lo podremos lograr?"

Volviendo hacia atrás por un momento se pueden ver más de cerca los problemas con los sistemas de archivos secuenciales. Para que se procesen las ventas contra el archivo CLIENTE en el programa de cuentas a cobrar, las ventas deben estar ordenadas por el ID del Cliente. Como es muy probable que las ventas se introduzcan aleatoriamente, el archivo VENTA debe ordenarse antes de que pueda utilizarse como entrada a este programa. Esto se muestra en la Figura 1.4. Aun cuando el archivo VENTA esté ordenado por el ID del Cliente, probablemente estará desordenado por completo con respecto al ID del Producto. Por esta razón no se puede calcular el precio total de una venta a partir del precio en el archivo PRODUCTO. Esto conduce a que los operadores introduzcan datos redundantes, lo que requiere un esfuerzo manual adicional e introduce una probabilidad mayor de error.

El requisito de que todos los archivos se procesen secuencialmente conduce también, de otra forma, a realizar trabajo adicional. Por ejemplo, otra de las aplicaciones de la compañía IPD calcula las comisiones por ventas y genera un informe de comisiones. Para calcular la comisión de un representante de ventas, hay que ordenar el archivo VENTA nuevamente, esta vez para ponerlo en orden según ID_REP. Ahora se puede procesar el archivo VENTA contra el archivo REPRESENTANTE_DE_VENTAS para crear los informes de comisiones correspondientes a los representantes de ventas (Figura 1.5).

Este ejemplo ilustra algunas de las limitaciones más serias de los sistemas de archivos puramente secuenciales. Estos problemas se resolvieron parcialmente con la introducción de los archivos de acceso directo y, particularmente, de los archivos secuenciales indexados (ISAM)[2], que se utilizaron ampliamente en los años sesenta. Los archivos de acceso directo permiten la recuperación de los registros aleatoriamente, a diferencia de los de acceso secuencial. El registro deseado en un archivo de acceso directo puede recuperarse inmediatamente. Los archivos ISAM son los archivos de acceso directo más utilizados en



[2] ISAM - *Indexed Sequential Access Method* (N. del T.).

clave. Campos de datos que identifican unívocamente un registro en un archivo.

procesos de tipo comercial. Estos archivos permiten que uno o más campos de datos –llamados conjuntamente **clave**— se utilicen para identificar precisamente qué registro se recuperará. Los archivos ISAM proporcionaron un medio potente y práctico para dotar de gran flexibilidad a las aplicaciones comerciales. (En el Capítulo 10 se discuten más detalladamente los aspectos físicos de los sistemas de bases de datos.)

Sin embargo, los archivos de acceso directo solamente proporcionaron una solución parcial. Para lograr una solución más completa a estos problemas fue necesario introducir los sistemas de gestión de bases de datos.

La información como un recurso

sistema de información. Sistema automatizado que organiza los datos para producir información.

En el diálogo anterior, Sanford sugirió la posibilidad de buscar el saldo de un cliente siempre que se necesitase. La pregunta "¿Cuál es el balance actual de los hermanos Watabe?" es una pregunta al **sistema de información**. Tales preguntas ilustran la transición tan significativa que tuvo lugar a finales de los años sesenta y principios de los setenta en el sentido de que los sistemas computacionales comerciales pasaron del *procesamiento de los datos al procesamiento de la información*. Este cambio reflejó una conciencia creciente con respecto a que la información era mucho más que simples registros relacionados con el negocio. Gradualmente, en las empresas comenzaron a darse cuenta del valor de la información y del enorme potencial que los sistemas computacionales representaban para organizar y administrar este recurso, recién reconocido como tal. A finales de los años sesenta, esto condujo a una demanda muy fuerte de **sistemas de información para la gestión**. Estos sistemas utilizarían los datos ya disponibles en el computador para brindar respuesta a un amplio espectro de preguntas de gestión o administración.

sistema de información para la gestión. Sistema que provee información para la gestión o la administración.

En este contexto se hace una distinción entre *datos* e *información*. Comúnmente, los **datos** se consideran como hechos aislados. Por ejemplo:

datos. Hechos aislados.

"Hnos. Watabe es una compañía ubicada en Japón"

es un hecho. Este hecho está contenido en un único registro en el archivo CLIENTE. Los archivos de un sistema contienen millares de tales hechos. Los archivos, por lo tanto, contienen datos. Por otra parte, la **información** corresponde a los *datos procesados*. En este sentido, se quiere decir que la información se puede ver como los datos organizados o resumidos. Por ejemplo, se podría querer conocer el saldo actual total de la compañía Hnos. Watabe o, quizás, se podría pedir el saldo actual promedio de todos los clientes en Europa. Las respuestas a tales preguntas serían informaciones.

información. Datos organizados o resumidos.

Por supuesto, cada hecho o elemento de los datos puede recibir el nombre de información. Pero se hace referencia, en primer lugar, a la información que sería valiosa para los distintos niveles de dirección de una empresa, incluso a nivel ejecutivo –especialmente para la toma de decisiones—. Esta información es normalmente de alto nivel, resumida a partir de un conjunto mucho más grande de hechos. Por consiguiente, la información es diferente de los datos.

En los años más recientes, el impacto significativo que ha tenido la información sobre la planificación y la toma de decisiones en las organizaciones ha conducido a un reconocimiento siempre creciente de que la información es un recurso que tiene valor y, por lo tanto, necesita estar organizada y administrada. Aun cuando en las empresas se acostumbre a trabajar con activos tangibles, tales como el dinero, las instalaciones y el personal, cuyo valor puede evaluarse con cierta precisión, ha sido muy difícil de medir el valor de la información. Sin embargo, está claro que si los directivos tienen buena información, es más probable que puedan tomar decisiones pertinentes y certeras con un mayor impacto positivo en su negocio. Y viceversa, si su información es pobre, ellos deben trabajar con más incertidumbre y es menos probable que tomen decisiones convenientes. El desarrollo de los sistemas de bases de datos se convirtió en crucial para proporcionar información correcta y oportuna a los directivos.

base de datos. Una colección de datos interrelacionados que se puede utilizar por uno o más programas de aplicación.

sistema de base de datos. Una base de datos, un sistema de gestión de bases de datos, con el *hardware* y el personal apropiados.

sistema de gestión de bases de datos (SGBD). Un sistema computacional que facilita la gestión de las bases de datos.

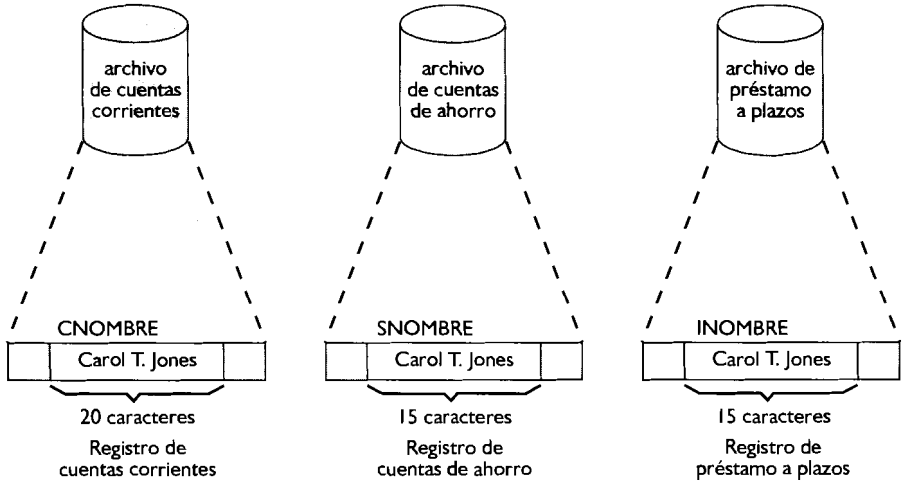
Una **base de datos** es una colección de elementos de datos interrelacionados que pueden procesarse por uno o más sistemas de aplicación. Un **sistema de base de datos** está formado por una base de datos, por un sistema computacional de propósito general —llamado **sistema de gestión de bases de datos (SGBD)**— que manipula la base de datos, así como por el *hardware* y el personal apropiados. Un SGBD normalmente se compra a un vendedor de sistemas computacionales y es el medio con el cual un programa de aplicación o un usuario final examina y manipula los datos almacenados en la base de datos. Al finalizar este capítulo se estudiarán los componentes de un sistema de base de datos con más detalle.

Un sistema de base de datos, adecuadamente diseñado, integra los datos comunes a varias unidades funcionales de la compañía y facilita su manipulación. Además de simplificar la inserción, la eliminación y la modificación cotidianas de los registros, los sistemas de base de datos facilitan la identificación y la cuantificación de las relaciones derivadas entre los elementos de los datos, la recopilación de la información en resúmenes estadísticos, la inferencia sobre las posibles tendencias del negocio y otras operaciones. Mediante tales facilidades, el sistema de base de datos transforma los datos puros en información.

Otras limitaciones de los sistemas de archivo tradicionales

A pesar de la introducción de los archivos de acceso directo, pronto se hizo obvio que a los sistemas de archivo de cualquier tipo era inherente un conjunto de deficiencias: (1) redundancia de los datos; (2) pobre control de los datos; (3) capacidades inadecuadas de manipulación de los datos, y (4) esfuerzo excesivo de programación.

Redundancia de los Datos. Una dificultad importante era que muchas aplicaciones utilizaban sus propios archivos especiales de datos. Así, algunos elementos de los datos eran comunes a varias aplicaciones. En un banco, por ejemplo, el mismo nombre de cliente podría aparecer en un archivo de cuentas corrientes, en un archivo de cuentas de ahorros y en un archivo de préstamos a plazos (Figura 1.6). Además, si bien el nombre del cliente era siempre el mismo, el campo relacionado con él frecuentemente tenía un nombre diferente en los diversos archivos de cuentas. Así, CNOMBRE en el archivo de cuentas corrientes era SNOMBRE en el archivo de cuentas de ahorros e INOMBRE en el archivo de préstamos a plazos. El mismo campo podría también tener una longitud diferente en los diversos archivos. Por ejemplo, CNOMBRE podría llegar a tener hasta 20 caracteres, pero SNOMBRE e INOMBRE podrían limitarse a 15 caracteres. Esta redundancia aumen-



tó los gastos de administración para el mantenimiento y el almacenamiento, así como el riesgo de inconsistencia entre las diversas versiones de los datos comunes.

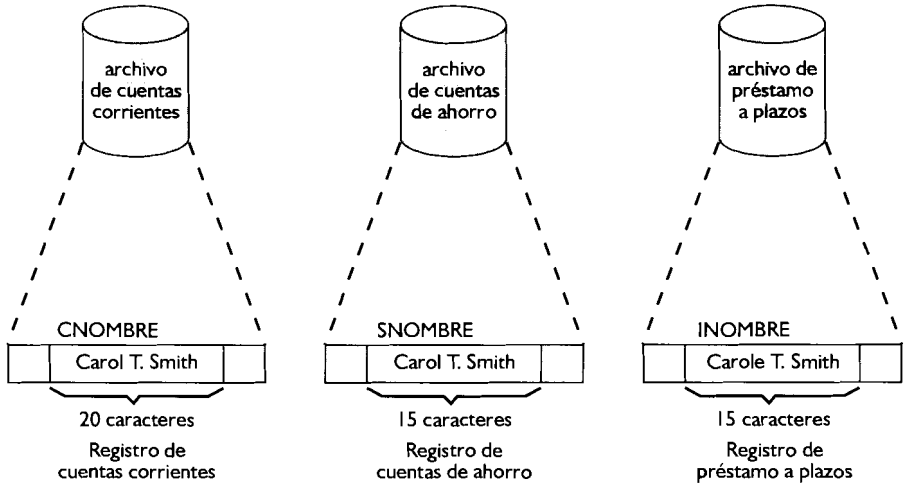
Supóngase que el nombre de un cliente se modificó de Carol T. Jones a Carol T. Smith. El campo del nombre del cliente podría actualizarse inmediatamente en el archivo de cuentas corrientes, actualizarse la semana siguiente en el archivo de cuentas de ahorros y actualizarse incorrectamente en el archivo de préstamos a plazos (Carole T. Smith) (Figura 1.7). Con el tiempo, tales diferencias pueden ocasionar una degradación seria en la calidad de la información contenida en los archivos de los datos. Estos tipos de inconsistencia de los datos pueden afectar también la exactitud de los informes. Suponga que se desea producir un informe para la gerencia mostrando todos los clientes que tienen una cuenta para cheques o una cuenta de ahorros y un préstamo a plazos. Carol T. Smith se omitiría erróneamente en el informe porque su nombre aparece como Carole T. Smith en el archivo de préstamos a plazos. Como se mostrará, los sistemas de base de datos pueden eliminar la redundancia de los datos, ya que todas las aplicaciones comparten un fondo común de datos. La información esencial como el nombre o la dirección del cliente aparecerá solamente una vez en la base de datos. De este modo, se puede introducir el cambio de un nombre o de una dirección nna sola vez y saber que las aplicaciones tendrán acceso a datos consistentes.

Pobre control de los datos. En los sistemas de archivos, como se señaló anteriormente, no había control centralizado al nivel de los elementos de los datos. El mismo elemento de los datos solía tener varios nombres, dependiendo del archivo en que estuviera contenido.

En un nivel más básico, siempre existe la posibilidad de que los diversos departamentos de una compañía sean inconsistentes en su terminología. Un banco, por ejemplo, puede utilizar el término cuenta para significar una cosa cuando se aplica a los ahorros y algo bastante diferente cuando se aplica a los préstamos. El término que tiene significados diferentes en contextos diferentes se llama un **homónimo.** Por el contrario, las palabras diferentes pueden significar la misma cosa. Una compañía de seguros puede referirse a una *póliza* y a un *caso* y puede querer significar lo mismo con ambas palabras. Dos términos que significan la misma cosa se llaman **sinónimos.** Un sistema de base de datos soporta el control centralizado de los datos y contribuye a eliminar la confusión ocasionada por los homónimos y por los sinónimos.

homónimo. Un término que tiene diferentes significados en diferentes contextos.

sinónimo. Término que significa lo mismo.



Capacidades inadecuadas de manipulación de los datos. Los archivos secuenciales indexados permitieron que las aplicaciones tuvieran acceso a un registro particular mediante una clave, como el ID del Producto. Por ejemplo, si se conocía el ID del Producto para las lámparas de mesa, se podía tener acceso directamente al registro que le corresponde dentro del archivo PRODUCTO. Esto fue suficiente mientras solamente se quiso un registro único.

Sin embargo, supongamos que se quiso un conjunto de registros interrelacionados. Por ejemplo, se podía estar interesado en identificar todas las ventas de la compañía IPD al cliente Maltzl. Quizás se necesitaba conocer el número total de ventas, o el precio promedio, o los productos que estaban siendo comprados y de cuáles fabricantes. Dicha información sería difícil, si no imposible, de obtener con un sistema de archivos porque estos sistemas son incapaces de proveer conexiones fuertes entre los datos contenidos en archivos diferentes. Los sistemas de base de datos se desarrollaron específicamente para facilitar la interrelación de los datos en archivos diferentes.

Esfuerzo de programación excesivo. Un nuevo programa de aplicación con frecuencia requería un conjunto completamente nuevo de definiciones de los archivos. Aunque un archivo existente podía contener algunos de los datos necesarios, la aplicación frecuentemente requería otros elementos de los datos. Como resultado, el programador tenía que recodificar todas las definiciones de los elementos de los datos necesarios ya existentes en el archivo, así como todos los elementos de los datos nuevos. De esta manera, en los sistemas orientados a archivos había una interdependencia muy fuerte entre los programas y los datos.

Aún más importante, la manipulación de los datos en los lenguajes orientados a archivos, como el COBOL, era difícil para las aplicaciones complejas. Esto significó que ambos esfuerzos en la programación de aplicaciones de gestión, tanto inicial como de mantenimiento, fueran significativos.

Las bases de datos brindan una separación entre el programa y los datos, de modo que los programas pueden ser, en cierta medida, independientes de los detalles de definición de los datos. Al garantizar el acceso a un fondo común de datos compartidos y al soportar lenguajes poderosos para la manipulación de los datos, los sistemas de base de datos eliminan una gran cantidad de programación inicial y de mantenimiento.

▼ Los sistemas de bases de datos

Los sistemas de bases de datos superan estas limitaciones de los sistemas orientados a los archivos. Al tolerar una estructura de datos centralizada, integrada, los sistemas de base de datos eliminan los problemas de redundancia y de control de los datos. Si una base de datos centralizada está disponible para toda la compañía y, por ejemplo, tuvo que cambiarse el nombre de un cliente, dicho cambio está disponible para todos los usuarios. Los datos se controlan por medio de un diccionario de datos/directorio (DD/D) de los datos, que en sí mismo está controlado por un grupo de empleados de la compañía, conocidos como los administradores de la base de datos (ABDs)[3]. Los nuevos métodos de acceso a los datos simplifican mucho el proceso de relacionar los elementos de los datos, a la vez que mejoran la manipulación de los datos. Todas estas características de los sistemas de base de datos simplifican el esfuerzo de programación y mantenimiento de los programas.

modelo de datos. Un método conceptual para estructurar los datos.

En la actualidad, estamos inmersos en varias décadas de largo esfuerzo por desarrollar sistemas de gestión de bases de datos cada vez más poderosos. Este proceso ha sido testigo del desarrollo evolutivo de los sistemas basados en tres **modelos de datos** fundamentales, y que no son más que métodos conceptuales para estructurar los datos. Estos tres modelos de datos son el jerárquico, en red y el relacional. En las dos secciones siguientes se esbozará el desarrollo histórico de los sistemas de base de datos, apoyado en estos tres modelos.

[3] En inglés DBAs - *DataBase Administrators* (N. del T.).

Sistemas de los modelos jerárquico y en red

Los archivos secuenciales indexados resolvieron el problema del acceso directo a un registro único dentro de un archivo. Por ejemplo, al revisar una vez más la Figura 1.3. Si tuviéramos el registro de la primera venta mostrado en el archivo VENTA, pero se quisiera conocer el nombre y la dirección del cliente involucrado en la venta, se podría utilizar simplemente el ID del Cliente (100) para buscar el registro del cliente en el archivo CLIENTE. Esto informa que el cliente que emitió la orden fue Watabe Bros.

Ahora supongamos que se quiere invertir el proceso. En lugar de querer conocer el cliente involucrado en una venta, se quieren conocer todas las ventas realizadas a un cliente determinado. Se comienza a partir del registro de cliente correspondiente a Watabe Bros, y ahora se quieren todos sus registros de venta. Esto no se puede hacer directamente en un sistema de archivo. Precisamente, para tales aplicaciones se desarrollaron originalmente los sistemas de base de datos

modelo jerárquico. Un modelo de datos que resume que todas las interrelaciones entre los datos pueden estructurarse como jerarquías.

Los primeros sistemas de base de datos, introducidos a mediados de los sesenta, estaban basados en el **modelo jerárquico,** que presume que todas las interrelaciones entre los datos pueden estructurarse como jerarqnías. Para ilustrar esto, se modifica ligeramente la base de datos de la Figura 1.3. Ahora, en lugar de ventas que sólo contienen un producto único, se tienen las facturas que a su vez contienen las líneas de factura. Cada cliente tiene varias facturas y cada factura tiene varias líneas. Cada líuea registra la venta de un producto único. La Figura 1.8 muestra algunos ejemplos. Los archivos FACTURA y LÍNEA DE FACTURA reemplazan el archivo VENTA en la Figura 1.3.

puntero. Una dirección física que identifica dónde puede encontrarse un registro sobre el disco.

La Figura 1.9 ilustra cómo se puede construir una jerarquía que muestra las interrelaciones entre los clientes, las facturas y las líneas de factura. Se considera que un cliente es el "propietario" de las facturas, las que a su vez son "propietarias" de las líneas de factura. En un sistema jerárquico de base de datos, estos tres archivos se conectan entre sí mediante punteros físicos o campos de datos añadidos a los registros individuales. Un **puntero (apuntador)** es una dirección física que identifica dónde puede encontrarse un registro sobre el disco. Cada registro de cliente contendría un apuntador al primer registro de factura correspondiente a ese registro de cliente. Los registros de factura contendrían a la vez punteros a otros registros de factura y a los registros de línea de factura. De esta manera, el sistema sería capaz de recuperar fácilmente todas las facturas y las líneas de factura que se aplican al cliente determinado.

CLIENTE

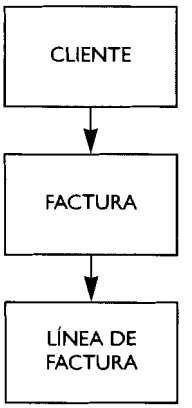
ID_CLIENTE	NOMB_CLIENTE	DIRECCIÓN	PAÍS	BALANCE INICIAL	PAGOS HASTA EL MES ACTUAL
100	Hnos. Watabe	Box 241, Tokio	Japón	45.551	40.113
101	Matlzl	Salzburg	Austria	75.314	65.200
105	Jefferson	B 918, Chicago	USA	49.333	49.811
110	Gómez	Santiago	Chile	27.400	28.414

FACTURA

FACTURA-#	FECHA	ID_CLIENTE	ID_REP
1012	02/10	100	39
1015	02/14	110	37
1020	02/20	100	14

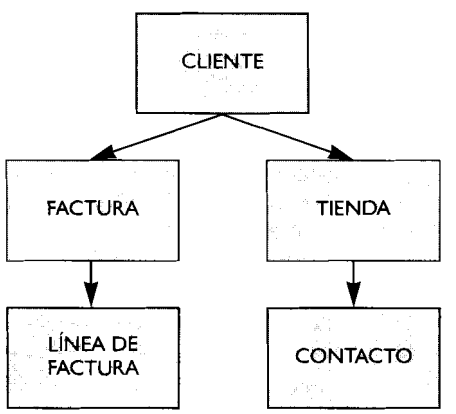
LÍNEA DE FACTURA

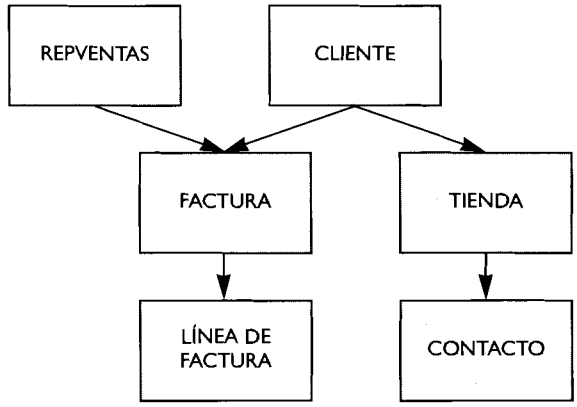
FACTURA-#	LÍNEA-#	ID_PRODUCTO	CANTIDAD	PRECIO TOTAL
1012	01	1035	100	2200,00
1012	02	2241	200	6650,00
1012	03	2518	300	6360,00
1015	01	1035	150	3300,00
1015	02	2518	200	4240,00
1020	01	2241	100	3325,00
1020	02	2518	150	3180,00



Suponga que estamos interesados en agregar informacióu sobre los clientes a la base de datos jerárquica. Por ejemplo, como los clientes son compañías de tiendas por departamentos, podríamos querer tener un listado de las tiendas para cada cliente. En ese caso, expandiríamos el diagrama de la Figura 1.9 para que se vea como el de la Figura 1.10. CLIENTE todavía está relacionado con FACTURA, la que está relacionada con LÍNEA DE FACTURA. Sin embargo, CLIENTE también se relaciona con TIENDA y TIENDA está relacionada con CONTACTO. Se llama CONTACTO a un comprador a quien vendemos la mercancía para una tienda en particular. Como se ve en este diagrama, CLIENTE está en la raíz de una jerarquía de la cual puede derivarse una gran cantidad de información.

Estas figuras muestran el tipo de interrelaciones entre archivos que pueden realizarse fácilmente utilizando el modelo jerárquico. Sin embargo, se comprobó muy rápidamente que este modelo tenía algunas limitaciones importantes, ya que no todas las interrelaciones podrían expresarse fácilmente en una estructura jerárquica. Por ejemplo, para tomar el caso actual un paso más allá, es obvio que nos interesamos no solamente por la relación entre los clientes y las facturas, sino que también nos interesa la relación entre las facturas y los representantes de ventas. Es decir, queremos conocer todas las facturas que ha producido un representante de ventas determinado, de modo que se puedan emitir las instrucciones para su comisión. Esta nueva interrelación se muestra en la Figura 1.11.





hijo. Un registro "subordinado" en una interrelación jerárquica.

padre. Un registro "propietario" en una interrelación jerárquica.

red. Una interrelación de datos en la cual un registro puede estar subordinado a registros de más de un archivo.

Sin embargo, este diagrama no es una jerarquía. En una jerarquía, un **hijo** puede tener solamente un **padre**. En la Figura 1.10, FACTURA es un hijo y CLIENTE es su padre. No obstante, en la Figura 1.11 FACTURA tiene dos padres —REPVENTAS y CLIENTE—. Estas redes se denominan diagramas. A causa de la necesidad obvia de manipular tales interrelaciones, a finales de los años sesenta se desarrollaron los sistemas de base de datos *en red*. Al igual que los sistemas de base de datos jerárquicos, los sistemas de base de datos en red emplearon punteros físicos para enlazar entre sí los registros de diferentes archivos.

El SGBD jerárquico dominante es el IMS de IBM, desarrollado a mediados de los sesenta. Entre mediados de los sesenta y principios de los setenta se desarrollaron y se comercializaron exitosamente varios SGBD en redes y este modelo de datos se normalizó, eventualmente, como el modelo CODASYL. En capítulos posteriores se estudiarán extensamente ambos modelos de datos y las facilidades que brindan para la definición y la manipulación de los datos.

Sistemas de bases de datos relacionales

El uso de punteros físicos era simultáneamente una fortaleza y una debilidad de los sistemas de bases de datos jerárquicos y en red. Los punteros eran una fortaleza porque permitieron la recuperación rápida de los datos que tuvieran interrelaciones predeterminadas. La debilidad estaba en el hecho de que estas interrelaciones tenían que definirse *antes* de que el sistema fuera puesto en explotación. Era difícil, si no imposible, recuperar datos basados en otras interrelaciones. En la medida en que los usuarios se familiarizaron con los sistemas de base de datos y con su potencia para manipular los datos, rápidamente encontraron estas limitaciones inaceptables, como lo muestra esta polémica en la compañía IPD.

"Cordelia, estamos cada vez más frustrados con la cantidad de preguntas que nuestro sistema de base de datos no puede responder fácilmente. Cuando Sandy y tú nos convencisteis de que debíamos cambiar hacia un sistema de base de datos en red, argumentaste que tendríamos la posibilidad de obtener respuestas a la mayoría de las preguntas que fuéramos capaces de formular."

Susan Broadbent, presidenta ejecutiva de la compañía IPD, y Dick Greenberg, gerente de ventas de IPD, están hablando con la administradora de la base de datos de

IPD (ABD), Cordelia Molini, sobre los defectos de su actual sistema de gestión de bases de datos en red. Esto no es una queja nueva.

Con cada nuevo sistema, los directivos de IPD se percataban de que comenzaban a hacer preguntas que el sistema no respondía como les hubiera gustado. Aunque el sistema en red les sirvió durante varios años, ahora estaba llegando al momento en el que cada vez es más y más difícil obtener la información que se necesita. Cordelia conocía bien estas frustraciones.

"Bueno, tal vez nosotros debimos decir muchas *preguntas que les gustaría hacer. La experiencia nos ha demostrado que existe una variedad muy extensa de preguntas que los usuarios quisieran formular." Cordelia continúa: "El problema es que el sistema en red depende de punteros físicos que conectan entre sí los datos en diferentes archivos. Si haces una pregunta que no corresponde naturalmente con esos punteros, no podemos responder a ella sin una cantidad considerable de programación. Ahora dime, ¿puedes darme una idea mejor de los tipos de preguntas para las cuales necesitas respuestas?"*

Dick respondió inmediatamente: "Cordelia, queremos responder a todo tipo de preguntas. Realmente no podemos caracterizarlas por tipo, pues esto sería muy restringido. Idealmente, nos gustaría tener la posibilidad de formular cualquier pregunta que quisiéramos y, si la respuesta está en los datos, que el sistema sea capaz de darla."

"Lo que tú dices indica que necesitamos considerar seriamente el cambio hacia un sistema relacional de bases de datos. Los punteros físicos no se usan en un sistema relacional. Los datos se pueden relacionar siempre que exista una conexión lógica, *luego no tenemos que preocuparnos por definir cuáles interrelaciones son las que el sistema utiliza con mayor probabilidad."*

Susan pregunta: "¿Significa esto, por ejemplo, que Dick puede preguntar si los productos hechos en Ghana se venden bien en Corea? O ¿con qué éxito un representante de ventas en Río está vendiendo los equipos electrónicos de Amsterdam?"

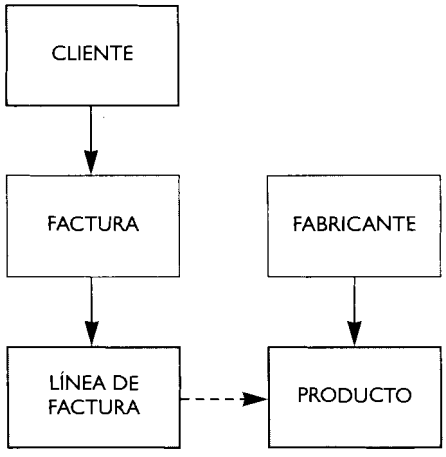
"Sí. Ambas preguntas son fáciles de responder en un sistema relacional. Además, te percatarás de que no tendrás que trabajar cerca de un programador tan a menudo. Los directivos que están deseosos de aprender un lenguaje de manipulación de datos relativamente simple pueden responder a muchas de sus propias preguntas haciéndolas directamente al sistema. Susan, ¿tú qué piensas?"

"Esto suena como algo que merece una investigación adicional. Si la tecnología es tentadora, deberíamos pensar seriamente en movernos hacia ella. ¿Por qué no profundizas en esto y simultáneamente haces una propuesta?"

En 1970, E. F. Codd publicó un artículo revolucionario (Codd, 1970) que desafió fuertemente el juicio convencional de la "condición" de las bases de datos. Codd argumentó que los datos deberían relacionarse mediante interrelaciones naturales, lógicas, inherentes a los datos, más que mediante punteros físicos. Es decir, las personas debían ser capaces de combinar los datos de fuentes diferentes, si la información lógica necesaria para hacer la combinación estaba presente en los datos. Esto abrió una nueva perspectiva para los sistemas de gestión de información, ya que las interrogaciones a las bases de datos no necesitarían, en adelante, limitarse a las interrelaciones indicadas por los punteros físicos.

Para ilustrar las deficiencias de los sistemas de bases de datos que se fundamentan en punteros físicos, considérese la Figura 1.12. En ella se muestra que CLIENTE, FACTURA y LÍNEA DE FACTURA están conectados por punteros físicos. FABRICANTE y PRODUCTO también están conectados. La línea punteada entre PRODUCTO y LÍNEA DE FACTURA indica que están relacionados *lógicamente*, ya que cada línea de factura se refiere a un producto específico.

Sin embargo, suponga que PRODUCTO no ha sido conectado con LÍNEA DE FACTURA mediante un puntero físico. ¿Cómo se puede obtener el informe siguiente?



Para cada cliente, identificar los fabricantes cuyos productos ha ordenado el cliente.

Esto requiere navegar desde CLIENTE a través de FACTURA y LÍNEA DE FACTURA, hasta PRODUCTO y, de ahí, subir hasta FABRICANTE. Puesto que la conexión física entre LÍNEA DE FACTURA y PRODUCTO no existe en la base de datos, esta navegación no se puede hacer mediante las facilidades normales de la bases de datos. En su lugar, para obtener la información pedida, deben utilizarse las técnicas antiguas y engorrosas del procesamiento de archivos. Ello necesitaría una cantidad considerable de compleja programación. En contraste con esto, los sistemas de base de datos que soportan la recuperación de los datos, tomando en consideración las interrelaciones *lógicas*, podrían resolver fácilmente tales problemas.

En su artículo, Codd propuso un modelo simple de datos en el que todos ellos se representarían en tablas constituidas por filas y columnas. A estas tablas se les dio el nombre matemático de *relaciones*, y por eso el modelo se denominó modelo relacional. Codd también propuso dos lenguajes para manipular los datos en las tablas: el álgebra relacional y el cálculo relacional (se estudiarán en el Capítulo 6). Ambos lenguajes soportan la manipulación de los datos sobre la base de operadores lógicos en lugar de los punteros físicos utilizados en los modelos jerárquico y en red.

Al manipular los datos sobre una base conceptual en vez de una base física, Codd introdujo otra innovación revolucionaria. En los sistemas de base de datos relacionales, los archivos completos de datos se pueden procesar con instrucciones sencillas. Sin embargo, los sistemas tradicionales requieren que los datos se procesen de registro en registro. El enfoque de Codd mejoró enormemente la eficiencia conceptual de la programación de la base de datos.

La manipulación lógica de los datos también hace factible la creación de lenguajes de interrogación más accesibles al usuario no especialista en computación. Aunque es bastante difícil crear un lenguaje que pueda ser utilizado por *todas* las personas sin considerar su experiencia previa en computación, los lenguajes relacionales de consulta hacen posible el acceso a las bases de datos para un grupo de usuarios cada vez mayor.

La publicación de los artículos de Codd, a principios de los años setenta, provocó una conmoción en la actividad de las comunidades de desarrollo de sistemas de investigación y de sistemas comerciales, en la medida en que trabajaban para producir un sistema de gestión de bases de datos relacional. El resultado fue la aparición de sistemas relacionales durante la última mitad de los setenta que soportaban lenguajes como el Structured Query Language (SQL), el Query Language (Quel) y el Query-by-Example (QBE). A

medida que las computadoras personales se hicieron populares durante los años ochenta, los sistemas relacionales para ellas también estuvieron disponibles. En 1986, el SQL se adoptó como la norma ANSI para los lenguajes relacionales de bases de datos. Esta norma se actualizó en 1989 y en 1992. En el Capítulo 7 se discuten diferentes aspectos del SQL-92.

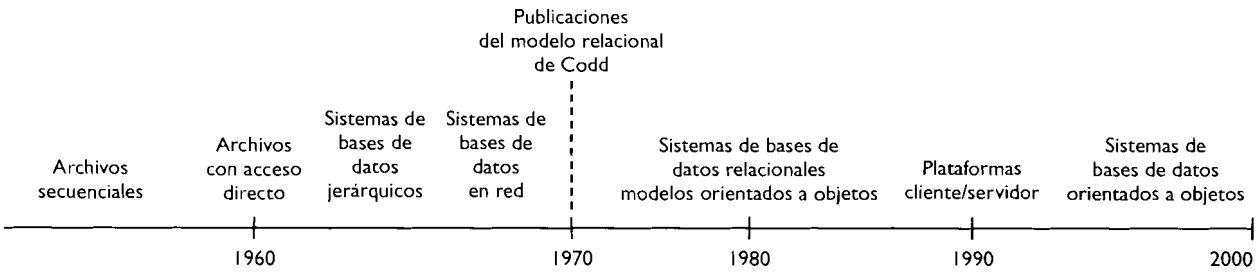
Todos estos desarrollos hicieron avanzar enormemente el estado del arte en los sistemas de gestión de bases de datos y aumentaron la disponibilidad de información en las bases de datos colectivas. El enfoque relacional ha resultado bastante ventajoso. Además, han continuado las promesas en cuanto a los resultados de las investigaciones con vistas a garantizar capacidades cada vez más poderosas a medida que se alcanza una comprensión más completa de las necesidades del usuario con respecto a los sistemas de bases de datos.

Actualmente, los sistemas relacionales son un estándar en el mercado, especialmente en operaciones comerciales. Naturalmente, tanto los sistemas orientados a archivos, como también los sistemas de base de datos jerárquicos y en redes, son todavía abundantes y, para ciertas aplicaciones, constituyen la solución más eficiente en función de los costos. Sin embargo, durante algún tiempo, la tendencia clara de las compañías ha sido migrar a los sistemas relacionales siempre que fuera posible.

Aun así, sería un error asumir que los sistemas de bases de datos relacionales, ahora disponibles, representan la última palabra en el desarrollo de los SGBD. Los sistemas relacionales de hoy aún están evolucionando y, en algunos aspectos significativos, cambiando su naturaleza subyacente para permitir a los usuarios plantear problemas más complejos. Desde nuestro punto de vista, uno de los cambios más importantes está ocurriendo en el área de las bases de datos orientadas a objetos. Eu el Capítulo 4 se estudiará un modelo conceptual de datos que contiene aspectos característicos utilizados en las bases de datos orientadas a objetos. Un desarrollo adicional de gran importancia es la aparición de la plataforma cliente/servidor como la base para los cálculos y el acceso a las bases de datos en una organización. En la sección siguiente se presenta una panorámica de este concepto.

La Figura 1.13 muestra la cronología del desarrollo histórico de los métodos de acceso a los datos, que ya se ha discutido. La Tabla 1.1 muestra una comparación entre algunas características de los diversos métodos de acceso a los datos.

Método de acceso a los datos	*Características*
Archivos secuenciales	Todos los registros en un archivo deben procesarse en secuencia
Archivos con acceso directo	Soportan el acceso directo a un registro específico. Es difícil el acceso a varios registros relacionados con un registro simple
Bases de datos jerárquicas	Soportan el acceso a varios registros relacionados con un registro simple Se restringen a las interrelaciones jerárquicas entre los datos Dependientes de punteros físicos predefinidos
Bases de datos en red	Soportan las interrelaciones jerárquicas y no-jerárquicas en redes entre los datos Dependientes de punteros físicos predefinidos
Bases de datos relacionales	Soportan todas las interrelaciones lógicas entre los datos El acceso a los datos es lógico, independiente de las técnicas de instrumentación física



▼ Direcciones actuales-plataformas cliente/servidor

"De acuerdo, Sandy, explícame por qué piensas que nos ayudaría hacer la conversión a una plataforma cliente/servidor. De cualquier modo, ¿qué es cliente/servidor?"

Susan y Sanford continúan la discusión tecnológica en que los encontramos comenzando este capítulo. La compañía IPD ha progresado mediante sistemas de archivos y de bases de datos y ahora está considerando el siguiente paso. Naturalmente, antes de hacer un movimiento ellos quieren saber hacia dónde van y si esto será eficiente en función de los costos.

"Una plataforma cliente/servidor *normalmente es una red local de área —LAN*[4]*— a la que se enlaza un conjunto de computadores personales y que contiene un computador especial que llamamos un servidor. Los computadores personales son los clientes, que solicitan al servidor diferentes servicios."*

"¿Por ejemplo?"

"El servidor les puede enviar programas tales como procesadores de textos u hojas de cálculo o puede ejecutar las consultas a la base de datos y enviarles los resultados. La idea básica es que cada máquina ejecute lo que haga mejor. El servidor recupera y actualiza los datos, el cliente realiza los cálculos especiales y presenta los datos al usuario.

"¿Esto será barato?"

"No necesariamente."

"¿Entonces, qué ventajas ofrece?"

"Nuestros sistemas serán mucho más potentes y fáciles de utilizar. Cada usuario final tendrá para trabajar una Interfaz gráfica de usuario[5]*. Asimismo, como hay un computador sobre cada escritorio, los sistemas tendrán muchas funciones adicionales. Seremos capaces de aumentar también las capacidades con mayor facilidad y a menor costo, simplemente, incrementando el número de máquinas. Y estaremos en una posición más flexible para sacar ventaja del hardware y del software nuevos."*

"Hasta aquí suena bien. Conforma un grupo para estudiar las ventajas y los riesgos. Si tu informe es favorable, podemos seguir adelante."

La introducción del PC IBM en 1981 estableció la estación de trabajo personal como una norma en la oficina. El tratamiento de textos, las hojas de cálculo y otros *software* justificaron por sí solos el uso de estas máquinas. Además, para ellos era natural estar enla-

[4] LAN, *Local Area Network* (N. del T.).

[5] Interfaz Gráfica de Usuario, *Graphical User Interface (GUI)* (N. del T.).

plataforma cliente/servidor. Una red local que consiste en computadores clientes, que reciben servicios de un computador servidor.

servidor de bases de datos. Un programa que corre en un computador servidor para dar servicios de bases de datos a las máquinas clientes.

interfaz Gráfica de Usuario (GUI). Pantallas y funciones que proporcionan al usuario final un medio gráfico para tener acceso al sistema de computación.

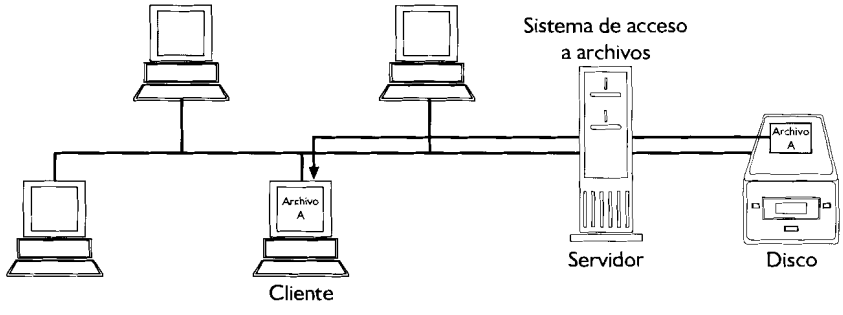
sistemas abiertos. El concepto de conectar una variedad de computadores con diferentes *hardware* y *software* para trabajar coordinadamente con el fin de lograr los objetivos del usuario.

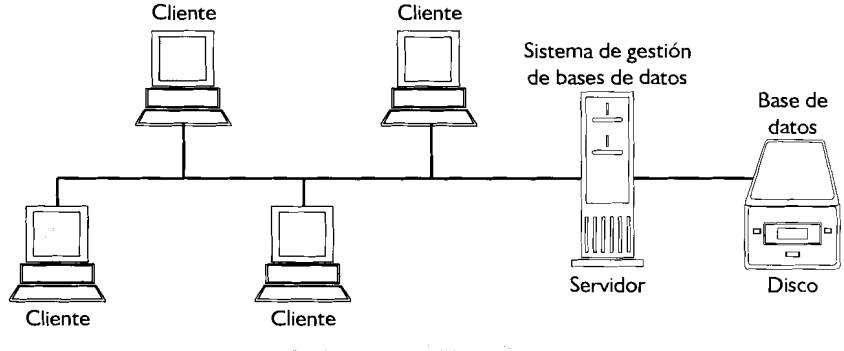
interoperabilidad. El estado que caracteriza a múltiples sistemas heterogéneos que se comunican y contribuyen a la terminación de una tarea común

zados entre sí para permitirle a los usuarios comunicarse por medio del correo electrónico y tener acceso a los recursos comunes, tales como impresoras y discos. Inicialmente los servidores se instalaron para controlar la impresión y el acceso a los archivos. Ellos constituyeron los *servidores de impresión* y los *servidores de archivos*. Por ejemplo, el servidor de archivos, al responder a la solicitud de un cliente sobre un archivo específico, enviaría el archivo completo a través de la red al computador cliente (Figura 1.14). Hoy, sin embargo, la mayoría de los servidores son **servidores de bases de datos** —programas que se ejecutan en el *hardware* servidor y proporcionan los servicios de las bases de datos a los computadores clientes (Figura 1.15)—. Así, un cliente que está ejecutando un proceso de una aplicación y necesita una interrogación (consulta) a una base de datos, envía la petición al servidor de la base de datos y éste le devolverá los datos solicitados. El programa de aplicación puede enviar también datos al servidor con la solicitud de actualizar la base de datos. El servidor efectuaría esta actualización.

La potencia de la plataforma cliente/servidor descansa en el concepto de división de funciones. El cliente es el computador frontal que tiene interfaz directamente con el usuario. Manipula la **Interfaz Gráfica de Usuario (GUI)** y realiza los cálculos y otros programas de interés para el usuario final. El servidor es el que gestiona fundamentalmente el acceso a los datos atendiendo las diferentes peticiones de los clientes, también realiza funciones de control y seguridad de acceso a los datos.

Conceptualmente, las plataformas cliente/servidor son parte del concepto de **sistemas abiertos**, en el cual todo tipo de computadores, sistemas operativos, protocolos de redes y otros *software* y *hardware* pueden interconectarse y trabajar coordinadamente para lograr los objetivos del usuario. Sin embargo, en la práctica, los problemas de alcanzar tal variedad de sistemas operativos, protocolos de redes, sistemas de base de datos y otros, que trabajen conjuntamente pueden ser en extremo peligrosos. El objetivo de los sistemas abiertos consiste en lograr la **interoperabilidad**, que es el estado de dos o más siste-





mas heterogéneos comunicándose y contribuyendo cada uno a alguna parte del trabajo que corresponde a una tarea común.

En cierto sentido, el enfoque cliente/servidor es la culminación de una percepción temprana de la potencia del cálculo distribuida conjuntamente con el control de y el acceso a los datos inherentes a un computador centralizado. Los visionarios anunciaron la inminencia de la computación distribuida hace más de dos décadas, pero sólo recientemente esto se convirtió en una realidad. Aunque muchos problemas difíciles de la interoperabilidad están por resolverse, las perspectivas de aumentar continuamente la potencia de cálculo y el acceso a las bases de datos al nivel del usuario final nunca han sido más estimulantes.

▼ Los sistemas de base de datos: el hardware, el software, los datos, las personas

Hasta aquí, se han discutido, en términos generales, los sistemas de base de datos y sus capacidades. Ahora es el momento de acercarse a los componentes que constituyen tal tipo de sistema. Un sistema de base de datos es algo más que simples datos o que los datos en combinación con un software de gestión de bases de datos. En una organización, un sistema de base de datos *completo* está compuesto por cuatro componentes: el hardware, el software, los datos y las personas.

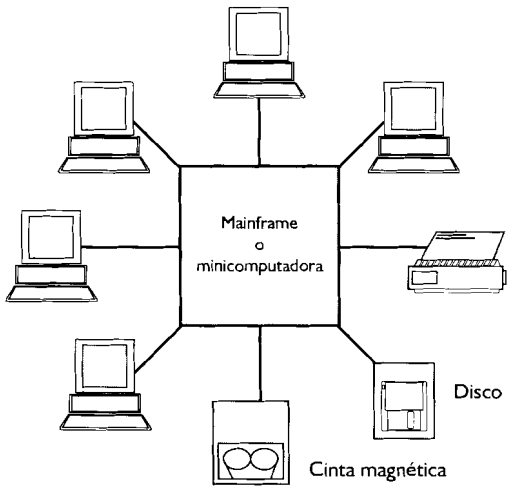
El hardware

El hardware es el conjunto de dispositivos físicos sobre los que reside una base de datos. Consiste en una o más computadoras, unidades de disco, vídeo-terminales, impresoras, unidades de cinta magnética, cables de conexión y otros equipos auxiliares y de conexión del equipamiento.

Las computadoras, utilizados para procesar los datos en la base de datos, pueden ser un *mainframe*, minicomputadoras o computadoras personales. En el ejemplo dado con anterioridad, la compañía IPD inicialmente comenzó el procesamiento con una minicomputadora y luego escaló a un *mainframe*. El *mainframe* y las minicomputadoras se han utilizado tradicionalmente de forma autónoma para soportar el acceso de varios usuarios a una base de datos común. Las computadoras personales se utilizan frecuentemente con bases de datos autónomas controladas y manipuladas por un usuario único. No obstante, también pueden conectarse a una red cliente/servidor, garantizando el acceso de varios usuarios a una base de datos común almacenada sobre unidades de disco y controlada por un computador servidor. El servidor en sí mismo puede ser una computadora personal más potente, una minicomputadora, un *mainframe* o una computadora más potente con multiprocesador. La Figura 1.16 ilustra una variedad de configuraciones de hardware.

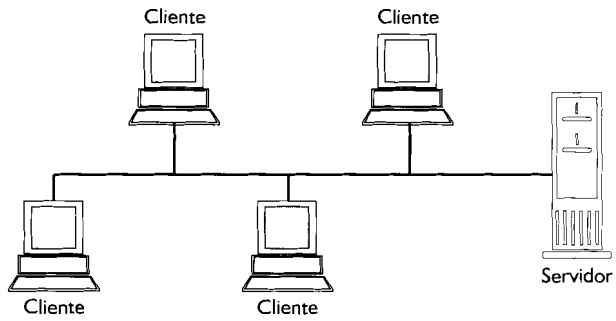
Las unidades de disco constituyen el mecanismo de almacenamiento principal para las bases de datos y son esenciales, pues ellas permiten el acceso directo, sin el cual el procesamiento de las bases de datos sería imposible. Las computadoras personales, los vídeo-terminales y las impresoras se utilizan para introducir y recuperar información de las bases de datos. Las unidades de cinta magnética garantizan un respaldo barato y rápido de los datos que están almacenados en las unidades de disco.

El éxito de los sistemas de base de datos ha dependido fuertemente de los adelantos en la tecnología del hardware. Para mantener y controlar la cantidad enorme de datos almacenados en una base de datos se requiere una memoria principal y un espacio de almacenamiento en disco muy grandes. Adicionalmente, se necesitan computadoras rápidas, redes y periféricos para ejecutar el alto número de accesos requerido para recuperar la iuformación en un tiempo aceptable en un ambiente que tenga una cantidad grande de usuarios. Afortunadamente, el hardware ha sido cada vez más potente y más barato durante los años de desarrollo de la tecnología de las bases de datos. Esto ha hecho posible el uso difundido de los sistemas de bases de datos.



(a) Mainframe o minicomputadora que permite el acceso desde varias terminales

(b) Computadora personal que usa una base de datos almacenada



(c) Computadoras personales conectadas en una red cliente/servidor, todas las cuales tienen acceso a la base de datos almacenada en un servidor

El software

Un sistema de base de datos incluye dos tipos de software:

- El software de propósito general para la gestión de bases de datos, comúnmente llamado sistema de gestión de bases de datos (SGBD) (en inglés, **DBMS**).
- El software de aplicación, que usa las facilidades del SGBD para manipular la bases de datos con el fin de llevar a cabo una función específica de la compañía, tal como la emisión de los estados o el análisis de las tendencias de las ventas.

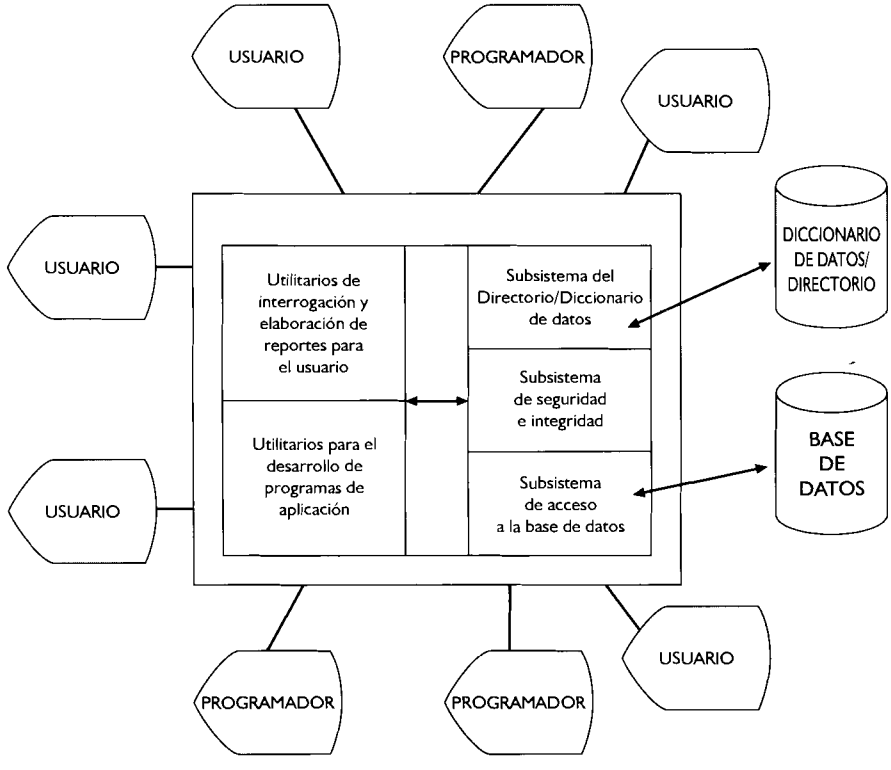
El software de aplicación generalmente se escribe por los empleados de una compañía para resolver un problema específico. Puede estar escrito en un lenguaje de programación estándar, tal como COBOL o C, o puede escribirse en un lenguaje (usualmente llamado lenguaje de cuarta generación) suministrado con el sistema de gestión de bases de

datos. El software de aplicación usa las facilidades del SGBD para el acceso a y la manipulación de los datos en la base de datos, proporcionando los informes o los documentos necesarios para los requisitos de información y de procesamiento de la compañía.

El *sistema de gestión de bases de datos* (SGBD) es un *software*, parecido a un sistema operativo o a un compilador, que brinda un conjunto de servicios a los usuarios finales, los programadores y otros (Figura 1.17). Como su nombre indica, el SGBD existe para facilitar la gestión de una base de datos. Con este fin, un SGBD típicamente brinda la mayoría de los servicios siguientes:

- Herramienta para la definición y el control centralizados de los datos, conocida como diccionario de datos/directorio (DD/ D) o catálogo
- Mecanismos de seguridad e integridad de los datos
- Acceso concurrente a los datos para varios usuarios
- Utilidades para la consulta, la manipulación y la elaboración de informes orientados al usuario
- Utilidades para el desarrollo de sistemas de aplicación orientados al programador

Diccionario de datos/Directorio (DD/D). El subsistema del diccionario/directorio de datos almacena las definiciones de todos los elementos de los datos en la base de datos. Esto incluye el nivel primario de los elementos de los datos (campos), las estructuras de los datos a nivel de grupo y a nivel de registro y los archivos o tablas relacionales. El DD/D no sólo mantiene esta información, sino que, además, almacena las interrelaciones que existen entre las diversas estructuras de datos. Adicionalmente, mantiene los índices que se utilizan para garantizar el acceso a los datos rápidamente. Asimismo, almacena las definiciones de los formatos de las pantallas y de los informes, que pueden utilizarse por varios programas de aplicación.



El diccionario de datos puede verse como una parte de la base de datos en sí misma. De esta manera, la base de datos es autodescriptiva, pues contiene información que describe su propia estructura. La información en el diccionario de datos se llama **metadatos** o "datos sobre los datos." Los metadatos están disponibles para la interrogación y la manipulación, como lo están los otros datos en la base de datos.

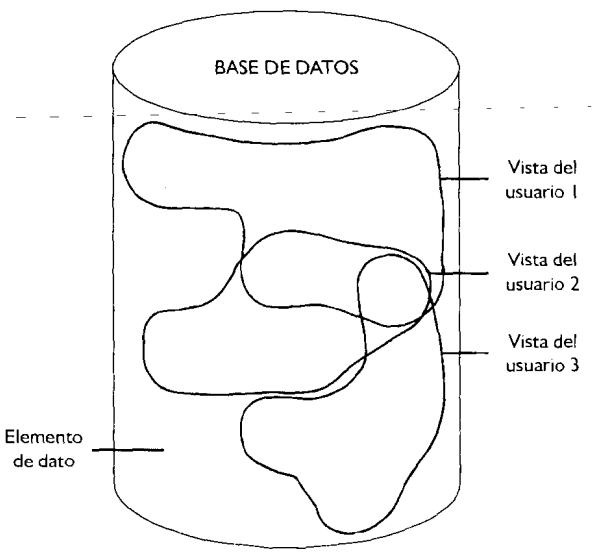
metadatos. Datos en el diccionario de datos que describen la base de datos.

Mecanismos de seguridad e integridad de los datos. La base de datos es un recurso valioso que necesita protección. El SGBD garantiza la seguridad de la base de datos limitando el acceso a la misma al personal autorizado. Los usuarios autorizados, generalmente, estarán restringidos en cuanto al acceso a ciertos datos en particular y a aquéllos que puedan actualizar. Tal acceso se controla frecuentemente mediante contraseñas y mediante las **vistas de los datos**, que constituyen definiciones de porciones restringidas de la base de datos, como se ilustra en la Figura 1.18. La integridad y la consistencia de la base de datos se protegen por medio de restricciones sobre los valores que pueden tomar los elementos de los datos y por las capacidades de recuperación y de respaldo suministradas por el SGBD. Las definiciones de las restricciones de los datos se almacenan en el diccionario de datos. La recuperación y el respaldo se soportan por el software, el que automáticamente almacena los cambios de la base de datos en el catálogo y brinda los medios para restablecer el estado actual de la base de datos en el caso de un fallo del sistema.

vista de los datos. Una definición de una porción restringida de la base de datos; también llamada una vista.

Acceso concurrente a los datos por varios usuarios. Una de las funciones principales del SGBD es garantizar el acceso, la recuperación y la actualización de los datos en la base de datos. El SGBD proporciona los mecanismos físicos que permiten a varios usuarios tener acceso de forma rápida y eficiente a diferentes datos relacionados. Esta posibilidad se extiende a usuarios localizados remotamente, quienes tendrán acceso a la base de datos mediante un sistema de telecomunicaciones. Las utilidades del SGBD brindan una interfaz con los sistemas de telecomunicaciones, de modo que las solicitudes de datos y las respuestas resultantes se enrutan adecuadamente.

La centralización de los datos en una base de datos aumenta la probabilidad de que dos o más usuarios quieran tener acceso a los mismos datos concurrentemente. Si el SGBD permite esto, los dos usuarios seguramente afectarían el trabajo del otro y podrían dañarlo. Así, es importante que el SGBD proteja los datos, a los que tiene acceso un usuario, de la actualización simultánea por otro usuario. Para hacer esto, el SGBD utiliza mecanismos



sofisticados de bloqueo para proteger los datos que se están actualizando por un usuario, mientras al mismo tiempo garantiza el acceso concurrente a la base de datos y un tiempo de respuesta del sistema a otros usuarios aceptable.

Consultas e informes dirigidos al usuario. Uno de los aspectos más valiosos de un SGBD es que brinda herramientas de manipulación de los datos dirigidas al usuario. Estos lenguajes de interrogación fáciles de usar permiten a los usuarios formular sus *consultas* y pedir informes únicos directamente de la base de datos. Esto releva al personal de programación de la compañía de la carga de formular estas *consultas* o de escribir paquetes de programas de aplicación de propósito especial.

Los generadores de informes están asociados a los lenguajes de *consulta*. Frecuentemente el lenguaje de *consulta* contendrá facilidades para estructurar los resultados de las *consultas* como informes. La *consulta* formulada puede almacenarse para su uso ulterior y sus resultados pueden producirse en forma de un informe ordinario. Cuando este es el caso, el lenguaje de *consulta* puede considerarse como un generador de informes. Además, los generadores de informes pueden también brindar facilidades mucho más poderosas que las disponibles en el lenguaje de *consulta*.

Facilidades para el desarrollo de aplicaciones. Aparte de facilitar al usuario el acceso a la base de datos para obtener información, el SGBD también garantiza una asistencia importante al programador de aplicaciones. Tales herramientas, como los generadores de pantallas, de menús y de reportes; los generadores de aplicaciones; los compiladores, y las facilidades para la definición de los datos y las vistas de los datos son estándares. Más importantes aún, los sistemas de base de datos modernos brindan componentes del lenguaje mucho más poderosos que los de los lenguajes tradicionales, lo que contribuye a que el proceso de programación sea apreciablemente más eficiente.

Los datos

Obviamente, ningún sistema de base de datos puede existir sin los datos, los hechos básicos sobre los que se fundamentan las necesidades de información y de procesamiento de una compañía. Sin embargo, el factor esencial a considerar es que los datos que conforman una base de datos tienen que ser cuidadosa y lógicamente estructurados. Las funciones del negocio deben analizarse, los elementos de los datos y las interrelaciones deben identificarse y definirse cabalmente y estas definiciones deben almacenarse de manera precisa en el diccionario de datos. Entonces, los datos pueden obtenerse e introducirse en la base de datos según la estructura definida. Una base de datos construida en armonía con estos procedimientos puede ser un recurso poderoso para garantizar información oportuna a la organización.

usuarios. Personas que necesitan información de la base de datos para desarrollar su responsabilidad primaria en el negocio.

profesionales de la computación. Personas responsables del sistema de base de datos y del paquete de programas de aplicación asociado.

procedimiento. Instrucciones escritas que describen los pasos necesarios para realizar una tarea determinada en un sistema.

Las personas

El caso de la compañía IPD permite identificar dos tipos diferentes de personas a las que concierne la base de datos. Susan Broadbent y Dick Greenberg son los **usuarios**, las personas que necesitan la información de la base de datos para desarrollar su responsabilidad primaria en el negocio, que en sí misma se encuentra en cierta área funcional diferente. Por el contrario, Sanford Mallon y Cordelia Molini son **profesionales de la computación**, personas cuya responsabilidad primaria en el negocio está en el diseño y el mantenimiento del sistema de base de datos y su paquete de programas de aplicación asociado para el beneficio de los usuarios. Ejemplos de personas en cada una de estas categorías pueden ser las siguientes:

Usuarios: Los ejecutivos, los gerentes, los administradores, el personal de oficina

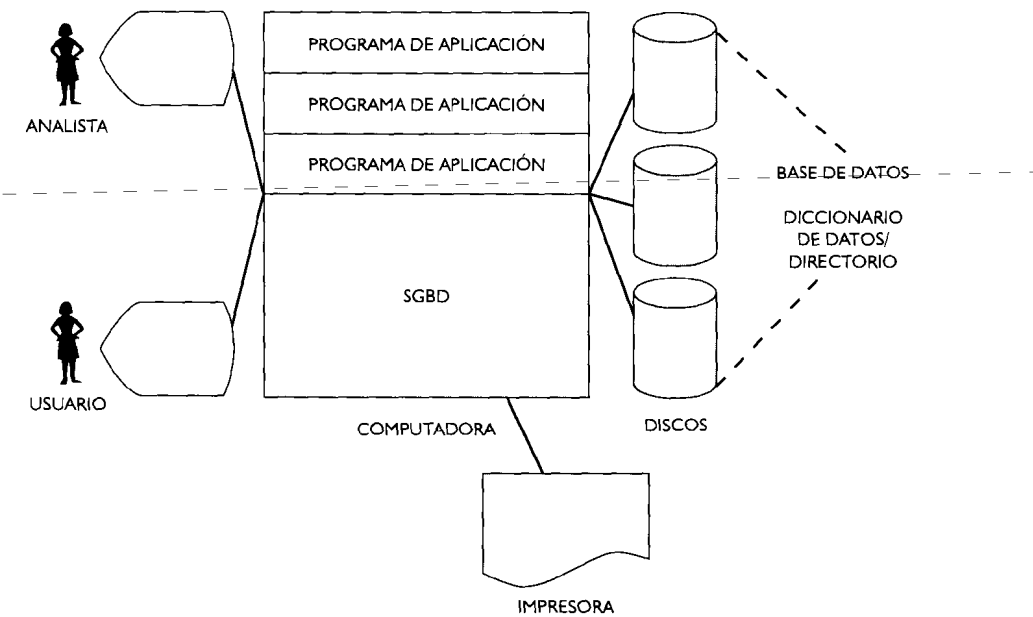
Profesionales de la computación: Los administradores de la base de datos, los analistas, los programadores, los diseñadores del sistema y de la base de datos, los administradores de los sistemas de información

Las personas encargadas de los **procedimientos** usados para lograr las metas del sistema constituyen una parte importante de este componente. Virtualmente, ningún sistema

automatiza por completo una tarea del usuario. Es necesario desarrollar procedimientos manuales para brindar una interfaz suave entre los usuarios del sistema y el sistema en sí mismo. Un ejemplo de procedimiento sería el control de auditoría por parte de los usuarios con respecto a que la cantidad total de dinero depositada en el banco correspondiente a un día determinado concuerde con la cantidad total de efectivo recibido mostrada por el sistema ese mismo día. Normalmente, existen muchos procedimientos en un sistema y el éxito del sistema, con frecuencia, depende mucho de la habilidad con que tales procedimientos se desarrollen para articular con las funciones del sistema, así como de la estructura del sistema en sí mismo.

Interrelación (relación) entre los cuatro componentes del sistema

La figura 1.19 resume la interrelación entre los cuatro componentes de un sistema de base de datos. Los *profesionales de la computación* (los diseñadores de base de datos y analistas) en consulta con los *usuarios* identifican las necesidades de *datos* y diseñan las estructuras de la base de datos para responder a estas necesidades. Entonces, las estructuras de la base de datos se especifican al SGBD mediante el diccionario de datos. Los *usuarios introducen los datos* en el sistema siguiendo *procedimientos* específicos. Los datos introducidos se almacenan en los medios de hardware, tales como discos y cintas. Los *programas de aplicación* que tienen acceso a la base de datos se escriben por los profesionales de la computación y por los usuarios para ejecutarlos sobre los *computadores*. Estos programas utilizan el lenguaje de órdenes del SGBD y hacen uso de la información contenida en el diccionario de datos. Estos programas generan información, que puede utilizarse por los ejecutivos y los gerentes para tomar decisiones en el negocio. Los programas de aplicación también pueden generar las notificaciones de facturas y otros documentos utilizados por los clientes del negocio. De este modo, puede verse que en un sistema adecuadamente diseñado y en funcionamiento, sus cuatro componentes —el hardware, el software, los datos y las personas— conforman un sistema único con el fin de alcanzar las metas de la organización.



En este capítulo se ha revisado el desarrollo de la tecnología de las bases de datos, comenzando con los métodos primarios de acceso a archivos y continuando con los enfoques principales en el procesamiento de las bases de datos. También se han identificado y discutido los cuatro componentes principales de un sistema de base de datos: el hardware, el software, los datos y las personas.

Al comienzo del proceso de datos, durante los cincuenta y el comienzo de los sesenta, la regla era el tratamiento de archivos secuenciales. Todos los datos se almacenaban en archivos secuenciales, que exigían el tratamiento de archivos completos por los programas de aplicación. Durante los sesenta, debido a que el almacenamiento en disco utilizando el acceso directo llegó a estar ampliamente disponible, el procesamiento de archivos de acceso aleatorio llegó a ser factible y popular. Este método permitió el acceso directo a datos específicos en un archivo.

En la medida en que los sistemas computacionales de procesamiento de datos se hicieron más importantes, los negocios comenzaron a reconocer que la información era un recurso corporativo de valor considerable. Estos percibieron más y más que los datos necesarios para contestar numerosas preguntas del negocio estaban disponibles en sus archivos de procesamiento de datos. Como consecuencia, comenzaron a presionar a los sistemas de información para la gestión en cuanto a la utilización de la potencia del computador para producir información a partir de los datos corporativos. Esto inició la demanda de los sistemas de bases de datos, los que garantizarían más efectivamente el acceso a los datos y su manipulación.

A mediados de los sesenta se introdujeron los primeros sistemas de bases de datos, cuyo fundamento era una estructura jerárquica de los datos. Estos sistemas permitieron la recuperación de múltiples registros asociados con un registro único de otro archivo. Inmediatamente después, se desarrollaron los sistemas de base de datos en redes que soportaron interrelaciones entre registros de archivos diferentes mucho más complejas. Ambos modelos de base de datos, el jerárquico y el en red, requirieron el uso de punteros físicos predefinidos para enlazar los registros relacionados.

En 1970, el artículo de E. F. Codd sobre el modelo de datos relacional revolucionó el pensamiento eu la industria de las bases de datos. El enfoque de Codd proponía el acceso y la manipulación de los datos únicamente desde el punto de vista de sus características lógicas. Durante los años setenta y ochenta se desarrollaron numerosos sistemas de base de datos relacionales y, en la actualidad, éstos dominan el mercado comercial.

En años recientes han proliferado los computadores personales en los puestos de trabajo, por lo que se han desarrollado las redes computacionales, permitiendo a los usuarios de estos computadores compartir recursos. Un computador, que funcioua como servidor de una red, garantiza el acceso a la base de datos desde las estaciones de trabajo en estos puestos, permitiendo uua división poderosa y eficiente de la tarea: El servidor recupera los datos, los que la máquina cliente solicitante procesa y presenta eu pantalla para su manipulación por parte del usuario final. Las redes computacionales en ambiente cliente/servidor han desarrollado un grado alto de sofisticación y se encuentran cada vez con más frecuencia en las empresas comerciales.

Desde el punto de vista conceptual, un sistema de base de datos en una organización grande está formado por el hardware, el software, los datos y las personas. La configuración del hardware comprende uno o nás computadores, unidades de disco, terminales, impresoras, unidades de cinta magnética, conexiones de red y otros dispositivos físicos. El software incluye un sistema de gestión de bases de datos (SGBD) y los programas de aplicación que utilizan el SGBD para tener acceso y manipular la base de datos. Los datos, que representan los hechos importantes para la organización, radican físicamente en el disco, pero se estructuran lógicamente de forma que se logre un acceso fácil y eficiente. Las personas, tanto los usuarios como los profesionales del sistema de base de datos, trabajan juntas para definir las características y la estructura del sistema de base de datos y para crear los programas de aplicación que garautizarán la información esencial para el éxito de la compañía.

1. Defina cada uno de los términos siguientes con sus propias palabras:
   a. sistema de proceso de datos
   b. proceso con acceso directo
   c. sistema de información para la gestión
   d. base de datos
   e. sistema de base de datos
   f. modelo de datos
   g. modelo jerárquico
   h. puntero
   i. red
   j. plataforma cliente/servidor
   k. sistemas abiertos
   l. interoperabilidad
   m. sistema de gestión de bases de datos
   n. vista de los datos
   o. procedimiento
   p. metadato

2. ¿Cuáles son las características más importantes de los sistemas de archivos con acceso secuencial y con acceso directo? ¿Cuáles son las limitaciones de los archivos con acceso secuencial que se remedian con los métodos de acceso directo?

3. Discuta la importancia de la información como un recurso orgánico. ¿Cómo han contribuido los sistemas de bases de datos a aumentar el valor de la información en las organizaciones?

4. Compare y contraste las características de los sistemas de bases de datos jerárquicos, en red y relacionales. ¿Cuáles necesidades de los negocios condujeron al desarrollo de cada uno de ellos?

5. Enumere y describa brevemente los cuatro componentes principales de un sistema de base de datos moderno.

6. Describa cada uno de estos componentes de un sistema de gestión de bases de datos (SGBD):
   a. Diccionario/directorio de datos (DD/D)
   b. Seguridad e integridad de los datos
   c. Acceso concurrente a los datos para varios usuarios
   d. Consulta, manipulación e informes de los datos
   e. Facilidades para el desarrollo de sistemas de aplicación

7. Enumere tres ejemplos para cada uno de los tipos de personas siguientes:
   a. Usuarios
   b. Profesionales de la computación

8. Defina cada uno de los elementos siguientes:
   a. ISAM
   b. IMS
   c. ABD (DBA)
   d. SQL
   e. Quel
   f. QBE

1. Haga corresponder los términos siguientes con sus definiciones:

—*datos*	**a.** Programa computacional que ejecuta una tarea de valor práctico.
—*clave*	**b.** Datos resumidos u organizados.
—*sistema de información*	**c.** Hechos aislados.
—*sinónimos*	**d.** Gente que necesita información de la base de datos.
—*padre*	**e.** Gente responsable del sistema de base de datos.
—*usuarios*	**f.** Sistema automatizado que organiza los datos para producir información.
—*programa de aplicación*	**g.** Términos que significan la misma cosa.
—*homónimo*	**h.** Registro "subordinado" en una interrelación jerárquica.
—*hijo*	**i.** Registro "propietario" en una interrelación jerárquica.
—*información*	**j.** Tiene significados diferentes en contextos diferentes.
—*profesionales de la computación*	**k.** Campos de los datos que identifican unívocamente un registro.
—*servidor de base de datos*	**l.** Proporciona medios convenientes para el acceso de los usuarios finales al sistema.
—*Interfaz gráfica del usuario*	**m.** Proporciona los servicios de la base de datos a las máquinas clientes.

2. Defina una clave para cada uno de los archivos de la Figura 1.3.

3. ¿Cuáles de los ejemplos siguientes pueden considerarse como datos y cuáles como información?
   a. El Mariscal Dobry recibió más comisión que cualquier otro representante de ventas este año.
   b. El Mariscal Dobry nació el 12 de diciembre de 1960.
   c. La región occidental produjo por encima de los $500.000 en ventas durante cada mes del último cuatrimestre.
   d. El producto A235 es lucrativo.
   e. El producto A235 se fabrica en Des Moines.

4. Para la base de datos de un banco, organice los archivos siguientes en una jerarquía: PAGO, CUENTA DE AHORROS, DEPÓSITO, CLIENTE, CUENTA DE PRÉSTAMOS, RETIRO DE DEPÓSITOS.

5. Para una base de datos de una compañía de embarque, organice los archivos siguientes en una red: EMBARQUE, TRANSPORTE, REMITENTE, PAQUETE, RECEPTOR.

6. Para los problemas 4 y 5, identifique los campos que pudieran encontrarse en cada archivo. Identifique los campos claves para cada archivo.

7. Explique cómo el proceso concurrente no controlado en un sistema de base de datos podría provocar daños en los datos en las situaciones siguientes:
   a. La reserva de vuelos en un sistema de reservación de una aerolínea.
   b. La actualización de las cantidades de un producto en un sistema de control de inventario.
   c. La actualización del balance de las cuentas para cheques en un banco.

1. Visite un centro de proceso de datos en su zona y hable con los administradores, los analistas y los programadores. Pregunte sobre su experiencia con sistemas de archivos tradicionales y con los sistemas de base de datos. Si ellos están utilizando un sistema de base de datos ahora, ¿utiliza éste alguno de los tres de modelos discutidos en el presente capítulo? ¿Cuáles son las ventajas y las desventajas percibidas por ellos en cuanto a los diversos enfoques que permiten el acceso a los datos? ¿Con qué configuración, desde el punto de vista del hardware, cuentan ellos? ¿Necesitaron escalar el hardware en el pasado y prevén ellos alguna necesidad en este sentido en el futuro próximo? ¿Cuáles son las principales necesidades de información para la gestión que garantiza el sistema?

2. Esboce la lógica de (a) un programa que utilice sistemas de archivos secuenciales y (b) un programa que utilice los archivos ISAM para resolver la demanda siguiente:

   Para cada cliente, identificar los fabricantes cuyos productos ha ordenado el cliente.

   Considere la estructura de archivos presentada en la Figura 1.3.

3. Estudie el artículo de Codd de 1970 que propone el modelo de datos relacional. Investigue los artículos subsiguientes que argumentaron los méritos relativos al modelo en red CODASYL con respecto a los del modelo relacional. Escriba un artículo que analice los méritos de ambos enfoques para los sistemas de bases de datos.

CAPÍTULO

2

SERVICIOS PARA CELEBRACIONES: UN EJEMPLO DE UTILIZACIÓN DE BASE DE DATOS

El caso de Servicios para Celebraciones
- Antecedentes
- Operaciones de los negocios
- Servicios de empleados
- Adquisición
- Planificación de tareas
- Promoción
- Estado financiero

Diseño e implementación de la base de datos
- Descripción del ciclo de recepción

Venta/Efectivo
- De las entidades a las tablas
- Creación de una tabla
- Diagrama revisado del modelo de datos
- Establecer las comprobaciones de validación

Características adicionales de los SGBD
- Introducción de datos a través de formularios simples
- Un formulario multi-tabla: el formulario solicitud

Obtener información para la gestión a partir de la base de datos

Resumen

Preguntas de comprobación

Problemas y ejercicios

Proyectos y cuestiones profesionales

Diane Bradbury estaba hablando con su hermana Karen. El fin de semana anterior hicieron preparativos de la recepción de la boda de una amiga y las cosas fueron bastante bien. Recibieron varios cumplidos y su amiga quedó muy satisfecha. Diane comentaba: "Tú sabes, Karen, a menudo hemos estado hablando de abrir un pequeño negocio. Recuerda el año pasado, cuando vimos aquella primorosa casa antigua que estaba en venta y pensamos que sería un sitio maravilloso para lugar de descanso. Por supuesto, no teníamos el dinero para comprar la casa, pero me parece que podemos comenzar un negocio de servicios para celebraciones sin tener que hacer una gran inversión. Creo que nos podría ir muy bien".

Karen le replicó: "Yo siento exactamente lo mismo. Yo sé que podemos hacer este tipo de trabajos de preparativos, pero ninguna de nosotras tiene experiencia en llevar el estado de los registros, las finanzas y todos los detalles que hay que llevar en un negocio. ¿Cómo nos la vamos a arreglar con esto?"

"No te preocupes," respondió Diane. "Mi esposo dice que muchos pequeños negocios son capaces de operar con bastante efectividad con un sistema de base de datos en microcomputadora. Esto es un software que permite conservar los registros convenientemente y mantener la información lista para poder acceder a ella. Él dice que cualquiera puede aprender a usarlo."

El caso de los Servicios para Celebraciones es un escenario fácil de seguir que introduce informalmente varios conceptos de diseño de base de datos que luego se desarrollarán más extensivamente en los capítulos posteriores.

Este capítulo nos ayuda a tener una panorámica del proceso de diseño e implementación de una base de datos. Esta experiencia servirá de motivación para temas posteriores, a la vez que generará las preguntas e ideas que serán estudiadas luego.

Después de estudiar este capítulo, el usuario:

- Tendrá una comprensión intuitiva del papel de un modelo de datos conceptual en el desarrollo de un sistema de base de datos.
- Tendrá una noción básica de cómo se traslada un modelo de datos a un modelo de implementación.
- Tendrá un conocimiento preliminar con algunas características comunes de los SGDB.

▼ El caso de Servicios para Celebraciones (Catering)

Antecedentes

Los Servicios para Celebraciones fueron iniciados por las dos hermanas, que estaban buscando una manera de hacer dinero extra en un trabajo que les interesara. Sus casas en Bainbridge Island están a un viaje corto en transbordador del centro de Seattle. Para bien o para mal, Bainbridge Island se ha desarrollado rápidamente en la medida que cada vez mayor cantidad de gente se ha sentido atraída por esta bella isla arbolada en Puget Sound —una localidad que ofrece un espacio abierto combinado con facilidades de acceso a las ventajas culturales y económicas de Seattle.

En paralelo con el crecimiento de Bainbridge Island han aparecido oportunidades para nuevos pequeños negocios. La isla atrae a numerosos turistas y han proliferado tiendas de piezas de arte, libros y comida local. Los entretenimientos también forman parte creciente de la vida de la comunidad en Bainbridge Island. Viendo una oportunidad de intentar algo por su cuenta, Karen y Diane Bradbury decidieron poner en juego sus habilidades culinarias con fines comerciales.

Tanto Karen como Diane eran enérgicas e imaginativas, también tienen talento en la preparación de platos. Además, ellas son queridas y tienen buenas relaciones con los amigos y vecinos de la isla. Su negocio fue fundado con la ayuda de una amiga mutua que era la esposa del director del Departamento de Música de una gran universidad en Seattle. El Departamento de Música fue renovado y encargado de soportar muchos de sus programas. Por consiguiente, el director y su esposa entretenían con frecuencia a grupos de parroquianos.

Después de atender exitosamente los servicios en dos eventos de la Escuela de Música, Karen y Diane fueron siempre las solicitadas para las celebraciones de eventos siguientes. Con este triunfo en la mano estaban en condiciones de adquirir nuevos clientes. La red que se fue tejiendo dio lugar a un negocio mayor que el que Karen y Diane podían atender por su cuenta, por lo tanto necesitaban mayor apoyo y crearon entonces los *Servicios para Celebraciones*.

En la actualidad, los Servicios para Celebraciones son propiedad de Diane y Karen Bradbury. Ellas tienen otros tres empleados: Peter Mancha, Dale Guber y Jeff Hoops.

Operación de los negocios

Los Servicios para Celebraciones tienen varios clientes que requieren los servicios de manera regular. Este negocio también es solicitado por clientes irregulares que acceden por vez primera. Diane y Karen desarrollan los presupuestos para cada trabajo de la manera siguiente: Primero se determina el tipo de comida, su preparación y su servicio. A este trabajo se le pone un precio por cubierto. Esta cantidad se multiplica por el máximo número de invitados esperados. Esta información se registra en un modelo de presupuesto del trabajo y luego se introduce en una microcomputadora.

Servicios de empleados

El veinte por ciento de las ganancias de cada trabajo se destina para compensar a los empleados. Si más de un empleado participa en un trabajo, este veinte por ciento se divide entre ellos. A los empleados se les paga cada dos semanas. Diane y Karen son las responsables de las compras de suministros y equipos. Diane lleva la cuenta de quién participa en cada trabajo y determina la cantidad a pagar a cada empleado. Karen tiene la responsabilidad de pagar todos los otros gastos.

Adquisición

Todas las compras son realizadas con cuentas con varios vendedores de la zona. Karen paga los recibos a los treinta días, a menos que haya algún descuento por pronto pago.

Planificación de tareas

Los empleados se asignan a cada tarea de acuerdo a sus habilidades y disponibilidad. Ocasionalmente se requiere alguna ayuda adicional durante parte del tiempo, pero esas necesidades por lo general se satisfacen con miembros de la familia de Karen y Diane. Diane normalmente hace las asignaciones de tareas con una semana de anticipación . Cuando se termina un trabajo, el supervisor (que pudiera ser Karen, Diane o algún empleado) definirá y rellenará un modelo con el listado de los equipos utilizados, quiénes trabajaron y cuántas horas.

Promoción

Diane y Karen han sido efectivas vendiendo estos servicios de celebraciones a través de sus contactos personales. También muchos de los trabajos han aparecido por recomendaciones de clientes satisfechos. No dejando nada al azar, Diane y Karen preparan ocasionalmente propaganda que se distribuye en las tiendas locales, que son frecuentadas por clientes potenciales. Ellas también han puesto anuncios en el periódico local de Bainbridge Island.

Estado financiero

Puesto que Diane y Karen operan los Servicios para Celebraciones desde sus propias casas, ellas tienen pocos sobrecostos. Inicialmente se compraron algunos equipos, y tienen lo que necesitan para el nivel actual del negocio. La mayoría de los costos proviene de las comidas y del personal. Las ganancias se determinan restando los salarios y los costos de las comidas de los ingresos recibidos por cada trabajo.

▼ Diseño e implementación de la base de datos

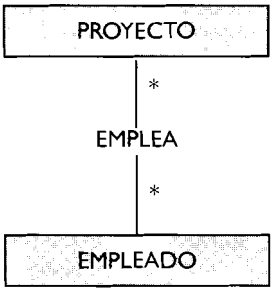
El propósito de esta sección es brindar algunas ideas concernientes a los pasos necesarios para diseñar una base de datos sencilla de Servicios para Celebraciones. Se usará el SGBD PARADOX PARA WINDOWS para ilustrar algunas de las características de implementación que son comunes a muchos SGBDs. Puesto que no se va e entrar en los detalles del sistema PARADOX , la presentación será de cierto modo genérica y fácil de entender.

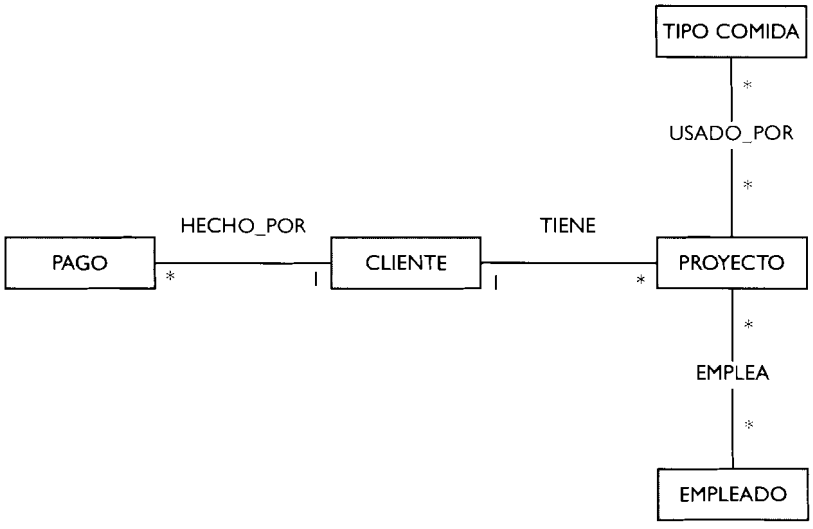
Descripción del ciclo de recepción venta/efectivo

El primer paso para desarrollar una base de datos para los Servicios para Celebraciones es crear un modelo de datos conceptual de sus acciones de negocios. Se necesitan las siguientes entidades para ilustrar el ciclo de recepción venta/efectivo: CLIENTE, PROYECTO, EMPLEADO, TIPO_COMIDA y PAGO. En nuestro modelo de datos se representarán estas entidades con rectángulos. Por ejemplo, la entidad PROYECTO es:

La Figura 2.1 es un diagrama del modelo de datos que muestra cómo se relacionan entre sí las entidades.

¿Qué es lo que dice la Figura 2.1 sobre el ciclo de recepciones ventas/efectivo? Veamos la interrelación entre la entidad PROYECTO y la entidad EMPLEADO.



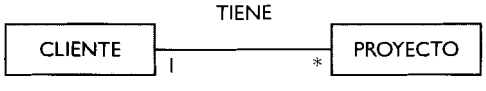


La línea entre las entidades significa que éstas están relacionadas. El "*" y el "1" dicen cómo están relacionadas. La interrelación entre PROYECTO y EMPLEADO se lee:

1. Cada PROYECTO puede emplear a uno o más (*) EMPLEADOs.
2. Cada EMPLEADO puede trabajar en uno o más (*) PROYECTOs.

En otras palabras, cada empleado puede trabajar en varios proyectos y en cada proyecto pueden trabajar varios empleados.

Examinemos otra interrelación. Véase la interrelación entre la entidad CLIENTE y la entidad PROYECTO.



De nuevo la línea entre las dos entidades dice cómo están relacionadas. El "1" y el "*" dicen cómo están relacionadas. Esta interrelación se lee:

1. Cada CLIENTE puede tener uno o más (*) PROYECTOs.
2. Cada PROYECTO puede tener sólo un CLIENTE.

En otras palabras, cada proyecto de celebración puede ser solicitado por un solo cliente, pero cada cliente puede solicitar varios proyectos.

En la Figura 2.1 se pueden ver dos clases de interrelaciones. Estas son:

1. Uno-a-muchos (1 a *), que es la que se muestra entre CLIENTE y PROYECTO, y entre CLIENTE y PAGO.
2. Muchos-a-muchos (* a *) que es la que se muestra entre PROYECTO y EMPLEADO, y entre PROYECTO y TIPO COMIDA.

Existe también otro tipo de interrelación entre tablas que no se ha usado en este ejemplo. Esta es la interrelación uno-a-uno (1 a 1).

Ya se sabe cuáles son las entidades involucradas y cómo éstas están relacionadas entre sí. Estamos entonces preparados para comenzar a hacer corresponder el modelo con una implementación.

De las entidades a las tablas

Una *tabla* puede concebirse como una implementación de una entidad del modelo. En una tabla los datos se organizan en campos y registros (equivalentemente en columnas y filas). Por ejemplo, la tabla CLIENTE puede verse como algo parecido a:

Nombre	*Dirección*	*Teléfono*
Chuck Brown	321 Schultzville Seattle, WA 92890	565-9980
Debbi Allen	15250 Riverside Lane Seattle, WA 98432	645-2322
Leroy Brown	52000 Basketball St. Seattle, WA 91130	685-4526
Al Franken	5230 Wesley Avenue Seattle, WA 93401	684-2388

Nombre, *Dirección* y *Teléfono* son ejemplos de campos. El campo *Nombre* es una columna que contiene solamente nombres. Un registro *(record)* es una fila que contiene el nombre de un cliente, su dirección y su teléfono. Como se puede observar, la tabla CLIENTE contiene cuatro registros, uno para cada cliente, y tres campos, *Nombre*, *Dirección* y *Teléfono*. Los campos describen propiedades importantes de las entidades.

Ahora se define esta tabla en un SGBD. Se necesita crear cinco tablas para los Servicios para Celebraciones. Estas son CLIENTE, PROYECTO, EMPLEADO, TIPO COMIDA y PAGO. Se comenzará con la tabla CLIENTE.

Creación de una tabla

Supongamos que se desea dividir la información sobre una dirección en cuatro campos: Calle, Ciudad, Estado (provincia) y Código postal. Se tiene entonces un total de seis campos: *Nombre*, *Calle*, *Ciudad*, *Estado*, *CP* y *Teléfono*.

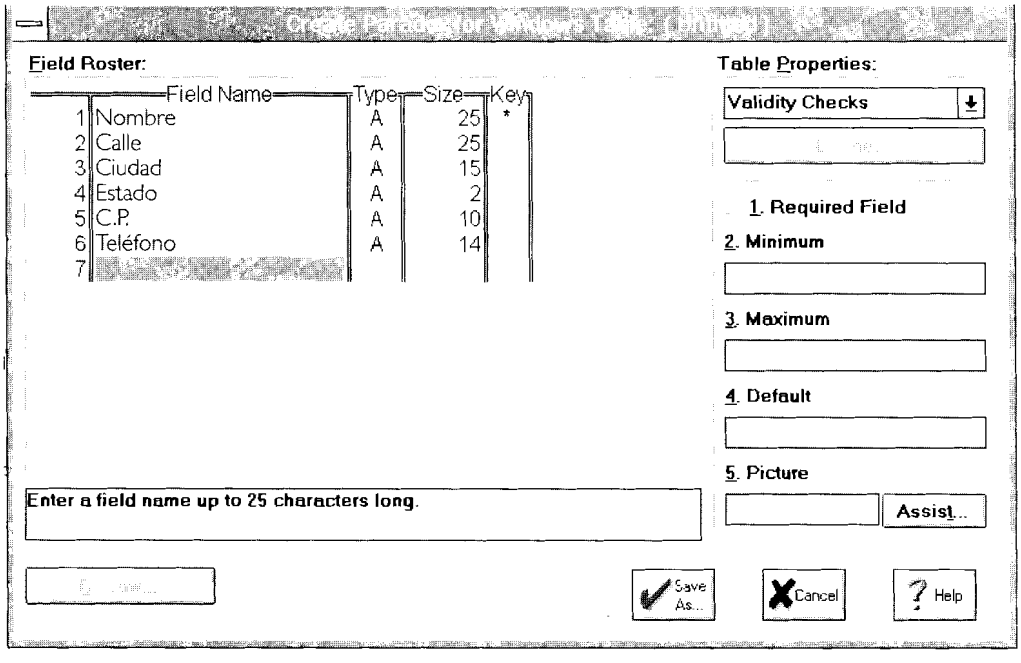
Una vez que se han escogido los campos, se diseña la tabla correspondiente en el SGBD. Véase la Figura 2.2. Aquí se pueden observar los requisitos típicos para diseñar una tabla de una base de datos. Se pueden ver los nombres de los campos (por ejemplo, *Nombre*, *Calle*, etc.), una indicación sobre el tipo de datos de cada campo (aquí la "A" significa alfanumérico), el tamaño de cada campo, y una indicación sobre cuál campo va a hacer las veces de clave (un campo cuyo valor es único para cada registro, no hay dos registros con el mismo valor para ese campo).

Ha sido creada la estructura de la tabla CLIENTE, que está ahora lista para recibir datos.

De forma similar se crean las otras cuatro tablas que se muestran en la Figura 2.3.

Diagrama revisado del modelo de datos

Con vistas a poder representar nuestro modelo de datos en su totalidad, se debe asegurar que las tablas se relacionan entre sí en la forma que deben. ¿Recuerda las tres clases de interrelaciones entre tablas? Por conveniencia las volveremos a repetir aquí:



Nombre de campo	Tipo	Tamaño	Clave
Proyec#	N		*
Fecha	D		
#Invitados	N		
Costo Est	$		
Cant. cargada	$		

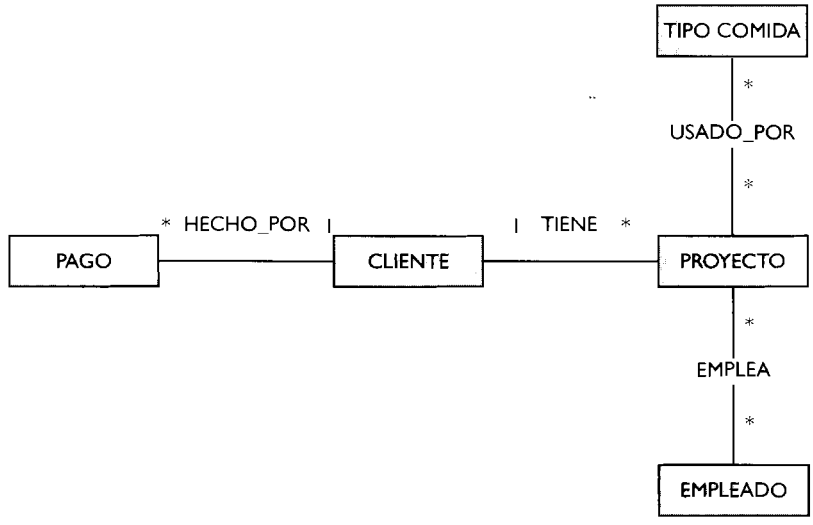
Nombre de campo	Tipo	Tamaño	Clave
SS#	A	11	*
Nombre	A	25	

Nombre de campo	Tipo	Tamaño	Clave
Nombre_Comida	A	25	*

Nombre de campo	Tipo	Tamaño	Clave
Proyec#	N		*
Cant. recibida	$		

1. Uno-a-uno (1 a 1) no existe en el modelo de Servicios para Celebraciones.
2. Uno-a-muchos (1 a *) se muestra en las interrelaciones HECHO-POR y TIENE.
3. Muchos-a-muchos (* a *) se muestra en las interrelaciones USADO-POR y EMPLEA.

Para estar seguros que estas interrelaciones están representadas correctamente en la tabla, se deben seguir ciertas reglas, basadas en la interrelación en particular, para cada tabla. Echémosle un segundo vistazo al diagrama.



Hay tres reglas de tablas que gobiernan las interrelaciones entre ellas.

Regla de Tabla 1:

Si dos tablas tienen una interrelación uno-a-uno (1 a 1), entonces el campo clave de una de las tablas debe aparecer en la otra tabla.

Ninguna de las interrelaciones de nuestro ejemplo se avienen a esta regla 1.

Regla de Tabla 2:

Si dos tablas tienen una interrelación uno-a-muchos (1 a *), entonces el campo clave de la tabla del (1) debe aparecer en la tabla del (*).

En el diagrama, dos interrelaciones admiten esta regla 2, la interrelaciones TIENE y HECHO-POR.

Para cumplir con la Regla de Tabla 2 para las tablas CLIENTE y PAGO, se pondrá *Nombre*, el campo clave de CLIENTE, en la tabla PAGO después del campo *Proyec#*. Puesto que *Nombre* es la clave de la tabla CLIENTE y se usa para conectarla a otras tablas, se le conoce como clave foránea.

Continuando se tiene

Regla de Tabla 3:

Si dos tablas tienen una interrelación muchos-a-muchos (* a *), entonces debe crearse una nueva tabla que tenga los campos claves de las dos tablas.

En el diagrama, dos interrelaciones cumplen la Regla de Tabla 3, USADO-POR y EMPLEA. Para satisfacer la regla se deben crear dos nuevas tablas. Primero se diseña con la que se enlazarán PROYECTO y TIPO_COMIDA. Esta tabla se llamará USADO_POR. Tendrá tres campos: *Proyec#* (primera clave), *Nombre_Comida* (segunda clave) y *Cantidad*. Note que entonces la tabla USADO_POR contiene los campos claves de PROYECTO y TIPO_COMIDA.

Nombre de campo	Tipo	Tamaño	Clave
Proyec#	N		*
Nombre_Comida	A	25	*
Cantidad	N		

Ahora diseñaremos la tabla que enlaza PROYECTO y EMPLEADO. Esta tabla se llamará EMPLEA. Tendrá tres campos: *Proyec#* (primera clave), *SS#* (segunda clave) y *Horas*. Usando los mismos pasos que en el caso anterior, se construye esta tabla EMPLEA.

Nombre de campo	Tipo	Tamaño	Clave
Proyec#	N		*
SS#	A	11	*
Horas	N		

Note cómo luce ahora el diagrama del modelo de datos en su versión modificada (Figura 2.4) y cómo las nuevas tablas cumplen con la Regla de Tabla 2.

Ya están diseñadas todas las tablas y se refieren entre sí correctamente. Por "se refieren entre sí correctamente" se entiende que las tablas se relacionan de forma que representan con precisión la estructura de los Servicios para Celebraciones.

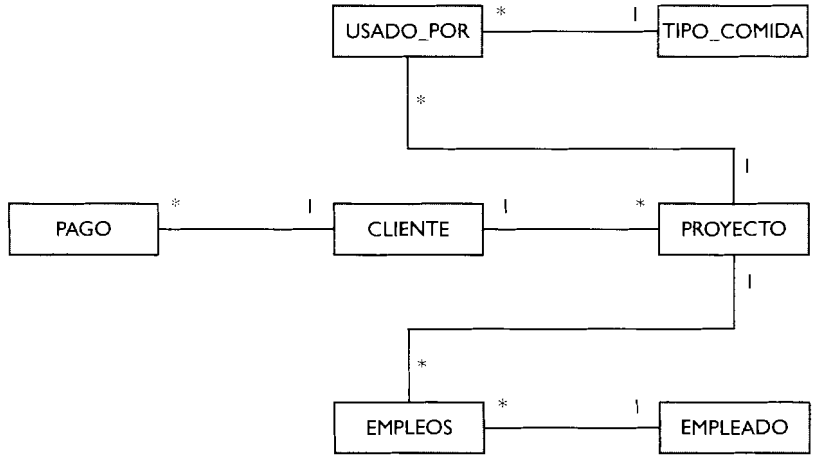
Establecer las comprobaciones de validación

La mejor manera de garantizar que los datos que se almacenen sean válidos es en primer lugar prevenir que no se puedan introducir datos incorrectos. Por esta razón es típico que los SGBDs permitan imponer restricciones (tests de validación) para cada campo en las tablas de una base de datos.

Los tipos de tests de validación más comunes son:

Campo obligatorio: Se debe introducir un dato en este campo antes de poder guardar el registro.

Mínimo: El menor valor que este campo puede aceptar.



Máximo: El valor mayor que este campo puede aceptar.
Por defecto: El valor que será puesto automáticamente en el campo por PARADOX si el usuario omite el valor.
Formato *(picture)*: Una plantilla de los datos válidos y el formato de este campo.

Con estos chequeos podemos estar seguros que el código postal, los números de teléfono y los números de seguridad social se introducen en la forma correcta; que los valores de tipo dinero ($) están dentro de un rango razonable, y que los campos que sean claves siempre contendrán un dato. La Figura 2.5 ejemplifica cómo añadir los chequeos de validación a nuestras tablas.

Nombre de campo	Campo obligatorio	Mínimo
Proyec#	✔	
Costo Est.		
Cant. cargada		

Nombre de campo	Campo obligatorio	Formato
SS#	✔	###_##_#####

Nombre de campo	Campo obligatorio
Nombre_Comida	✔

Nombre de campo	Campo obligatorio	Mínimo
Proyec#	✔	
Cant. recibida		0.00

Nombre de campo	Campo obligatorio	Mínimo	Máximo
Proyec#	✔		
Nombre_Comida	✔		
Cantidad		0	999

Nombre de campo	Campo obligatorio	Mínimo	Formato
Proyec#	✔		
SS#	✔		###-##-####
Horas		0	

▼ Características adicionales de los SGBD

Introducción de datos a través de formularios simples

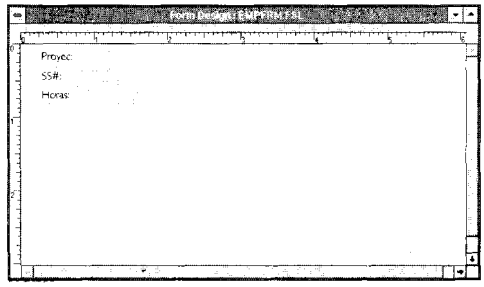
Un formulario *(form)* es una manera de ver o de entrar los datos en una tabla —un registro de cada vez—. Los formularios pertenecen a las tablas; es decir, no se puede diseñar un formulario a menos que ya se haya creado la tabla para la que se va a definir éste. Los formularios contienen todos o algunos de los campos de la tabla respectiva.

La razón por la que se desea diseñar formularios es para facilitar la introducción de datos. Cuando se empieza a poner información en las tablas es conveniente tener formularios que nos faciliten el proceso de entrada de datos. Aunque se pueden introducir los datos directamente en las tablas, usando formularios es un modo a menudo más rápido, más efectivo y más preciso de hacerlo.

Se han creado las tablas para representar el ciclo de recepción de ventas/efectivo de los Servicios para Celebraciones. Los formularios que se considerarán serán para registrar una venta, registrar la recepción de un pago y registrar otras informaciones que abarcan las ventas.

El ejemplo más simple es un formulario que permite justamente entrar una tabla. Un proyecto de servicio implica la labor de empleados. Cuando el proyecto termine, el Servicio para Celebraciones necesita conocer quién trabajó en el proyecto y por cuánto tiempo. Para entrar estos datos se puede crear el **Formulario Trabajo.** Note que este formulario introduce información en la tabla EMPLEA, la cual contiene tres campos: *Proyec#* (primera clave), *SS#* (segunda clave) y *Horas*.

En la Figura 2.6 se muestra un ejemplo.



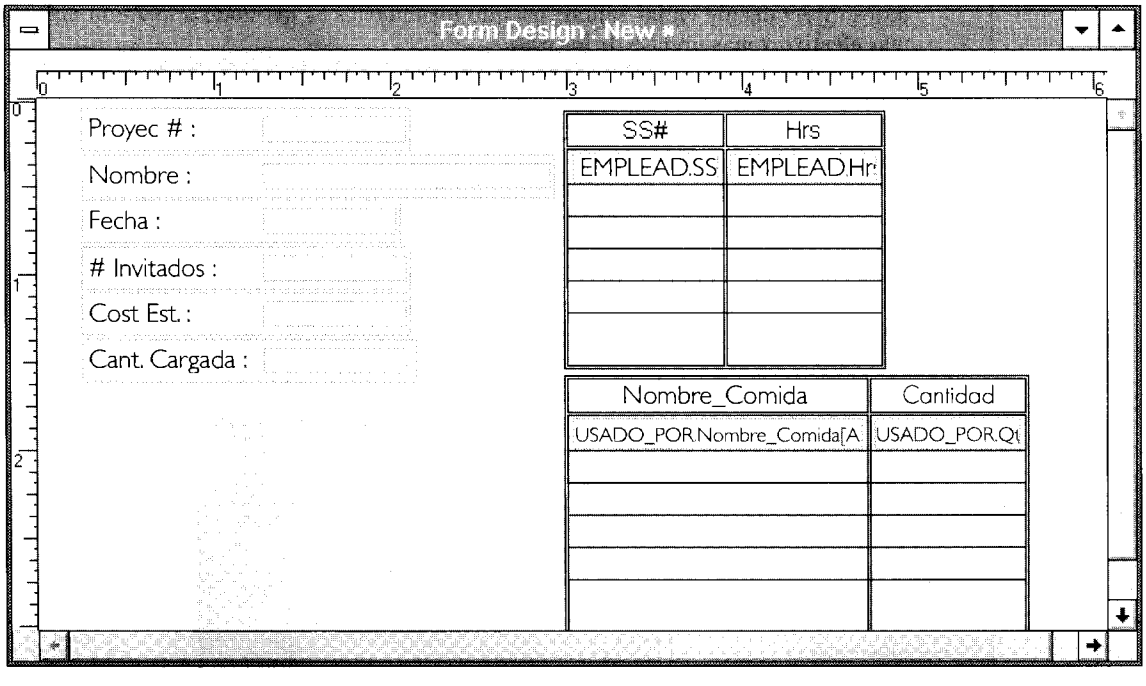
Un formulario multi-tabla: el formulario solicitud

Un formulario multi-tabla es un modo elegante de introducir los datos de varias tablas a la vez. Se hará una demostración con una multi-tabla **Formulario Solicitud.** El Formulario Solicitud nos ayudará a poner en varias tablas información sobre las ventas. Cuando un cliente solicita un servicio para una celebración, la información que éste da afecta a tres de nuestras tablas.

Cuando se recibe una solicitud, se registra cierta información (a) y se pone ésta en las tablas específicas (b):

1. (a) El número de proyecto de servicio
   (b) PROYECTO, USADO_POR
2. (a) El nombre del cliente
   (b) PROYECTO, CLIENTE
3. (a) La fecha de la celebración, el número de invitados esperados, el costo estimado del servicio y el total que se le cargará al cliente
   (b) PROYECTO
4. (a) La calle, ciudad, Estado, código postal y número telefónico del cliente
   (b) CLIENTE
5. (a) Los tipos de comida que el cliente desea y la cantidad que desea de cada una
   (b) USADO_POR

Como se puede observar, la información va a parar a tres tablas. Creando un formulario multi-tabla, se puede enviar la información correcta al campo y tabla deseada en un solo paso. Sin mucho más detalle, en la Figura 2.7 se ve un ejemplo del **Formulario Solicitud** final. La Figura 2.8 nos muestra ejemplos de datos que se introducen usando formularios.



Nombre	Calle	Ciudad	Estado	CP	Teléfono
Baskerville, Eugene	120 Fir Lane	Redmond	WA	98204	123-4560
Jordan, Harry	14 Elm	Winslow	WA	98223	245-9366
Smith, Carrie	25 Alpine Dr.	Everett.	WA	98882	546-4322

Proyec#	Nombre	Fecha	#Invitados	Costo Est	Cant. Cargada
1	Jordan, Harry	21/3/93	100	$600,00	$850,00
2	Smith, Carrie	15/4/93	50	$325,00	$425,00
3	Baskerville, Eugene	17/4/93	25	$275,00	$350,00
4	Smith, Carrie	1/5/93	25	$325,00	$560,00

Proyec#	Nombre_Comida	Cantidad
1	mousse de chocolate	125
1	cerveza	75
1	ponche	150
1	bistec	100
2	7-Up	75
2	hamburguesa	75
2	gelatina	50
2	pastel	50
3	croquetas de carne	60
3	cóctel de frutas	50
3	ensalada verde	50
3	Sprite	60
4	espárragos	45
4	patatas fritas	60
4	jugo de arándanos	50
4	turrón	35
4	costillas barbacoa	40

▼ Obtener información para la gestión a partir de la base de datos

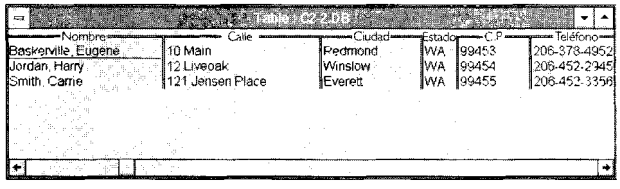
Ahora que ya (1) se han definido las tablas que se necesitan en la base de datos, (2) se han definido las interrelaciones entre las tablas (diagrama de tablas y modelo de datos), (3) se han creado las tablas, y (4) se han introducido los datos en las tablas, estamos listos para usar la base de datos para obtener información que nos ayude a gestionar los Servicios para Celebraciones.

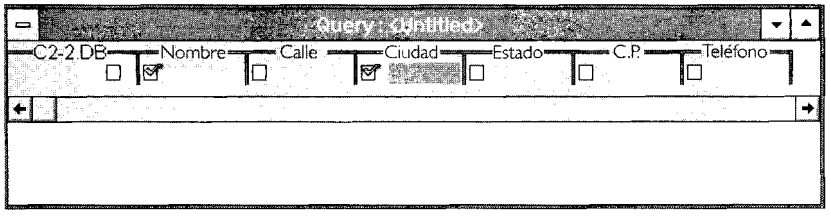
Una consulta es una simple pregunta que se puede responder con la información de la base de datos. Se ilustrará un lenguaje que permite al usuario hacerle preguntas a una base de datos usando plantillas visuales en lugar de órdenes en texto. Tanto los lenguajes visuales como los lenguajes textuales han emanado de la teoría de los lenguajes relacionales. De todos modos, estos temas serán aclarados en los capítulos posteriores.

Caminemos a través de una consulta simple.

Consulta 1. Esta consulta nos da una lista de los nombres de todos los clientes junto con las ciudades en las que viven. Hay que hacer lo siguiente:

1. Seleccionar **File | New | Query** del menú principal [1].
2. En el cuadro de diálogo seleccionar archivo (Select File) que aparecerá, debemos escoger la tabla que estará implicada en la consulta. Se selecciona CLIENTE.DB [2] y se pulsa **OK.** Aparecerá entonces una plantilla de consulta en blanco que contiene todos los campos de la tabla CLIENTE.
3. Se pulsa en las cajas de chequeo *(check box)* que aparecen al lado de los campos *Nombre* y *Ciudad.* Esto significa que hemos seleccionado estos campos y que serán por tanto los únicos que aparecerán en la salida. La ventana de consulta lucirá como la de la Figura 2.9(a).
4. Se pulsa en el botón, , de la barra rápida *(Speedbar)* o se pulsa la tecla **F8.** Esto provoca que se ejecute la consulta y nos da la tabla de respuesta (Figura 2.9(b)).





[1] Archivo | Nuevo | Consulta (N. del T.)

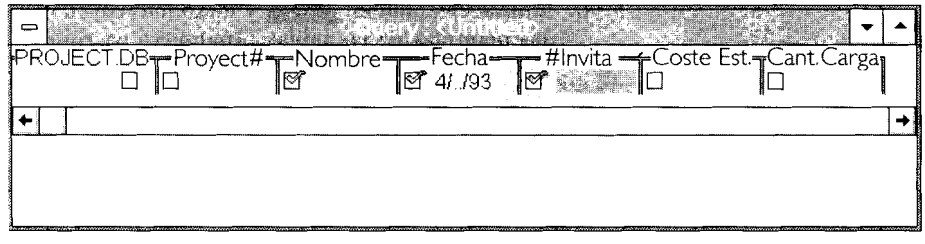
[2] Por uniformidad con lo que supone el SGBD PARADOX, la extensión de los archivos, que son tablas de la base de datos, se ha dejado en inglés ('DB' proviene de Data Base) (N. del T.)

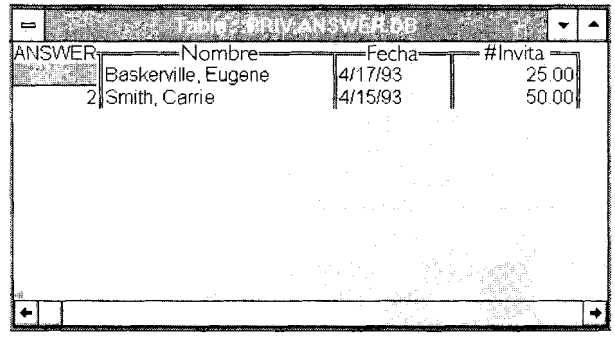
Se puede hacer una gran número de consultas útiles.

Consulta 2. Supongamos que queremos una lista de todos los trabajos de servicio que están planificados para el mes de abril de 1993, los nombres de los clientes y cuántos invitados se esperan en cada evento. Se debe proceder de la siguiente manera:

1. Seleccionar **File | New | Query** del menú principal.
2. Puesto que la columna comprende información de la tabla PROYECTO, seleccionamos PROYECTO.DB del cuadro de diálogo de seleccionar archivo (Select File) y pulsamos en **OK.** De nuevo se verá una plantilla de consulta en blanco, pero en este caso será para la tabla PROYECTO.
3. Se pulsa en las cajas de chequeo que están al lado de los campos *Nombre, Fecha e #Invitados.*
4. Pulsamos de nuevo en el campo *Fecha* y tecleamos "../4/93". Los dos puntos indican cualquier fecha válida en abril del 93, por lo que cualquier proyecto con fecha en abril aparecerá en la tabla de respuesta. La consulta se ve como en la Figura 2.10(a).
5. Se pulsa en el botón. , o se presiona **F8** y el resultado de la consulta se muestra como en la Figura 2.10(b).
6. Cierre ambas ventanas y decida si se quiere salvar la consulta o no. Si queremos salvarla debemos pulsar en **Yes** e introducir un nombre para la consulta. Luego se pulsa en **OK.** Si no queremos salvar la consulta, pulsaremos en ***No.***

Hasta ahora las consultas sólo han involucrado a una tabla. Veremos que es elemental enlazar los datos de varias tablas diferentes usando elementos ejemplo. Esto se demuestra en la próxima consulta.



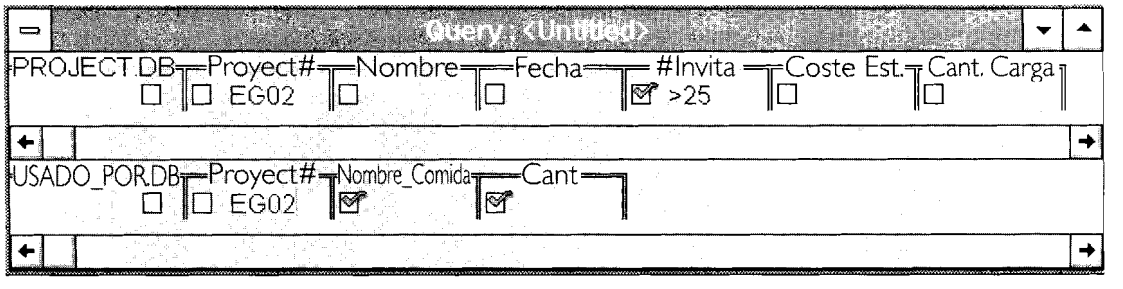


Consulta 3. Esta consulta pregunta por las comidas y las cantidades que han sido usadas en proyectos de más de 25 invitados. Note que las comidas y las cantidades aparecen en la tabla USADO_POR, pero que el número de invitados por proyecto está en la tabla PROYECTO. Para llevar a cabo una consulta multi-tabla se hace lo siguiente:

1. Seleccionar **File | New | Query** del menú principal.
2. Seleccionar PROYECTO.DB y pulsar en **OK.** No se preocupe que la tabla USADO_POR se seleccionará de otra manera. De nuevo se verá la plantilla de consulta en blanco para la tabla PROYECTO.
3. Pulsar en el botón de adicionar de tabla (Add Table) de la barra rápida *(speed bar)*. De nuevo saldrá el cuadro de diálogo para seleccionar un archivo (File dialog box). Seleccionar la tabla USADO_POR (archivo USADOPOR.DB [3]) y pulsar en **OK.** Se verá ahora una plantilla en blanco para consultar ambas tablas.
4. Marcar las cajas de chequeo próximas a los campos *#Invitados*, *Nombre_Comida* y *Cantidad*...
5. Pulsar en el campo *#Invitados* y teclear ">25".

Se debe hacer ahora el enlace entre las dos tablas. El enlace obvio para esto es el campo *Proyec#*. Para hacer este enlace se pulsa el botón de reunión de tablas (Join Tables) , que también aparece en la barra rápida. Después de pulsar este botón se debe pulsar en el campo *Proyec#* de las plantillas *(templates)* de ambas tablas PROYECTO y USADO_POR. PARADOX crea un único elemento ejemplo *(example element)* y pone éste en ambas plantillas. La consulta multi-tabla se puede ahora completar con éxito.

6. Pulsar en el botón y luego hacerlo en el campo *Proyec#* en cada una de las plantillas de las tablas. Aparecerá en rojo un EG01 en ambos campos que indica que el enlace fue exitoso. Esta consulta luce ahora como la que se muestra en la Figura 2.11(a).
7. Pulsar en el botón de ejecutar consulta (Run Query), , o presionar **F8.** La tabla de respuesta se muestra en la Figura 2.11(b).



[3] Recuerde que en muchos sistemas operativos los nombres de archivo sólo pueden tener hasta 8 caracteres. En el original en inglés no existe esa diferencia porque la tabla se llama USED_By (que hacen 7 caracteres) (N. del T.)

Tabla: PARTANSWER.DB

ANSWER	#Invita	Nombre_Comida	Cant.
	35,00	asparagus`	45,00
2	35,00	baked potato	60,00
3	35,00	cranberry juice	50,00
4	35,00	fruit flip	35,00
5	35,00	prime rib	40,00
6	50,00	7-up	75,00
7	50,00	hamburger	75,00
8	50,00	jello	50,00
9	50,00	shortcake	50,00
10	100,00	chocolate mousse	125,00
11	100,00	root beer	75,00
12	100,00	sparkling punch	150,00
13	100,00	steak	100,00

En este capítulo se han introducido elementos del diseño e implementación de bases de datos que se estudiarán con más rigurosidad en los capítulos siguientes. La aplicación fue realista, pero lo suficientemente simple, de modo que esperamos que el lector haya ganado en claridad sobre el método. Se ha desarrollado la aplicación desde el diseño hasta la implementación. La explicación sobre la implementación nos demostró el poder de un SGBD sin tener que haber entrado en demasiado detalle.

1. Describa con sus propias palabras la diferencia entre una tabla y un campo.
2. ¿Cómo se representa una tabla en el diagrama de un sistema de información?

1. Usando un ejemplo sencillo de un sistema de negocios, bosqueje un diagrama similar al que se ha utilizado en este capítulo. Identifique las interrelaciones entre las tablas como uno-a-uno, uno-a-muchos y muchos-a-muchos.
2. Usando los resultados de (1), crear las tablas correspondientes.
3. Usando los resultados de (2), crear formularios para la entrada de datos y entre algunos datos como ejemplo.

1. Si el usuario está familiarizado con, o tiene acceso a otro SGBD, use el ejemplo de este capítulo para comparar y contrastar los métodos de creación de tablas, entrada de datos y formulación de consultas.
2. Encuentre algún negocio local (o alternativamente invéntese alguno) que proporcione las bases para desarrollar una pequeña aplicación de base de datos. Diseñe y realice dicha aplicación.

CAPÍTULO

3

SISTEMAS DE BASES DE DATOS EN LAS ORGANIZACIONES

Compartir datos y bases de datos
- Compartir datos entre unidades funcionales
- Compartir datos entre diferentes niveles de usuarios
- Compartir datos entre diferentes localidades
- El papel de la base de datos

Planificación estratégica de bases de datos
- La necesidad de planificar la base de datos
- El proyecto de planificación de la base de datos
- El ciclo de vida del desarrollo de la base de datos (CVDBD)

Bases de datos y gestión de control
- Diseño de la base de datos
- Formación del usuario
- Seguridad de la base de datos e integridad
- Rendimiento del sistema de base de datos

Riesgos y costos de las bases de datos
- Conflictos en las organizaciones
- Fracasos en el desarrollo de proyectos
- Mal funcionamiento del sistema
- Costes imprevistos
- Necesidad de un personal cualificado

Separar la representación lógica y física de los datos
- Arquitectura de tres-niveles de una base de datos
- Uso de la arquitectura tres-niveles en este libro

Desarrollo de la base de datos
- Diseño de base de datos y el CVDS[1] tradicional
- El ciclo de vida del desarrollo de la base de datos (CVDBD)
- El caso de la Corporación Zeus
- Planificación preliminar
- Estudio de viabilidad
- Definición de requisitos
- Diseño conceptual
- Implementación
- Evaluación y perfeccionamiento del Esquema de base de datos

Construir capacidades en el desarrollo de bases de datos

Resumen

Preguntas de repaso

Problemas y ejercicios

Proyectos y cuestiones profesionales

[1] Siglas de Ciclo de Vida de Desarrollo de Sistemas (SDLC en el original en inglés por *Systems Development Life Cycle* (N. del T.).

"Como puedes ver, Susan, este plan estratégico de base de datos destaca el camino que tomará el sistema de información de IPD para los próximos cinco años. En cinco años completaremos la instalación de los sistemas de bases de datos en todas nuestras oficinas."

Cordelia Molini, la administradora de base de datos de International Product Distribution (IPD)[2]*, le está presentando un informe de planificación de bases de datos para cinco años a Susan Broadbent, presidenta de IPD. IPD acaba de terminar su proyecto de planificación estratégica de base de datos, del cual Cordelia fue el líder de proyecto.*

Susan pregunta: "¿Has tenido en cuenta el problema de preservar la seguridad de nuestros datos en un entorno internacional?

"Sí, y hemos analizado los costos y riesgos que se derivan de ello. Hemos considerado también las necesidades de información de las áreas funcionales de IPD a diferentes niveles de dirección. Cuando revises los informes en detalle, verás que hemos propuesto los planes para varios proyectos de desarrollo de bases de datos junto con sus costos estimados."

"Este es un trabajo excelente. Debemos felicitar a vuestro equipo por habernos dado la armazón de un sistema de base de datos que encaja tan bien con nuestros planes de negocios."

En este capítulo se centrará la atención en el entorno organizativo, en el cual existen las bases de datos y en cómo estas bases de datos se interrelacionan con su entorno. Además se examinará el proceso de desarrollo de un sistema específico de base de datos. Para facilitar la discusión daremos una simple definición de lo que es una base de datos. Una **base de datos** es una organización de una colección de datos que se interrelacionan, se comparten y se controlan. Después de leer este capítulo, el usuario deberá ser capaz de:

base de datos. Una colección de datos interrelacionados, compartidos y controlados.

- Estudiar cómo en una organización se comparten los datos entre diferentes áreas funcionales, niveles de dirección y localizaciones geográficas.
- Explicar por qué y cómo se hace la planificación estratégica de una base de datos en una organización.
- Comprender la función de control de la administración de base de datos.
- Hacer los diagramas y explicar los fundamentos estándares de la estructura de una base de datos, la arquitectura tres-niveles de ANSI/SPARC.
- Describir los pasos en el ciclo de vida del desarrollo de la base de datos y sus interrelaciones.

Las primeras cuatro secciones de este capítulo tratan aspectos relativos a la gestión de compartir los datos en una organización, la planificación estratégica de los datos, la gestión del control de los datos y los riesgos y los costos de las bases de datos. La segunda mitad del capítulo tiene que ver específicamente con los temas de gestión relativos al diseño y la realización de una base de datos. De modo que se estudiarán las preguntas de cómo separar la estructura física y lógica de los datos y de cómo desarrollar una base de datos a través de una metodología del ciclo de desarrollo.

[2] Distribuidora Internacional de Productos (N. del T.).

▼ Compartir datos y bases de datos

El sistema de base de datos de IDP pasó por varios estadios de evolución antes de llegar a su nivel de sofisticación actual. Para entender mejor por qué IDP decidió pasar a un sistema de base de datos, debemos volver atrás varios años y ver una conversación entre Susan Broadbent y Sanford Mallon, jefe de información oficial de IDP.

"Sandy, pienso que nuestra compañía se está volviendo muy descentralizada. En la medida en que nuestros grupos funcionales crecen, éstos están más y más aislados. Yo creo que muchos grupos no toman en cuenta lo que otros grupos están haciendo."

"Siento tener que escuchar esto, Susan, pero ¿qué tiene que ver esto conmigo?"

"¡Gracias mil! Ya veo que la actitud aislacionista ha alcanzado también a los Sistemas de Información. Está bien, me explicaré. Nuestros grupos producen informes administrativos para ellos mismos, pero ellos no comparten información con otros grupos. Por ejemplo, Marketing (mercadotecnia) tiene todos los datos sobre la satisfacción o insatisfacción de los clientes con los productos. Sin embargo, Evaluación de Productos nunca ve estos datos. ¿Por qué no podemos lograr que nuestra gente comparta la información de manera más efectiva?"

"Realmente —replicó Sandy— Evaluación de Productos ha visto estos informes de Marketing, pero dicen que los formatos son muy confusos. Podríamos escribir los programas para darle a la gente de Evaluación la información que necesitan, pero Marketing no deja que otros grupos tengan acceso a sus datos. Ellos temen que se puedan extraviar o confundir los datos."

"Tú has estado hablando antes de pasar a un sistema de base de datos. ¿Nos ayudaría esto a resolver este problema?"

"Ciertamente que nos ayudaría. Estaríamos capacitados para compartir los datos de una manera controlada, de modo que Evaluación pueda ver lo que necesita sin dañar a Marketing. Por supuesto que le hemos asegurado a los de Marketing —y a todas sus áreas funcionales— que, si ellos ponen algo de control, el sistema de base de datos les protegerá sus datos. Además les puedo probar que un buen sistema de base de datos les puede brindar muchas ventajas adicionales."

"Sandy, esto me suena bien. Si le presentas estas ideas a los grupos funcionales, estoy segura que estarán de acuerdo."

Quizás la diferencia más significativa entre un sistema basado en archivos y un sistema de base de datos es que los datos se comparten. Esto requiere un cambio importante en la mentalidad de los usuarios, que están acostumbrados a sentirse "dueños" de los datos resultantes de sus actividades cotidianas. Compartir los datos también requiere un cambio importante en la forma en que se manejan y administran los datos dentro de la organización. Esto se debe en parte al volumen de datos que es necesario organizar e integrar. Para ayudar a comprender este reto de usar una base de datos para compartir datos, echémosle un vistazo más de cerca a la naturaleza de los datos que la organización necesita compartir. Se considerarán tres formas de compartir los datos: (1) entre las unidades funcionales; (2) entre los niveles de dirección; (3) entre localidades que están geográficamente dispersas.

Compartir datos entre unidades funcionales

El término *compartir datos* sugiere que personas en áreas funcionales diferentes compartan un grupo común de datos, cada cual para sus propias aplicaciones. Si no se compartiesen los datos, la gente de Marketing pudiera tener sus archivos de datos, los grupos de adquisición los suyos, los grupos de contaduría los suyos y así sucesivamente (Figura 3.1(a)). Cada grupo se beneficia sólo de sus propios datos.

En contraste, el efecto de combinar los datos en una base de datos produce sinergia: es decir, los datos combinados tienen más valor que la suma de los datos en los archivos por separado. Esto no sólo permite que cada grupo continúe teniendo acceso a sus datos, sino que, bajo límites razonables de control, también pueda tener acceso a los otros datos (Figura 3.1(b)). En un entorno así, el departamento de Marketing, por ejemplo, se beneficia porque tiene acceso a los datos de Adquisición, especialmente a la evaluación de los productos, lo que los provee de información invaluable para las campañas de Marketing. En compensación, Evaluación de Productos gana el acceso a Marketing y puede realimentarse sobre la satisfacción de los consumidores. A este concepto de combinar los datos para un uso común se le llama **integración de datos** (*data integration*).

integración de los datos. Combinar los datos para un uso común.

Compartir datos entre diferentes niveles de usuarios

Diferentes niveles de usuarios necesitan compartir datos. Normalmente se distinguen tres niveles de usuarios: personal, mandos intermedios y ejecutivos. Estos niveles corresponden con los tres diferentes tipos de automatización de los sistemas de negocios que han evolucionado durante las tres últimas décadas: **procesamiento electrónico de datos** (PED)[3], **sistemas de información de gestión** (MIS)[4] y **sistemas de apoyo a la toma de decisiones** (STD)[5]. En Sprague (Sprague y Watson, 1989, p11-12) se da una descripción de éstos:

procesamiento electrónico de datos. Automatización computacional del papeleo al nivel operacional de una organización.

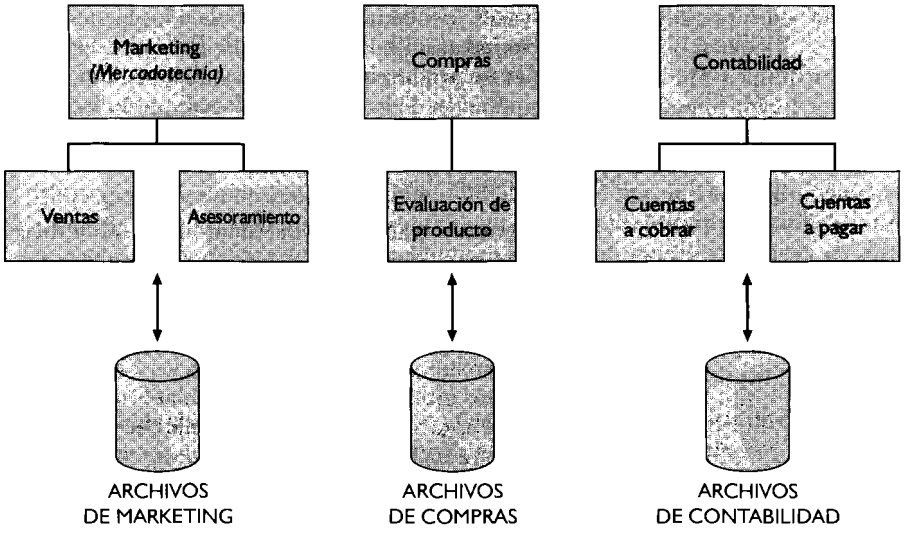
sistemas de información de gestión. Sistema automatizado cuyo foco de atención es la información para el nivel de dirección intermedio.

sistemas de apoyo a la toma de decisiones. Sistema automatizado que provee información estratégica para los altos directivos.

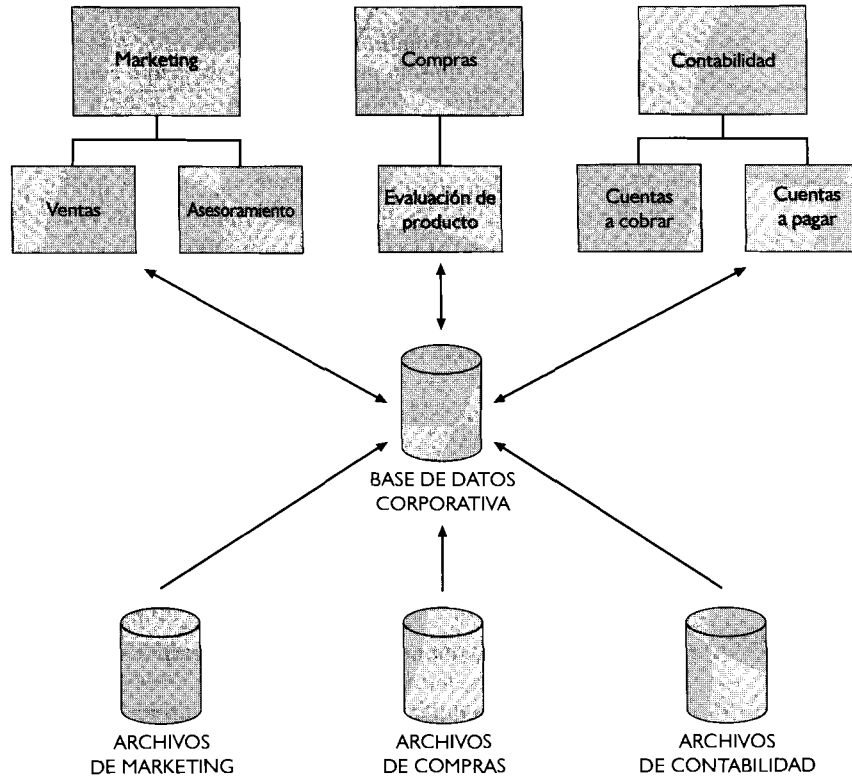
Los PED se aplicaron por primera vez a los niveles operacionales más bajos de la organización para automatizar el trabajo en papel. Sus características básicas incluyen:

- foco de atención en el nivel operativo del almacenamiento, el procesamiento y el flujo de los datos;
- procesamiento eficiente de las transacciones...
- informes resúmenes para los dirigentes...

El enfoque de los MIS elevan el foco de atención a las actividades de los sistemas de información, con un énfasis adicional en la integración y planificación de la función de los sistemas de información. En la práctica, estas características de los MIS incluyen:



[3] En inglés, *Electronic Data Processing (EDP)* (N. del T.).
[4] Del inglés, *Management Information System*, por ser siglas establecidas se dejarán en inglés (N. del T.).
[5] En inglés, *Decision Support System (DSS)* (N. del T.)

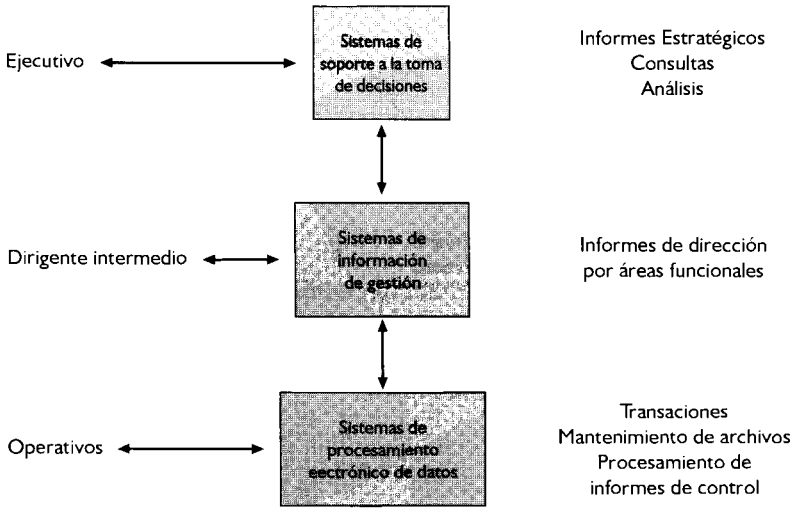


- foco en la información orientada a los mandos intermedios...
- una integración de las tareas de PED por sus funciones en los negocios, tales como MIS de producción, MIS de marketing, MIS de personal, etc.;
- generación de encuestas e informes, usualmente con una base de datos...

Un STD se centra en un nivel aún más alto dentro de la organización, con énfasis en las características siguientes:

- interés centrado en la decisión, orientado a dirigentes de alto nivel y ejecutivos que toman decisiones;
- énfasis en la flexibilidad, la adaptabilidad y la respuesta rápida....
- apoyo a los estilos personales de toma de decisión de los dirigentes.

Las interrelaciones de estos sistemas a los diferentes niveles de dirección se ilustra en la Figura 3.2. Estos niveles de usuarios y sistemas requieren naturalmente de tres tipos diferentes de datos. El usuario en el nivel operacional necesita datos para los procesos de transacciones. Esto pudiera incluir datos para las nuevas cuentas o cambios en las cuentas existentes, compras, pagos y otros. Estos datos detallados podrían luego resumirse para la información que se necesita en otros niveles más altos. Por ejemplo, el nivel de MIS podría utilizar resúmenes para indicar cuáles representantes de ventas fueron los más productivos, o cuáles fueron los menos. Los ejecutivos en el nivel más alto usan los sistemas de apoyo a la decisión para descubrir las tendencias a largo plazo que se pueden aplicar a su propia corporación, así como para identificar el entorno económico, social y político en el que deben operar. Los STD le ayudan a tomar decisiones, como la de construir una nueva fábrica, comenzar o clausurar una línea de producción y por el estilo. De modo que un STD usa los resúmenes de datos de su propia compañía, así como datos demográficos, de mercado y otros de fuentes externas.



Compartir datos entre diferentes localidades

Una compañía con diferentes sucursales distribuidas sobre una gran área geográfica tiene datos importantes. Compartir estos datos es un problema significativo. En la medida que las ramas de oficinas de IDP crecen, el intercambio de datos entre las oficinas locales y entre las ramas se va haciendo cada vez más inadecuado. Esto se ilustra en la conversación siguiente.

"Déjame ver si entiendo bien lo que tú piensas, Dick. Con oficinas regadas por todo el globo se va haciendo cada vez más y más difícil obtener los informes a tiempo."

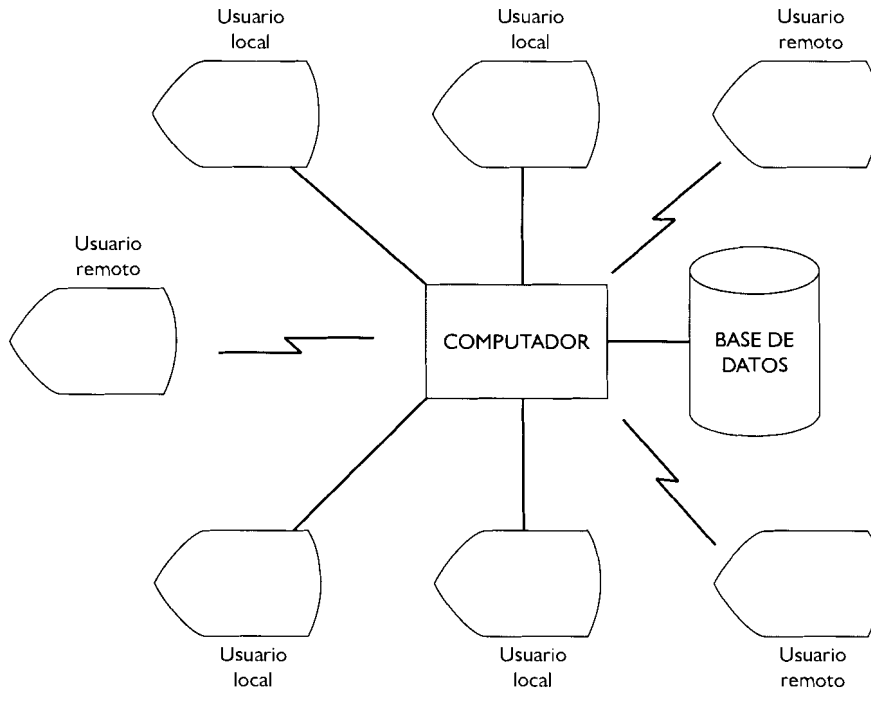
"Exacto. Por ahora ellos están enviando sus datos a Chicago y nuestra gente los introduce en el sistema, pero ya entonces es tarde para que podamos tomar una buena decisión en los negocios. Es más, las oficinas internacionales desean usar los datos allí mismo donde están y no tener que estar haciendo copias extras de éstos."

"Creo que es hora de que consideremos un sistema de base de datos distribuida. Es posible, con las tecnologías actuales, conservar los datos en las oficinas donde éstos se originan y luego poder accederlos desde cualquier parte del mundo vía líneas de datos. Podríamos hacer consultas en Sidney que usen datos que están en Bruselas y Buenos Aires. Esto sería sofisticado, pero pienso que podríamos hacerlo a un costo efectivo."

base de datos centralizada. Una base de datos que está físicamente situada en una única localización.

Una **base de datos centralizada** (*centralized database*) es una base de datos que está físicamente situada en un único lugar, controlado por una sola computadora (Figura 3.3). La mayoría de las funciones para las que se crean las bases de datos se llevan a cabo más fácilmente si la base de datos está centralizada. Es decir, es más fácil actualizar, hacer copias de seguridad, consultar y controlar el acceso a una base de datos, si sabemos exactamente dónde está y cuál es el software que la controla.

El tamaño de la base de datos y el computador sobre el cual reside no tienen relación con la base de datos donde está centralmente localizada. Una compañía pequeña con su base de datos sobre una computadora personal (PC) tiene una base de datos centralizada, de la misma forma que una gran compañía con muchos computadores, pero cuya base de datos está completamente controlada por una gran computadora (*mainframe*).



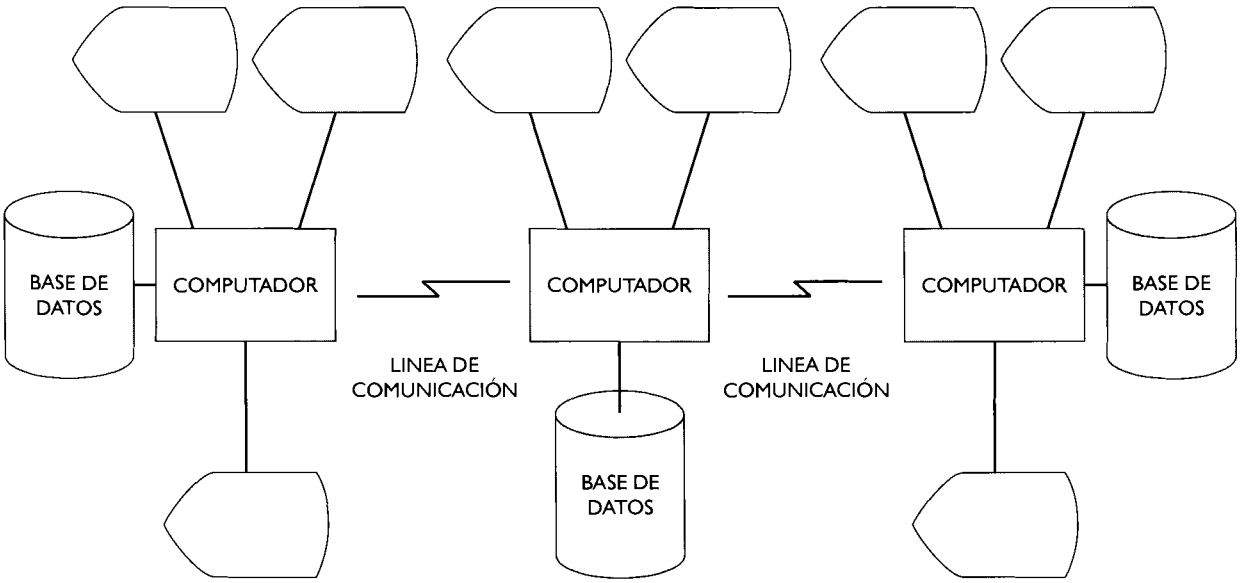
Sin embargo, en la medida en que la compañía crece, tiende a abrir nuevas ramas de oficinas con sus propias necesidades de datos. *International Product Distribution* es un buen ejemplo. Una vez que IDP empezó a abrir oficinas por todo el mundo, los directivos empezaron a darse cuenta que no podían contar con todos los informes a tiempo elaborados por el sistema de base de datos central de las oficinas de Chicago. Además, los directivos locales deseaban conservar los datos de su interés en sus propias bases de datos en lugar de en un sistema de base de datos ubicado a miles de kilómetros de distancia. Una vez que las tecnologías de las comunicaciones han mejorado, IDP ha comenzado un serio estudio para implantar un sistema de base de datos distribuida.

sistema de base de datos distribuida. Un sistema de base de datos compuesto de varios sistemas en sitios locales, conectados por líneas de comunicación.

Un **sistema de base de datos distribuida** (*distributed data base system*) está compuesto de varios sistemas de bases de datos operando en los sitios locales y conectados por líneas de comunicación (Figura 3.4). Una consulta o una actualización deja de ser un proceso simple controlado por un único módulo de software, sino que pasa a convertirse en varios procesos cooperando entre sí y ejecutando en sitios diferentes y controlado por módulos de software independientes. Claro está, para que un sistema de bases de datos distribuida funcione con efectividad, deben estar disponibles tecnologías adecuadas de comunicación y los SGBDs del sistema deben ser capaces de comunicarse entre sí con adecuadas interfaces con las facilidades de comunicación.

Los sistemas de bases de datos distribuidas son atractivos porque hacen posible la ubicación local de los datos: Los datos residirán en los lugares donde se necesitan con más frecuencia. Con la disponibilidad de potentes software de base de datos cliente/servidor para redes locales (LAN)[6] e incluso sobre PCs, es razonable crear sistemas distribuidos que permitan a los usuarios locales manipular los datos locales, mientras que al mismo tiempo se dan los medios para que usuarios no locales y dirigentes localizados centralmente puedan acceder a los mismos datos según lo requieran sus necesidades. Este enfoque mejora la relación

[6] Del inglés *Local Area Network* (N. del T.).



costo-efectividad y la autonomía local. El costo, sumándole el derivado de las comunicaciones, es más complejo de determinar para el sistema en su conjunto. Este es un problema que tienen que resolver los diseñadores del sistema. Otras ventajas y limitaciones de los sistemas de bases de datos distribuidas se estudiará en el Capítulo 12, que trata este tema en detalle.

El papel de la base de datos

Como ha demostrado este análisis, lograr el objetivo de compartir los datos es complejo. Con esto en mente, dediquémosle una segunda revisión a la definición de una base de datos efectiva. Como ya se ha dicho, una base de datos es una colección de datos interrelacionados, compartidos y controlados. Tanto el compartir como el controlar los datos se facilita mediante la *integración*. De este modo, esta definición contiene tres criterios para decir cuándo una base de datos es efectiva.

Primero, los datos deben estar compartidos. Como se ha visto, los datos pueden estar compartidos entre unidades funcionales, entre niveles de dirección y entre diferentes unidades geográficas.

Segundo, el uso de los datos debe estar controlado. El control se proporciona por un sistema de gestión de base de datos (SGBD) cuyas facilidades son administradas por un personal que se le conoce como administradores de bases de datos. En el Capítulo 1 se han destacado las facilidades de los SGBD y se volverán a tratar más extensamente en el Capítulo 13. Las funciones de los administradores de bases de datos serán discutidas más tarde en este capítulo y de nuevo en el Capítulo 11.

Tercero, los datos se integran de una forma lógica, de modo que se eliminen redundancias, que se resuelvan las ambigüedades en la definición y que se mantenga la consistencia interna entre los mismos. La estructura lógica de la integración es lo que hace que el compartir y controlar los datos pueda ser práctico cuando se trabaja a gran escala. Sin la integración, sería extremadamente difícil administrar y mantener la consistencia entre grandes cantidades de archivos diferentes. Las técnicas para lograr la integración mediante la estructuración lógica de los datos se presentarán en las Partes II y III.

▼ Planificación estratégica de bases de datos

Moverse de una situación en la que los datos son privados y fragmentados a una en la que los datos son compartidos es más fácil decirlo que hacerlo. Para tener éxito, los datos se deben ver como recurso colectivo y por lo tanto otros recursos colectivos se deben destinar a su desarrollo, su realización y su utilización en una o más varias bases de datos. Un elemento esencial en este proceso es la planificación de la base de datos. La **planificación de la base de datos** es un esfuerzo colectivo estratégico para determinar las necesidades de la organización para un extenso período de tiempo en el futuro. Una planificación estratégica exitosa de un proyecto de base de datos debe preceder al diseño e implementación de las bases de datos que satisfagan las necesidades de información de la organización.

planificación de la base de datos. Esfuerzo estratégico para determinar las necesidades de información de un extenso período de tiempo.

La necesidad de planificar la base de datos

La planificación de una base de datos está condicionada por las necesidades de información de la organización, que a su vez están determinadas por el plan de negocios de la compañía, tal y como se ilustra en la Figura 3.5. Por ejemplo, la corporación formula su plan estratégico de negocios para los próximos cinco años. Cumplir con los objetivos de este plan depende de la disponibilidad de cierto tipo de información que ha sido identificada. La información sólo se puede obtener si los datos, tal y como se identificaron en la planificación de la base de datos, están disponibles. Esto determina la necesidad de proyectos de desarrollo que creen nuevas bases de datos, o mejoren o integren bases de datos existentes.

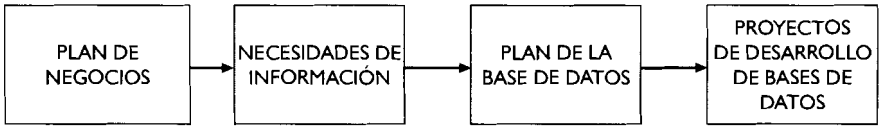
La planificación de las bases de datos tiene ventajas significativas. James (en Umbaugh, 1985, p. 6) cita varias ventajas de un plan formal de recursos de información; éstas incluyen:

- Expresa la comprensión que tienen los dirigentes de los recursos informativos...
- Identifica y justifica los requisitos de recursos... y ayuda a asegurar que los recursos estén disponibles...
- Identifica las oportunidades para una efectiva gestión de los recursos, incluyendo la colaboración entre los departamentos y divisiones de la organización...
- Especifica los planes de acción para alcanzar los objetivos...
- Puede... sensibilizar y servir de gran estímulo de cómo dirigir a los empleados en todos los niveles, para que centren sus esfuerzos en aumentar su productividad y hacerlos sentir que son parte genuina de la empresa.

El proyecto de planificación de la base de datos

La planificación estratégica de la base de datos empieza por la directiva de mayor rango. Ellos asignan los recursos e identifican el personal que participará en el proyecto. Con las comisiones de la dirección, los miembros de los equipos tienen los recursos necesarios para llevar a cabo con éxito un proyecto.

El equipo del proyecto debe tener una amplia experiencia en sistemas de información y en otras áreas funcionales de la compañía. Martin (1983) recomienda un grupo de cuatro miembros a tiempo completo, dos de sistemas de información y dos que estén familiarizados con la mayoría de las áreas de la compañía. Todos los miembros del equipo deben ser empleados cualificados y capaces, puesto que su trabajo tendrá un gran impac-

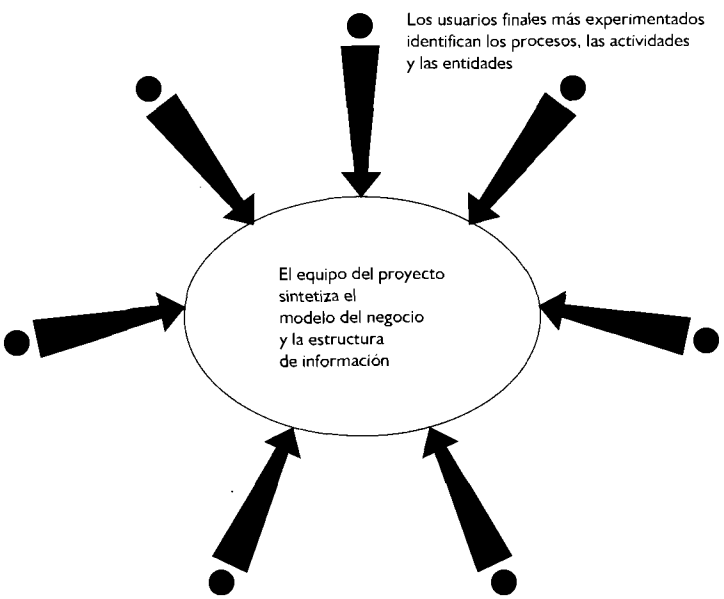


to en la organización para muchos años. Si ellos no tienen conocimientos de alguna metodología para llevar a cabo el estudio, podría emplearse a algún consultor externo que actúe como asesor para entrenar al equipo en una metodología conveniente. El líder del equipo no debe ser un consultor externo, sino un empleado permanente y posiblemente el jefe de administración de base de datos. En el caso con el que se comenzó este capítulo, Cordelia Molini, la administradora jefa de base de datos de IPD, fue la designada como líder de proyecto de IPD para la planificación estratégica de base de datos.

Durante el proyecto, tal equipo interactúa con los directivos de mayor nivel de cada una de la áreas primarias de usuarios, como se muestra en la Figura 3.6. Los usuarios finales de mayor nivel identifican los procesos principales, las actividades y las entidades que se usan en el procesamiento manual o automatizado de la información. El equipo del proyecto sintetiza estos datos en un modelo de información corporativa que se incluye como parte de un comprensivo plan de base de datos.

Martin aclara que, para que el proyecto cumpla sus objetivos dentro de la organización, éste no debería demorarse más de seis meses. Al cabo de ese tiempo, se debería entregar al director general un informe que cubra al menos los próximos cinco años. Este informe debe incluir el análisis de lo siguiente:

- Necesidades de información de las áreas funcionales.
- Necesidades de información de los diferentes niveles de dirección.
- Necesidades de información de las localidades geográficas.
- Un modelo de estas necesidades de información.
- Volúmenes anticipados de datos por localidad geográfica, proyectado para el período bajo estudio.
- Un estimado preliminar de los costos asociados a las actualizaciones del sistema.
- Recomendaciones para un desarrollo detallado de las nuevas o mejoradas bases de datos junto con sus planes.



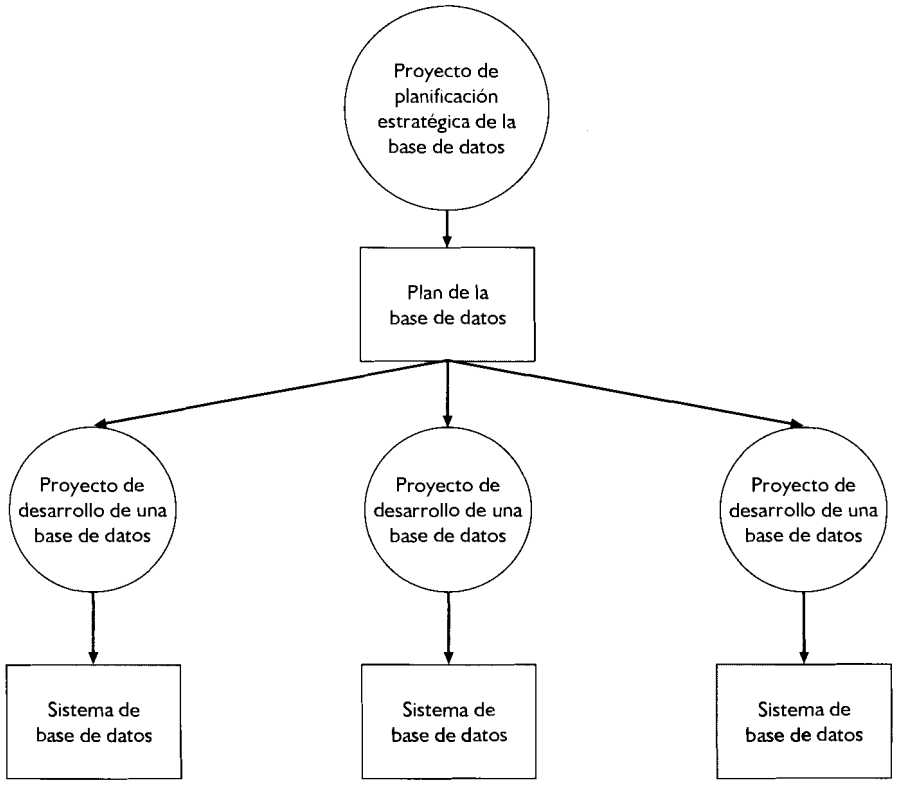
Fuente: James Martin, MANAGING THE DATA BASE ENVIRONMENT, © 1983, p. 658. Adaptado con permiso de Prentice Hall, Englewood Cliffs, NJ.

El equipo del proyecto no debe hacer un modelo de información detallado en este plan. Los modelos detallados se deben desarrollar en los proyectos subsiguientes de diseño de la base de datos. En lugar de esto, tal y como James denota, el equipo del proyecto debería identificar los elementos estables de la estructura de información de la organización, elementos que no se alteren con cambios organizacionales. De acuerdo con Martin, al final de los seis meses, el modelo de información debe estar al menos en un 90 por 100 finalizado. Esto implica que la mayoría de los elementos principales tendrían que haber sido identificados, por lo que el modelo tendrá un gran valor para una planificación estratégica.

El ciclo de vida del desarrollo de la base de datos (CVDBD)

El equipo de planificación estratégica puede concluir que la organización debería tener varias bases de datos en lugar de una única base de datos corporativa. Muchas compañías han llegado a esta conclusión debido a la diversidad de sus operaciones. La tarea de elaborar una única base de datos para todas las necesidades podría ser tan arriesgado y caro que el proyecto en sí nunca llegara a terminarse. En estos casos, comprensivas bases de datos en varias áreas bien definidas puede ser una mejor opción. Por supuesto, que ahora se dificultaría la comunicación entre estas bases de datos, pero esto se considera un peligro menor.

El plan estratégico de base de datos recomienda el número y el tipo de bases de datos que deberán desarrollarse, como se ilustra en la Figura 3.7. Este indica también un cronograma para estos proyectos. Después que el plan reciba la aprobación final, se puede



entonces llevar a cabo los planes para diseñar y realizar las bases de datos específicas. La metodología para hacer esto es el ciclo de vida del desarrollo de la base de datos.

ciclo de vida del desarrollo de la base de datos (CVDBD). Un proceso para el diseño, implementación y mantenimiento de una base de datos.

El **ciclo de vida del desarrollo de la base de datos (CVDBD)** incluye información recolectada para determinar las necesidades de los usuarios; el diseño del esquema de la base de datos (la estructura lógica) para satisfacer esas necesidades; la selección de un SGBD para dar soporte al uso de la base de datos; el desarrollo de programas para utilizar la base de datos y una revisión de las necesidades de información de los usuarios en el contexto de la base de datos desarrollada.

Puesto que este ciclo de vida constituye el área de mayor interés para este libro, se estudiará en detalle en la última parte de este capítulo. Es más, todo el resto del libro tiene que ver con el desarrollo de técnicas para llevar a cabo los pasos del diseño e implementación del ciclo de vida.

▼ Bases de datos y gestión de control

administración de la base de datos. Personal con la responsabilidad de controlar y proteger la base de datos.

Al ser un recurso de la compañía que es de un valor significativo, la base de datos requiere de protección y control. Esta responsabilidad usualmente se le asigna al **administrador de la base de datos (ABD).** El ABD debería coordinar el diseño de la base de datos, el entrenamiento a los usuarios para acceder a la base de datos, guiar el desarrollo e implementación de los procesos de seguridad de los datos, proteger la integridad de los valores de los datos y asegurarse de que el rendimiento del sistema es satisfactorio.

En una organización pequeña, una persona puede llevar a cabo todas estas responsabilidades. Sin embargo, con frecuencia estas funciones se le asignan a un grupo de personas. Esto es lo que ocurre en la grandes organizaciones, donde las responsabilidades ABD se dividen entre varias personas, dirigidas por un administrador jefe.

Las funciones del ABD incluyen:

- diseño de la base de datos
- formación del usuario
- seguridad e integridad de la base de datos
- rendimiento de la base de datos

Diseño de base de datos

diseño conceptual de bases de datos. Identificación de los elementos de datos, interrelaciones y restricciones para una base de datos.

restricción de valor. Una regla que define los valores permisibles para un dato en específico.

diseño físico de la base de datos. Determinación de los dispositivos de almacenamiento, métodos de acceso e índices para usar la base de datos.

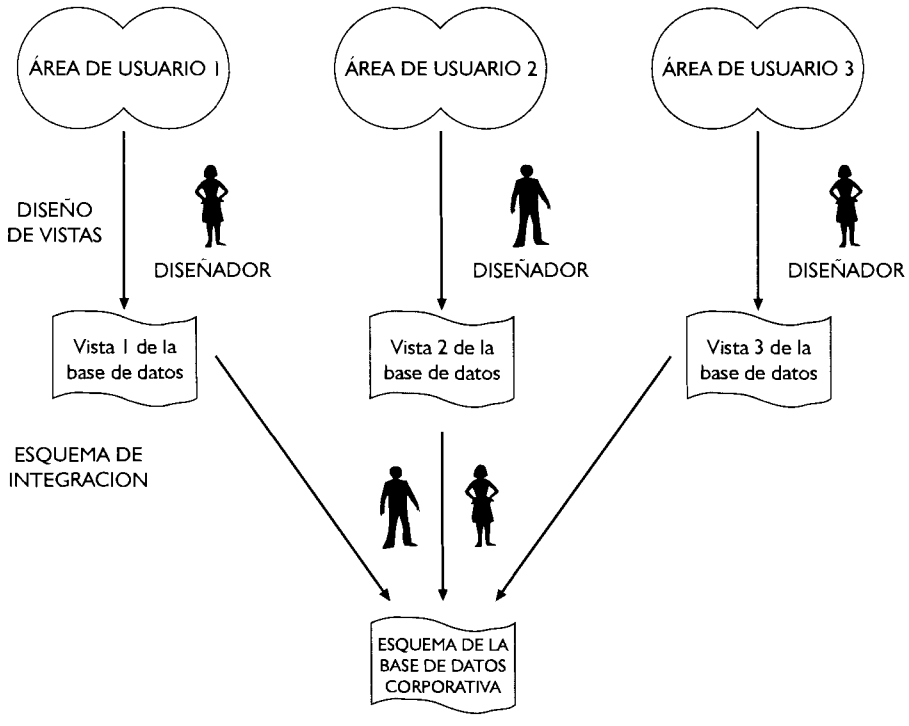
El **diseño conceptual de la base de datos** consiste primariamente en la definición de los elementos de datos que se van a incluir en la base de datos, las interrelaciones que existen entre ellos y las restricciones que se aplican a los valores de los datos. Una **restricción de valor** es una regla que define los valores permisibles para un determinado dato. El **diseño físico de la base de datos** determina la estructura física de la base de datos e incluye decisiones tales como qué métodos de acceso serán utilizados para recuperar los datos y qué índices se podrían incluir para mejorar el rendimiento del sistema.

Para llevar a cabo el diseño conceptual, el equipo de ABD debe incluir personal que sea experto en diseño conceptual, así como que tengan habilidades para tratar con grupos de usuarios. Los diseñadores del ABD trabajan con los usuarios de diferentes áreas y diseñan *partes* de la base de datos. Tales porciones de la base de datos se llaman **vistas** y están destinadas a ser utilizadas por el grupo original. Estas vistas luego deben integrarse en un **esquema** completo, el cual define la estructura lógica de toda la base de datos. Este proceso se ilustra en la Figura 3.8.

El proceso de diseño conceptual requiere la resolución de conflictos entre diferentes grupos de usuarios. Por ejemplo, diferentes grupos pueden usar el mismo término de forma contradictoria. O los grupos pueden ser celosos de sus datos y resistirse a que otros los accedan, como se ilustró al principio del capítulo con el conflicto entre el grupo de Marketing de IPD y el grupo de evaluación de producto. Se deben establecer controles razo-

vista. Una definición de una porción restringida de la base de datos; también llamada vista de datos.

esquema. Una definición de la estructura lógica de la base de datos completa.



nables para definir qué grupos pueden acceder a qué datos. El ABD debe ser capaz de negociar los conflictos que puedan surgir.

Un equipo de miembros técnicamente orientados es el que debe llevar a cabo el diseño físico; por lo tanto, deben conocer con precisión cómo el SGBD maneja el acceso a los datos y cuáles métodos de acceso serían más eficientes. Su objetivo es optimizar la combinación total de hardware, software y costo de los recursos humanos. Idealmente deben buscar cómo minimizar el tiempo de respuesta del sistema dentro de las restricciones de los costos de hardware y software.

El diseño y la implementación inicial de la base de datos debe estar seguido de un continuo mantenimiento de la estructura lógica y física de la base de datos. Cambios en los requisitos de los negocios, en las capacidades de software y hardware y en los volúmenes de procesamiento producen cambios tanto en el diseño físico como lógico de la base de datos.

El ABD ocupa un lugar estratégico en la definición y establecimiento de los estándares de la compañía. Puesto que el diseño de la base de datos se controla centralmente, el ABD puede definir estándares sobre cómo se deben poner los nombres; sobre los formatos de los datos; sobre los formatos de las pantallas, de los informes y de los archivos. Esto simplifica la documentación y los requisitos de entrenamiento y facilita la integración de los sistemas dentro de la compañía.

Las decisiones de diseño se documentan en los diccionarios de datos. El ABD controla el contenido de este diccionario de datos y registra en éste como metadatos los nombres de los elementos de datos, los archivos, las pantallas, las formas de los informes y otros. Como se señaló en el Capítulo 1, los metadatos en el diccionario de datos pueden ser consultados y manipulados. Sin embargo, su manipulación la controla con mucho cuidado el ABD, puesto que los datos de este diccionario son vitales para el funcionamiento adecuado del sistema de base de datos como un todo.

Formación del usuario

Muchas de las ventajas resultantes de compartir los datos sólo pueden manifestarse en la práctica si los usuarios entienden cómo usar las facilidades del SGBD. El ABD es responsable de educar a los usuarios en la estructura de la base de datos y en su acceso a través del SGBD. Esto se puede hacer con sesiones formales de entrenamiento, interactuando con los usuarios para crear las vistas de la base de datos, a través de manuales de usuarios y memorándums periódicos y mediante los centros de información de la compañía.

centro de información. Un área en la que los usuarios tienen facilidades para gestionar su propia información.

Un **centro de información** es un área donde se les da a los usuarios facilidades para gestionar su propia información. En este centro se pueden brindar paquetes auxiliares de manipulación de datos, así como entrenamiento y servicios simples de programación. El ABD proporciona el equipo a tales centros para proporcionar a los usuarios herramientas propias para resolver sus problemas de procesamiento de información, liberando con ello al equipo de los sistemas de información de estar respondiendo a las numerosas y "simples" demandas que son parte natural de las necesidades de información de cualquier compañía.

Seguridad e integridad de los datos

El concepto de combinar los datos de una organización en un lugar común accesible a todos tiene sus ventajas y sus desventajas. La ventaja obvia de compartir los datos puede desplazarse por la desventaja de que los datos se pueden trastocar o dañar por usuarios que no tengan una responsabilidad o autoridad sobre los mismos.

El ABD asigna la propiedad de los datos en una vista de la base de datos al grupo que lo originó. El grupo propietario puede entonces otorgar el acceso a los datos en la vista a otros miembros dentro de la organización. Este acceso se puede restringir a porciones de los datos, permitir **acceso sólo para recuperación** (*retrieve only access*), o permitir acceso y actualización. La información que tiene que ver con los derechos de acceso a los datos es mantenida por el ABD en el diccionario de datos.

acceso sólo para recuperar. Acceso a la base de datos pero sin poder hacer actualización

El acceso a la base de datos es finalmente controlado por un mecanismo de contraseñas (*password*), un usuario que intente tener acceso al sistema debe dar una contraseña que será validada por el sistema. El sistema le permite al usuario validado sólo los derechos de acceso que estén registrados en el diccionario de datos. El SGBD controla el acceso, a la vez que mantiene información estadística sobre los datos accedidos e introducidos por el usuario. El ABD es el responsable de la asignación de contraseñas y del control de los privilegios de acceso. De esta manera, el ABD reduce grandemente el riesgo de que un grupo dañe los datos de otro.

integridad de los datos. Precisión y consistencia de los valores de los datos en la base de datos.

La **integridad de los datos** tiene que ver con el problema de mantener la precisión y consistencia de los valores de los datos. Los mecanismos de seguridad, tales como las contraseñas y las vistas de los datos, protegen la integridad de los datos. Adicionalmente se pueden mantener en el diccionario de datos restricciones sobre los valores. Desgraciadamente, la definición de las restricciones de los valores y garantizar las mismas es una de las áreas de mayor debilidad de los sistemas actuales de gestión de bases de datos. Se pueden identificar muchas más restricciones, que se pueden definir sobre los SGBD. Por lo tanto, puede ser necesario hacer programas que verifiquen las restricciones para los nuevos datos que se introduzcan. Tal tipo de programas se le puede encargar razonablemente al equipo de ABD.

Los mecanismos de copias de seguridad y restauración soportados por el SGBD deben preservar los datos de cualquier fallo del sistema. Sin embargo, el ABD debe definir procedimientos para recuperar datos perdidos. Los usuarios deben saber qué hacer ante una caída del sistema, para tener que reintroducir sólo los datos necesarios.

Rendimiento del sistema de base de datos

Un sistema de base de datos al que accedan simultáneamente muchos usuarios puede responder a veces muy lentamente debido a que los problemas físicos asociados a los usuarios

que están compitiendo por los mismos recursos no son triviales. De modo que el equipo de ABD debería incluir personal técnicamente capacitado que pueda diagnosticar y resolver los problemas de tiempo de respuesta del sistema. La solución del problema puede implicar la compra de hardware, la reorganización física de los datos en el disco, la construcción de índices para el acceso rápido a grandes volúmenes de datos, o escribir software especial para mejorar el tiempo de acceso. A veces, el ABD puede incluso decidir mantener copias redundantes de los datos con tal de mejorar el rendimiento del sistema. Sin embargo, tales redundancias deben estar controladas para evitar problemas de inconsistencias en los datos.

▼ Riesgos y costos de las bases de datos

Es natural que en un libro sobre sistema de bases de datos se haga énfasis en sus características positivas. El efecto sinergético de las bases de datos, la estandarización del diseño de los datos, los procedimientos de manipulación, los beneficios de seguridad, el poder de los lenguajes de manipulación de datos y la amplitud de funciones de sistema que brindan los SGBD son todos beneficios positivos.

Sin embargo, los sistemas de bases de datos tienen también sus inconvenientes. Los principales son los conflictos organizacionales, los fracasos en el desarrollo del proyecto, las malas funciones del sistema, el aumento de los sobrecostos y la necesidad de disponer de un personal cualificado.

Conflictos en las organizaciones

Poner los datos en una base de datos común puede que no sea políticamente factible en algunas organizaciones. Ciertos grupos de usuarios pueden no estar de acuerdo en ceder el control de los datos, lo cual es necesario para extender la integración de los mismos. Es más, el riesgo que implica compartir los datos —por ejemplo, un grupo puede dañar los datos de otro grupo— y los problemas potenciales del sistema que pueden limitar los accesos de un grupo a sus propios datos pueden ser vistos más como un contratiempo que como un beneficio. Tales "problemas humanos" pueden impedir una implementación efectiva de un sistema de base de datos.

Fracasos en el desarrollo de proyectos

Los proyectos a desarrollar en un sistema de base de datos puede fracasar por varias razones. En ocasiones son los dirigentes los que no están en primer lugar convencidos del valor del sistema de base de datos. Un proyecto de base de datos que parezca muy largo pudiera ser cancelado.

Un proyecto muy grande en alcance puede ser imposible de completarse en un tiempo razonable. En este caso, los dirigentes y los usuarios comienzan a desencantarse y el proyecto fracasa. En este caso una mejor estrategia podría ser dividir el proyecto de base de datos en varios proyectos para desarrollar varias bases de datos o una base de datos en varias etapas.

Finalmente, durante el transcurso del desarrollo del proyecto, el personal clave puede abandonar inesperadamente la compañía. Si no puede encontrarse personal para reemplazarlo, entonces el proyecto pudiera no terminarse con éxito.

Malfuncionamiento del sistema

Cuando el sistema de cómputo no esté operativo, todos los usuarios directamente implicados en el acceso a la base de datos deben esperar a que el sistema vuelva a estar de nuevo en funcionamiento. Esto puede implicar una larga espera. Además, si el sistema o el *software* de aplicación falla, esto pudiera ocasionar un daño permanente en la base de datos. Por lo tanto, es muy importante evaluar cuidadosamente todo el software que tenga un efecto directo sobre la base de datos para estar seguros que esté todo lo libre de errores que

sea posible posible. Si la organización no usa una base de datos, no está expuesta a este riesgo, puesto que los datos y su software están distribuidos.

Un sistema de base de datos distribuida también reduce el riesgo de averías de hardware en un sistema de base de datos centralizado, ya que el sistema distribuido funciona sobre varias computadoras. Si una computadora del sistema falla, el sistema puede continuar operando en los otros computadores. Por supuesto, no se puede acceder a los datos que estén en la computadora que ha fallado, pero por lo demás el sistema continúa operando intacto.

Costes imprevistos

El enfoque de desarrollar y usar una base de datos puede requerir de inversiones tanto en hardware como en software. El hardware para correr un gran SGBD debe ser eficiente y puede requerir generalmente de más memoria principal y almacenamiento en disco que un simple sistema basado en archivos. También se requieren dispositivos de cinta para hacer respaldo rápido de la base de datos. Adicionalmente, el SGBD como tal puede ser caro. El SGBD usualmente incluye muchas facilidades, algunas de las cuales pueden no necesitarse en una organización en particular. No obstante, estos recursos pueden costar dinero y usar disco y memoria del sistema. Estos sobrecostos pudieran no siempre brindar los beneficios correspondientes. De modo que, para obtener algunos beneficios de un SGBD, una compañía pudiera tener que comprar cosas que tenga en planes de usar.

El SGBD puede también aumentar los costos de operación, ya que requiere mayor tiempo de ejecución. Por ejemplo, un sistema de aplicación que use un SGBD usualmente corre más lentamente que un sistema que no use un SGBD. El SGBD como tal es un sistema de propósito general construido para manejar una variedad de aplicaciones. Por consecuencia, pudiera no ser tan eficiente como un sistema que se desarrolle para tratar una aplicación en específico. Un sistema que requiera de un tiempo extremadamente eficiente de ejecución podría no ser capaz de usar con efectividad un SGBD que es de propósito general.

Cualquier sistema que use la actualización en línea (*on-line*) requiere de una copia de seguridad explícita para proteger los datos de averías del sistema. Sin embargo, un sistema de base de datos con un gran volumen de datos a compartir requiere de procedimientos más poderosos de respaldo y restauración. La explotación continua de las transacciones, incluyendo estar registrando copias "antes" y "después" de los registros de la base de datos, es parte de los sobrecostos del sistema y consume tiempo y recursos de hardware.

Necesidad de personal cualificado

La función de administración de la base de datos requiere de personal cualificado que sea capaz de coordinar las necesidades de los diferentes grupos, de diseñar las vistas, integrar las vistas en un único esquema, establecer los procedimientos que restauren los datos y afinar la estructura física de la base de datos para obtener criterios de rendimiento aceptables. Esta combinación de habilidades representa un sobrecosto humano en la adopción y uso de una base de datos. Es decir, una vez que las funciones de administración de la base de datos hayan sido establecidas, el costo del grupo de ABD es un gasto perenne.

Hay un riesgo adicional involucrado en la identificación del personal para la ABD, ya que si no se encuentra gente con las destrezas requeridas, la función de la ABD puede no realizarse adecuadamente. Esto podría provocar problemas significativos y pudiera incluso implicar el fracaso de la implementación de una base de datos. Debido al valor crítico de un sistema de base de datos, se debe tener mucho cuidado en escoger, para la administración de bases de datos, sólo a personal cualificado.

▼ Separar la representación lógica y física de los datos

En la primera mitad de este capítulo hemos tenido que ver con los aspectos de gestión de alto nivel relacionados con compartir los datos, planificación estratégica y control de los datos.

También se discutieron los riesgos y los costos de los sistemas de bases de datos. En el resto del capítulo se examinarán los aspectos relativos al diseño y la implementación concreta de la base de datos como tal. En esta sección se estudiará la separación de los aspectos lógicos y físicos de los datos. Luego se verá el enfoque de ciclo-de-vida para crear y mantener una base de datos. Esta estructura sentará las bases para la mayoría del resto del libro.

Como se dijo en el Capítulo 1, el acceso a los datos ha progresado durante las pasadas décadas desde los métodos físicos de los primeros procesamientos de archivos hasta las variadas formas de procesamiento de base de datos. Un aspecto importante de este desarrollo se encuentra en los sistemas de bases de datos relacionales de hoy día. Uno de los aspectos más importantes de la "revolución" relacional fue el concepto de separar la estructura lógica y la manipulación de los datos, tal y como lo concibe el usuario final, de la representación física que requiere el hardware de la computadora. Este importantísimo concepto ha sido muy discutido y es actualmente universalmente aceptado. Es esencial para la filosofía de la estructura de base de datos que propone el modelo ANSI/SPARC y que se estudiará a continuación.

Arquitectura de tres-niveles de una base de datos

La distinción entre la representación lógica y física de los datos fue reconocida oficialmente en 1978, cuando el comité ANSI/SPARC propuso un esqueleto generalizado para sistemas de bases de datos (Tsichritzis y Klug, 1978). Este esqueleto brinda una **arquitectura de tres-niveles**; los tres niveles de abstracción bajo los que podría verse una base de datos. Los tres niveles son el conceptual, el externo y el interno.

arquitectura de tres-niveles. Estructura estándar de base de datos consistente en los niveles conceptual, externo e interno.

El **nivel conceptual** es el nivel en el que se hace el diseño conceptual de la base de datos. El *diseño conceptual de la base de datos* implica el análisis de las necesidades de información de los usuarios y la definición de las clases de datos que se necesitan para satisfacer estas necesidades. El resultado del diseño conceptual es el *esquema conceptual*, una simple y lógica descripción de todos los elementos de los datos y sus interrelaciones.

nivel conceptual. Nivel estructural de la base de datos que define el esquema lógico de la base de datos.

El **nivel externo** consiste de las vistas que tiene el usuario sobre la base de datos. Cada grupo de usuarios tendrá su propia vista de la base de datos. Cada una de estas vistas nos da una descripción de los elementos de los datos y sus interrelaciones orientadas al usuario y de las cuales se compone la vista. Ésta se puede derivar directamente del esquema conceptual. La colección de todas las vistas de usuario forma el nivel externo.

nivel externo. Nivel estructural de la base de datos que define las vistas de los usuarios.

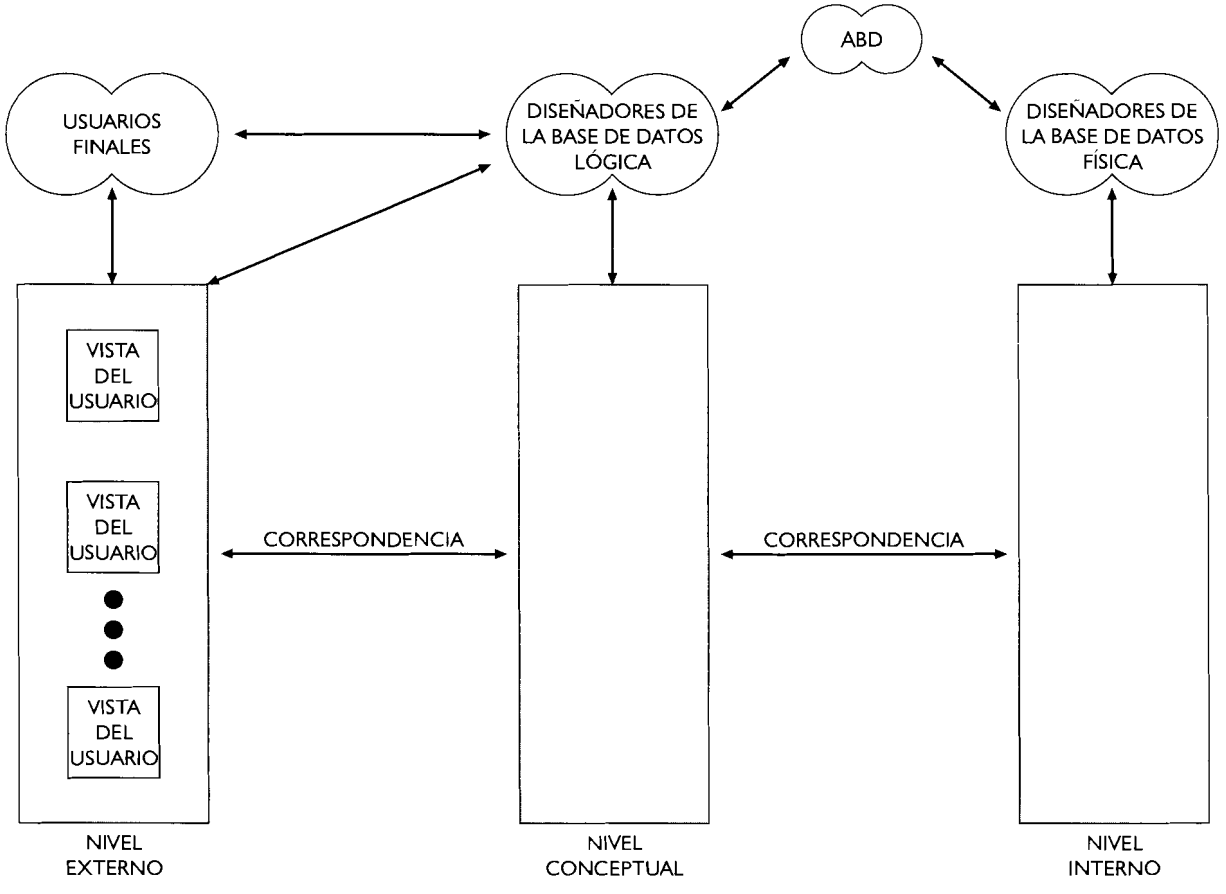
El **nivel interno** nos da la vista física de la base de datos —los dispositivos de disco, las direcciones físicas, los índices, los punteros y demás—. Este nivel es responsabilidad de los diseñadores de la base de datos física, quienes deciden cuáles dispositivos físicos contendrán los datos, cuáles métodos de acceso se usarán para recuperar y actualizar los datos y cuáles medidas se tomarán para mantener o mejorar el rendimiento de la base de datos. Ningún usuario tiene que ver con esta vista en calidad de usuario.

nivel interno. Nivel estructural de la base de datos que define la vista física de la base de datos.

Como señalan Elmasri y Navathe (1989), la implementación de estos tres niveles requiere que el SGBD haga corresponder cada nivel con el otro (Figura 3.9). Para entender esto, recordemos que la base de datos existe en realidad sólo a nivel interno. Para representar los datos en los niveles conceptual y externo del usuario, el sistema debe ser capaz de trasladar las direcciones físicas y apuntadores en sus correspondientes nombres lógicos e interrelaciones. Esta traducción también debe tener lugar en el sentido inverso —del lógico al físico—. El precio de este proceso de traducción es un sobrecosto mayor. El beneficio es la independencia de la representación lógica de los datos de su representación física.

Uso de la arquitectura tres-niveles en este libro

En este libro nos dedicaremos con alguna amplitud a la arquitectura de tres niveles ANSI/SPARC. En este capítulo y en las Partes II y III (Capítulos del 4 al 9) se estudiarán los niveles conceptual y externo y cómo se pueden desarrollar éstos como esquemas que satisfa-



gan las necesidades de los usuarios. En la medida que en este capítulo se revise el ciclo de vida del desarrollo de la base de datos, se verá cómo las vistas de usuario del nivel externo pueden desarrollarse e integrarse en un único nivel conceptual para la base de datos completa.

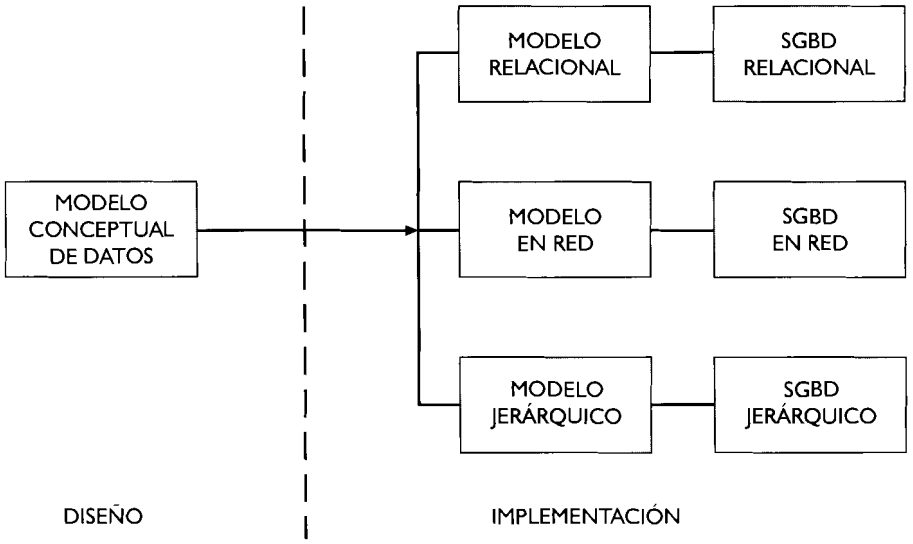
Al estudiar los niveles conceptual y externo en la Parte II se examinará la metodología de diseño conceptual y se mostrará cómo un diseño creado con esta metodología puede realizarse con los sistemas de gestión comerciales de bases de datos (Figura 3.10). Estos sistemas comerciales (basados en los modelos relacional, jerárquico y en red) son en sí mismos parte del camino entre el nivel puramente físico y el nivel puramente lógico. Ellos representan el estado actual del arte con respecto a las implementaciones de los SGBD.

En el Capítulo 10 se estudiará el nivel interno. Nuestro examen de este nivel incluye la descripción de las estructuras físicas que se requieren para manipular los datos de un modo eficiente. También se verá cómo estas estructuras físicas pueden reflejar los esquemas definidos en los niveles conceptual y externo.

ciclo de vida del desarrollo de la base de datos (CVDBD). Un proceso para el diseño, la implementación y el mantenimiento de un sistema de base de datos.

▼ Desarrollo de la base de datos

Esta sección se centrará en el procedimiento de desarrollo del esquema conceptual de la base de datos, identificando los datos que se incluirán en la base de datos y desarrollando los programas para actualizar y procesar los datos. Este proceso se denomina **ciclo de vida del desarrollo de la base de datos (CVDBD)**. Comenzaremos analizando los ciclos de vida tradicionales para el desarrollo de sistemas (CVDS).



Diseño de la base de datos y el CVDS tradicional

ciclo de vida del desarrollo de sistemas (CVDS). Un proceso para el desarrollo de sistemas.

Normalmente, los textos de análisis de sistemas describen un procedimiento para el desarrollo de sistema denominado **ciclo de vida del desarrollo de sistemas (CVDS)**. Este procedimiento consiste típicamente de pasos tales como el estudio de viabilidad, definición de requisitos, diseño del sistema, programación y prueba, así como revisión y mantenimiento.

Surge una pregunta natural: ¿Cómo encajar el diseño de base de datos con todo esto? Meyer (1988) argumenta que para incrementar su efectividad, los desarrolladores de sistemas necesitan echar una mirada de cerca a los presupuestos del CVDS tradicional. El CVDS hace énfasis en la identificación de las funciones de negocios y en el desarrollo de los sistemas de aplicación para llevar a cabo estas funciones. Como Meyer destaca, los métodos del CVDS están basadas en un **enfoque orientado a funciones**. Es decir, estos métodos ven a los sistemas desde el punto de vista de las funciones que ellos llevaban a cabo en lugar de los datos que realizan a las funciones. Por esta razón, el análisis estructurado enfatiza en los diagramas de flujos, siguiendo el rastro del progreso de los datos a través de una secuencia de transformaciones y refinando a éstas a través de una serie de niveles. Lo mismo es cierto para el diseño estructurado, que ve a un sistema como una función que se descompone sucesivamente en niveles o subfunciones.

enfoque orientado a funciones. Ver a un sistema desde la perspectiva de las funciones que debe realizar.

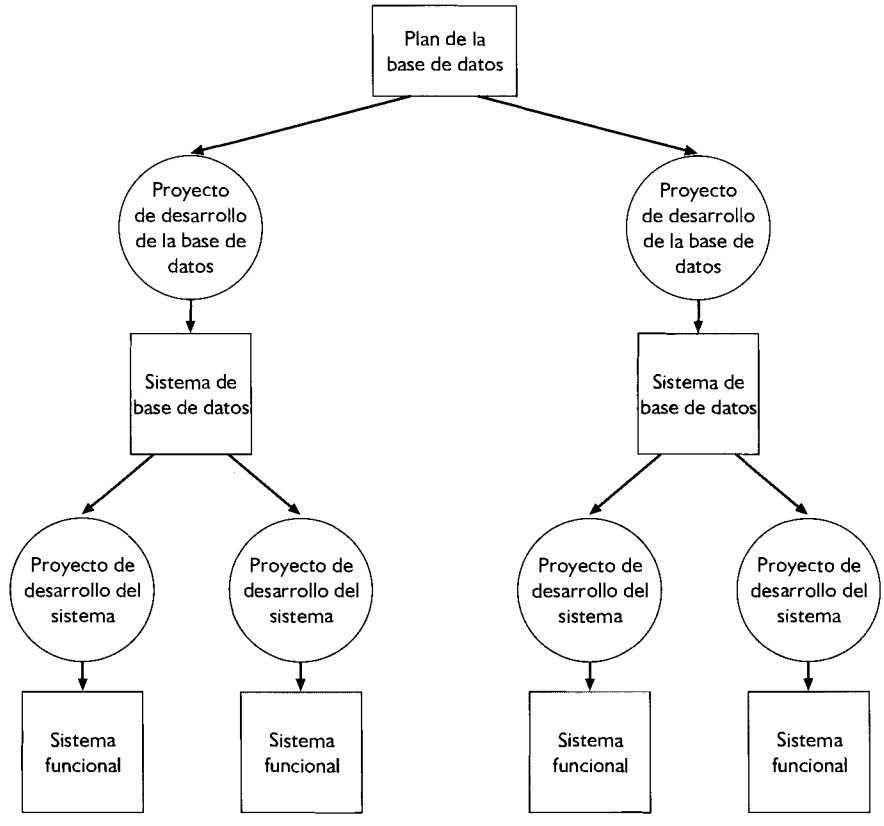
Concentrándose en las funciones, estos métodos infravaloran los datos y en especial la *estructura* de los datos que son manipulados por las funciones. El resultado, dice Meyer, es que estos sistemas tienen valor durante poco tiempo en relación con las necesidades a largo plazo de los usuarios. Esto sucede debido a que, al poco tiempo de haber instalado un sistema, las funciones se convierten en un subconjunto de las funciones que los usuarios realmente desean. Casi inmediatamente los usuarios ven una gran variedad de servicios adicionales que quisieran en el sistema. Estas necesidades causan problemas a los sistemas orientados a funciones, puesto que su diseño puede requerir de una revisión importante para poder acomodar a las funciones adicionales.

enfoque orientado a datos. Centra la atención en el análisis de los datos utilizados por las funciones.

En contraste, el **enfoque orientado a datos** centra el foco de atención en el análisis de los datos utilizados por las funciones. Esto tiene dos ventajas: (1) Los datos son una parte considerablemente más estable de los sistemas que las funciones. Esto es debido a que un conjunto determinado de elementos datos se puede combinar en una gran variedad de formas para obtener las respuestas a varias posibles preguntas. Si viéramos cada posible pregunta como una función del sistema, es fácil demostrar que el conjunto de posibles funciones sería siem-

pre mucho más grande que el número de campos de datos. (2) Como se verá en la Parte II, la propia estructura de un esquema de base de datos requiere de un análisis sofisticado de las clases de datos y de sus interrelaciones. Una vez que se haya construido un esquema de base de datos que sea lógico, podrían diseñarse la cantidad que se quiera de sistemas funcionales para sacar provecho del esquema de datos. Sin embargo, sin un esquema tal, la base de datos sólo podría ser útil para una única aplicación. Por tanto, el enfoque orientado a funciones puede ser bueno para el desarrollo a corto plazo, pero pierde su valor real a largo plazo.

Se ve pues que usando un enfoque orientado a datos, los datos pasan a ser los cimientos sobre los cuales se puede construir una gran variedad de sistemas funcionales diferentes. La Figura 3.11 amplía el plan de base de datos de la Figura 3.7, mostrándonos como éste se fundamenta en uno o más sistemas de bases de datos, los cuales a su vez se fundamentan en múltiples sistemas funcionales.



El ciclo de vida del desarrollo de la base de datos (CVDBD)

La Figura 3.12 muestra el ciclo de vida del desarrollo de una base de datos (CVDBD), que consiste en seis etapas:

1. Planificación preliminar
2. Estudio de viabilidad
3. Definición de requisitos
4. Diseño conceptual
5. Implementación
6. Evaluación y mantenimiento de la base de datos

FIGURA 3-12. CICLO DE VIDA DE DESARROLLO DE UNA BASE DE DATOS

Como puede verse, este ciclo de vida difiere del CVDS sólo en lo que enfatiza: El CVDBD se concentra en el desarrollo de una *base de datos* comprensiva y en los programas necesarios para procesar la misma. En el resto de esta sección describiremos las tareas principales de cada etapa, ilustrándola con la experiencia de la Corporación Zeus.

El caso de la Corporación Zeus

La Corporación Zeus se originó como un proyecto que fue terminado mientras los hermanos Blue estaban pasando un curso de gestión de administración de negocios[7] *en una de las escuelas de negocios del mediooeste. El proyecto centra la atención en la creación de una compañía para fabricar y comercializar calzado de carreras. Tanto Bill como Steve Blue fueron corredores de fondo y han continuado corriendo después de haber terminado el bachillerato. Como parte del curso de estrategia de negocios, ellos desarrollaron un plan para fabricar y distribuir calzado de carreras. A medida que empezaron a entrar en detalles, sus deseos iniciales de simplemente cumplir con los requisitos del curso fueron creciendo. En ese tiempo, muy pocas compañías comercializaban genuinos zapatos de carrera y ellos sentían que el mercado sería receptivo. Cuando su instructor los incitó, decidieron probar sus ideas creando la Corporación Zeus.*

Los hermanos alquilaron un local, compraron el equipamiento básico y una reserva inicial de materias primas. Su primer gran reto —hacer que las zapaterías aceptaran un producto nuevo y no probado— fue vencido ampliamente porque le regalaron zapatillas gratis a una serie de corredores de elite, que dieron su testimonio gratuitamente. Después de dos años pobres, la calidad de su producto comenzó a pagarse y los ingresos se duplicaron en los dos años siguientes y siguieron creciendo en los años subsiguientes hasta alcanzar un nivel de alrededor de los $50 millones.

Los sistemas de Zeus han evolucionado básicamente en la medida que la propia compañía fue creciendo, desarrollándose nuevas aplicaciones en tanto se iban identificando nuevas necesidades. Dos años atrás, un consultor ayudó a la compañía a establecer un plan de negocios a largo plazo y un plan correspondiente para la gestión de los sistemas de información (MIS). Una parte central del MIS fue el desarrollo de un sistema de base de datos colectivo.

[7] En el original inglés, la sigla MBA (*Management Bussiness Administration*) (N. del T.).

Planificación preliminar

planificación preliminar. Planificación de una base de datos que ocurre durante el proceso de planificación estratégica de la base de datos.

La **planificación preliminar** de un sistema de base de datos en específico tiene lugar durante el proyecto de planificación estratégica de la base de datos. Después que empieza el proyecto de implementación de la base de datos, el modelo general de información producido durante la planificación de la base de datos es revisado y mejorado si es necesario. Durante este proceso, la firma recoge información para responder a las preguntas siguientes:

1. ¿Cuántos programas de aplicación están en uso y qué funciones realizan?
2. ¿Qué archivos están asociados con cada una de estas aplicaciones?
3. ¿Qué nuevas aplicaciones y archivos están bajo desarrollo?

Esta información puede usarse para establecer las interrelaciones entre las aplicaciones en curso y para identificar los usos que hacen de la información las aplicaciones. También ayuda a identificar futuros requisitos del sistema y para apreciar los beneficios económicos de un sistema de base de datos. Esto se documenta en un modelo conceptual de datos generalizado.

Estudio de viabilidad

estudio de factibilidad. Parte del CVDBD que determina la viabilidad económica, tecnológica y operativa de la base de datos.

Un **estudio de viabilidad** implica la preparación de un informe con las características siguientes:

1. Viabilidad tecnológica. ¿Hay tecnología adecuada disponible para dar soporte al desarrollo de la base de datos?
2. Viabilidad operacional. ¿Tiene la compañía personal, presupuesto y experticidad interna para hacer que un sistema de base de datos tenga éxito?
3. Viabilidad económica. ¿Se pueden identificar los beneficios? ¿Los beneficios costearían el sistema que se desea? ¿Se pueden medir los costos y los beneficios?

El estudio de viabilidad de Zeus fue realizado por un equipo interdepartamental formado por un analista de sistema, un ingeniero, un especialista de Marketing *(Mercadotecnia)*, un supervisor de producción, un analista financiero y un especialista de base de datos. Ellos estuvieron conducidos por un comité asesor corporativo de MIS formado por los dirigentes de alto nivel de cada una de las áreas funcionales. El comité asesor aprobaba cada fase antes de que se comenzase una nueva fase. El foco en cada fase fue el siguiente:

viabilidad tecnológica. Determinación de la disponibilidad de hardware y el software para el sistema de base de datos.

viabilidad operacional. Determinación de la disponibilidad de experticidad y del personal que se necesita para el sistema de base de datos.

viabilidad económica. Estudio de costo beneficio del sistema de base de datos propuesto.

1. La primera fue un estudio de **viabilidad tecnológica** para determinar si el hardware y el software estaban en disposición de dar el servicio que requerían las necesidades de información. Esto incluyó el análisis de si las capacidades y recursos estaban ya presentes en la compañía o si habría que comprarlo, y de si sería necesario un entrenamiento. Esto llevó a la conclusión de que todo el hardware ya estaba presente en Zeus, pero que había que comprar el SGBD.
2. **Viabilidad operacional.** Esta incluye el análisis de las habilidades y los requisitos laborales necesarios para desarrollar el sistema. Un análisis preliminar sugirió que aunque Zeus tenía analistas y programadores experimentados y competentes, se requeriría una buena dosis de entrenamiento. Las necesidades de entrenamiento alcanzaba a los usuarios y a algún personal de sistemas. El personal de la ABD fue identificado y entrenado previamente del personal de sistemas.
3. El estudio de la **viabilidad económica** se convirtió en un reto. Los beneficios esperados de instalar una base de datos fueron difíciles de cuantificar. Esta es justamente una experiencia típica. El equipo del proyecto abordó este reto buscando las respuestas a las preguntas siguientes:

a. ¿Cuándo se pueden esperar beneficios?
b. ¿Es políticamente factible compartir datos por departamentos usuarios?
c. ¿Qué riesgos se corren con la implementación de un sistema de base de datos?
d. ¿Qué aplicaciones se implementarán y qué beneficios se esperan de esas aplicaciones?
e. ¿Qué está haciendo la competencia?
f. ¿Cómo ayudaría el sistema de base de datos a cumplir los planes a largo plazo que tiene la corporación?

En lo referente al costo, los factores más simples de medir son el dinero que tiene que salir del bolsillo para pagar el hardware, el software y la programación. Aunque estos costos no están en su totalidad sujetos a la gestión de control, la firma pudiera disfrutar de ahorrar gastos si se establecen estándares antes de que el software de base de datos se instalase, pero no instalando un lenguaje de consulta si antes no se tienen datos suficientes que garanticen su uso efectivo.

El equipo del proyecto también mencionó costos ocultos asociados con la implementación de la base de datos. Este es un resultado típico de los cambios imperceptibles de la forma en que funcionan los sistemas. Por ejemplo, es fácil subestimar el tiempo que se necesita para integrar sistemas de aplicación independientes. Adicionalmente, cambios en el software pueden requerir mejoras no previstas en el hardware para poder asegurar el rendimiento adecuado. Tales cambios aumentan rápidamente los costos de desarrollo.

El resultado de cada examen de viabilidad fue favorable y el comité asesor aprobó seguir adelante con la definición de los requisitos.

Definición de requisitos

definición de requisitos. Determinación de los requisitos de información de las áreas administrativas y funcionales.

La **definición de los requisitos** involucra a la definición del alcance de la base de datos, la identificación los requisitos de información de las áreas funcionales y administrativas y la determinación de los requisitos de software y el hardware. Los requisitos de información se determinan por las respuestas a cuestionarios, entrevistas con los directivos y usuarios de oficina y los informes y formularios que están en uso actualmente. El modelo de información general que se crea durante la planificación de la base de datos se expande a modelos para cada una de las áreas funcionales. Estas son las bases para el diseño detallado de la base de datos que se llevará a cabo en la etapa siguiente. Los resultados de esta etapa se describen en las cuatro tareas siguientes.

Aunque no todas las firmas siguen pasos idénticos, hay algunas nociones básicas que están identificadas con una definición exitosa de los requisitos. En Zeus se completaron las tareas siguientes:

1. El alcance del sistema de base de datos se definió analizando los requisitos de información de los directivos. El equipo también consideró si la base de datos debería ser distribuida o centralizada y qué necesidades de teleproceso se necesitan. Esto produjo una breve narración que describía el alcance del sistema. Se decidió que una base de datos centralizada se ajustaba mejor a la mayoría de las áreas funcionales.
2. Los requisitos de usuarios a nivel administrativo y operacional se documentaron con un modelo de información generalizado para cada área funcional, a la par que las definiciones de los sistemas de aplicación necesarios para satisfacer tales requisitos. Los modelos de información necesarios para cada área funcional fueron modelos conceptuales de datos (que se discuten en el Capítulo 4). También fueron documentados los requisitos de los usuarios a partir de narraciones de las entrevistas con los usuarios, informes y formularios impresos (Figura 3.13) y respuestas a encuestas (Figura 3.14)

Corporación Zeus
Pez, 27 (Granada)

Orden de compra

Número de orden 1848
Fecha 21/06/97

Fecha 7/07/97	Vendedor		Departamento	Agente compras
	Código	Nombre	Inventario partes	Mackoy As.
	215	Zapatos		

N.º Parte	Descripción	Cantidad	Precio unidad	Precio final
4831	Cordones	5000	.17	850.00
4922	Plantillas	1000	2.30	2300.00
			Total	3150.00

SUS FUNCIONES

Describa su área de responsabilidad. _____
¿Cuáles son sus principales obligaciones que requieren información de aplicaciones computacionales? _____

USO DE LA INFORMACIÓN

1. ¿De qué aplicaciones recibe información? _____
2. ¿Con cuánta frecuencia recibe la información? _____
3. ¿Qué es lo que se hace con esta información? _____
4. ¿Qué precauciones de seguridad debe tomar con respecto a la información? _____
5. ¿Para qué aplicaciones rinde datos? _____
6. ¿Están contemplados cambios para algunas de sus actividades actuales que involucren a alguna de las informaciones anteriores? Por favor, descríbalos brevemente _____

3. Los requisitos generales de hardware/software se establecieron junto con los niveles de rendimiento que éstos deben satisfacer. Las consideraciones en este área incluyen el número de usuarios que acceden normalmente al sistema, el número de transacciones que se introducen en el sistema cada día y el volumen de impresión que se requeriría. Esta información se utilizó para determinar el tamaño y tipo de computadora y el tipo de SGBD que se necesitaba, así como el volumen de espacio en disco y las capacidades de impresión. Como resultado de esto se elaboró un informe descriptivo con figuras, ilustrando la configuración de hardware y el software.
4. Se elaboró un borrador de plan de fase-tiempo para el desarrollo del sistema, incluyendo la identificación de las aplicaciones iniciales. Los principios conductores fueron: (1) Las aplicaciones deberían ser relativamente pequeñas y no críticas para limitar el impacto de cualquier problema que surgiese al introducir la base de datos; o (2) éstas deberían estar dirigidas a usuarios muy receptivos al desarrollo del sistema de base de datos.

Diseño conceptual

diseño conceptual. Creación del esquema de nivel conceptual para la base de datos.

La etapa de **diseño conceptual** crea el esquema conceptual de la base de datos. Se desarrollan las especificaciones hasta el punto en que puede comenzar la implementación. Durante esta etapa se crean modelos detallados de las vistas de los usuarios y se integran en un modelo conceptual de datos que registra todos los elementos colectivos que se deben mantener en la base de datos.

En Zeus el diseño conceptual se centró en el desarrollo de un modelo que reflejase la realidad de la firma. El desarrollo del modelo fue guiado por la información contenida en los manuales de política y procedimiento, así como en las guías y modelos generalizados para las áreas funcionales a partir de la definición de requisitos. A través de profundas entrevistas con los usuarios y un cuidadoso examen de los formularios de la compañía, se perfeccionaron estos modelos. Luego se integraron como parte del diseño conceptual y conviertieron en la base para la implementación.

La tarea del modelado de diseño conceptual fue un componente clave para las recomendaciones que hacían los consultores del CVDBD. Al principio, Steve Blue, el vicepresidente ejecutivo de Zeus, estaba escéptico con esta técnica. Éste interrogó cuidadosamente a la consultora Linda Kelly:

"Linda, ¿qué es lo que no está bien en la forma como hacemos las cosas ahora?

"Steve, tus métodos actuales están muy ligados a la forma de pensar orientada a archivos. Has heredado este enfoque de los primeros tiempos, cuando la tecnología no dejaba otra opción que diseñar las bases de datos como archivos. Pero el hecho es que simplemente los problemas no están estructurados de esa manera. Tú necesitas un enfoque que sea mucho más natural. En otras palabras, necesitas pasar de un enfoque orientado al aspecto físico a un enfoque orientado al aspecto lógico. Una metodología de diseño conceptual permitiría lograr esto."

"Ahora, espera un minuto. Tal y como yo lo entiendo, un sistema de base de datos relacional nos movería de la orientación física a la lógica."

"Una base de datos relacional nos ayudará en primer lugar con la manipulación de los datos, que ahora serán estructuras de datos. El modelo relacional como tal es orientado a archivos, puesto que las tablas relacionales son justamente como archivos. El modelo relacional está bien para trabajar con problemas que sean relativamente simples. Pero para hacer un análisis riguroso de los problemas complejos de negocios, se necesita una metodología más poderosa. Por ello es que yo recomiendo el diseño conceptual de base de datos."

"¿Qué quieres decir con conceptual*?*

Linda explicó: "La metodología de diseño conceptual también es llamada modelado semántico. Semántico quiere decir significado. En nuestro modelado capturamos el significado de los datos y las interrelaciones. En esta metodología de diseño pensamos en términos de objetos en lugar de archivos. Los objetos son cosas como los representantes de ventas, los productos, los directivos, las ventas y así. Aún más importante, pensamos en términos de interrelaciones con nombre entre los objetos. Por ejemplo, un directivo dirige a un representante de ventas. El directivo y el representante de ventas son objetos. Dirige *es una interrelación entre ellos. En tanto los problemas se van haciendo más complejos, el diseño conceptual ayuda a pensar con más claridad y a tener con más claridad en nuestra mente un gran número de complejas interrelaciones. Sin esta metodología nuestro sistema se volvería sin remedio en algo muy complejo."*

"Bien, estoy de acuerdo en que necesitamos manejar problemas más complejos. ¿Esto significa que no podemos usar un sistema de gestión de base de datos relacional?"

"No, no quiere decir eso. Piensa en el modelo conceptual como una herramienta que crea diseños claros que rápidamente se pueden convertir en modelos relacionales de alta calidad. Tendrás mejores sistemas, pero éstos continuarán corriendo encima del software relacional existente en los SGBD."

"Parece como si tuvieras sentadas todas las bases. Sigamos adelante con esto."

Primero, al ejecutar Zeus el CVDBD, el paso del diseño conceptual consistía en la creación de modelos conceptuales detallados de las vistas de los usuarios en diferentes áreas funcionales. Estos modelos juntos constituyen el nivel externo en la arquitectura tres-niveles. Segundo, estas vistas de usuario se integraron en un único modelo conceptual, dando entonces el esquema de la base de datos a nivel conceptual. Con todo esto terminado, la gente de Zeus estaba lista para pasar a la implementación.

Implementación

Implementación de la base de datos. Las etapas requeridas para cambiar un diseño conceptual a una base de datos funcionando.

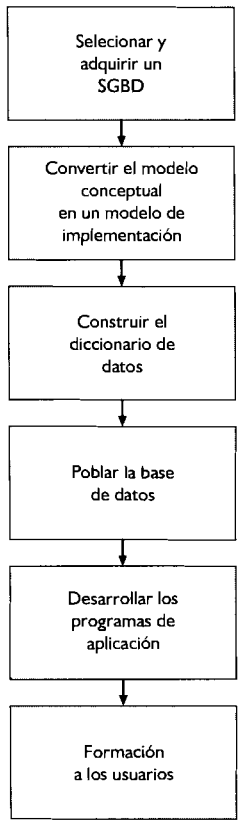
Durante la **implementación de la base de datos** se selecciona y adquiere un SGBD. Luego el modelo conceptual detallado se convierte al modelo de implementación del SGBD, se construye el diccionario de datos, se puebla la base de datos, se desarrollan los programas de aplicación y se entrenan los usuarios (Figura 3.15).

En Zeus, una implementación comienza con la selección y adquisición de un SGBD. Por varias razones ellos escogieron un SGBD relacional. El próximo paso fue hacer corresponder el modelo conceptual de la firma con un modelo de implementación relacional usando los procedimientos que se describen en el Capítulo 5. Luego se definieron las estructuras de la tabla resultante para el SGBD. Esto se hizo usando el lenguaje de definición de datos (LDD) suministrado por el SGBD para desarrollar el diccionario de datos.

La construcción del diccionario de datos (DD) es un paso crucial en la implementación porque el DD es un repositorio central de definiciones de estructuras de datos en la base de datos. Ya que éste contiene información sobre la autoridad de acceso, las reglas de seguridad y los controles respectivos, el DD actúa como un centro de control del sistema. Este se usa para imponer los estándares de datos y eliminar los problemas de coordinación de los datos compartidos entre las aplicaciones.

El paso siguiente fue cargar la base de datos, introduciendo los archivos de datos que ya tenía Zeus. Esto se hizo con programas de conversión que usaron el lenguaje de manipulación de datos (LMD, **DML** en inglés) suministrado con el SGBD.

Luego el equipo revisó las vistas de usuario y las aplicaciones que usarían la base de datos. Para algunos usuarios se decidió que un programa especial de aplicación escrito por el equipo de programación brindaría la forma más simple de acceso a la base de datos con la menor cantidad de problemas de seguridad. Esto la mayoría de las veces es cierto cuando se trata de oficinistas y otros empleados que registran los negocios diarios de la compa-



ñía. El equipo consideró que a otros usuarios era mejor formarlos en el uso del lenguaje de manipulación de datos para que accedieran directamente a la base de datos.

El paso final fue desarrollar procedimientos para usar la base de datos y establecer las sesiones de formación en estos procedimientos y las demás facilidades del sistema.

Evaluación y perfeccionamiento del esquema de la base de datos

La evaluación implica entrevistas con los usuarios para determinar si se han omitido algunos datos necesarios. Se hacen los cambios que se necesiten. Con el tiempo el sistema se mantiene, introduciéndole mejoras y añadiéndole nuevos programas y elementos de datos en la medida que cambian y se amplían las necesidades del negocio.

▼ Construir capacidades en el desarrollo de la base de datos

Como muestra la Tabla 3.1, la mayor parte de este libro está dedicada a ayudarle a desarrollar capacidades en la ejecución del ciclo de vida del desarrollo de la base de datos. Por tanto, la parte II cubre el diseño de base de datos. Los Capítulos 4 y 5 describen el proceso y la metodología del modelado conceptual de datos, que es esencial para una construcción *exitosa* de una base de datos con fundamento lógico y muestran cómo el modelo conceptual se puede convertir a un modelo de implementación relacional. La Parte III se

Paso del CVDBD	*Capítulo*
Planificación preliminar	3
Estudio de viabilidad	3
Definición de requisitos	3,4
Diseño conceptual	2,3,4,9
Implementación:	
Selección del SGBD	13
Conversión al modelo de implementación	2,5,7,8,9,15,16
Modelo físico	10
Desarrollo del diccionario de datos	12
Desarrollo de programa de aplicación	2,7,8,9,15,16
Control de la gestión	11,12

centra en la implementación de base de datos relacionales. Los Capítulos 6, 7, 8 y 9 introducen los lenguajes relacionales, teóricos y aplicados y analizan las facilidades disponibles en los sistemas cliente/servidor. El Capítulo 10 concluye con una explicación del modelo físico.

La Parte IV amplía el tema de la implementación de bases de datos, discutiendo la administración del entorno de la base de datos. Ésta trata con los temas de gestión, tales como la administración de la base de datos, la seguridad y la integridad (Capítulo 11), las bases de datos distribuidas (Capítulo 12) que fueron ya mencionadas en este capítulo. La selección de un SGBD se trata en el Capítulo 13. La Parte V contiene un capítulo dedicado a estudiar temas avanzados de sistemas basados en el conocimiento y sistemas orientados a objeto (Capítulo 14). La parte final, la Parte VI, cubre los modelos de base de datos legales, en red (Capítulo 15) y el jerárquico (Capítulo 16).

En la medida en que se estudie este material se desarrollarán nuevas capacidades en el diseño de base de datos y en la manipulación de los datos que serán esenciales en un entorno de base de datos para los negocios. También se comprenderán los aspectos de gestión que afectan significativamente muchas de las decisiones que tienen que tomarse con relación a los sistemas de bases de datos. El material de apoyo que se da en este libro le ayudará a comprender el contexto de los sistemas de bases de datos en el mundo de los negocios de hoy. Al adquirir estos conocimientos, comprendiendo los aspectos claves de la gestión y familiarizándose con el contexto de las bases de datos, el usuario estará equipado para desenvolverse en los entornos avanzados de los sistemas de gestión de información.

En este capítulo hemos examinado el contexto organizacional en el cual funciona un sistema de base de datos. En la primera mitad del capítulo se vio que el concepto de compartir los datos se puede ver desde tres perspectivas —entre diferentes áreas funcionales, entre diferentes niveles de dirección y entre diferentes localidades geográficas—. Se discutió el concepto de planificación estratégica de la ABD y se indicó cómo esto establece los fundamentos para todos los sistemas de bases de datos a usar en la organización. Luego se examinó la función de administración de la base de datos, destacando la responsabilidad de este grupo en: (1) el diseño físico y lógico de la base de datos; (2) la formación a los usuarios en la estructura y en los procedimientos requeridos para accederla y actualizarla; (3) la protección de la base de datos ante un mal uso, acceso inapropiado y daños accidentales; (4) el aseguramiento de que el sistema como un todo funcione a un nivel aceptable. Por último se analizaron algunos de los riesgos y costos de un enfoque de bases de datos en una organización, incluyendo los conflictos organizacionales de compartir los datos, los

fallos potenciales en el desarrollo del proyecto de base de datos, mal funcionamiento en el sistema de base de datos y el daño consiguiente en la base de datos, el incremento de los costos y la necesidad de tener un personal sofisticado para el desarrollo de la base de datos y para su operación.

La última mitad del capítulo se dedicó a la discusión del desarrollo de la base de datos. Se explicó la arquitectura de tres-niveles de ANSI/SPARC. También se contrastó el enfoque orientado a funciones del ciclo de vida tradicional en el desarrollo de sistemas (CVDS) con el enfoque orientado a datos del ciclo de vida del desarrollo de la base de datos (CVDBD). Se destacaron los pasos del CVDBD y se describieron las actividades incluidas en cada paso.

La arquitectura de tres-niveles describe una base de datos como consistente de los niveles conceptual, externo e interno. El nivel conceptual es el esquema lógico que define a la base de datos completa desde la perspectiva de la organización. El nivel externo consiste en las diferentes vistas, mediante las cuales los usuarios a lo largo de la organización entienden, acceden y actualizan la base de datos. El nivel interno es la definición física de la base de datos mediante la cual el SGBD controla y actualiza la base de datos.

El CVDS tradicional ve a los sistemas desde el punto de vista de las funciones que éstos realizan. Puesto que los elementos de datos tienden a ser más estables que las funciones, la estructura de la base de datos debería desarrollarse antes que el desarrollo de sistemas funcionales. Un sistema de base de datos desarrollado adecuadamente sirve de base para muchos sistemas funcionales que puedan basarse en él.

El CVDBD consta de seis pasos: planificación preliminar, estudio de viabilidad, definición de requisitos, diseño conceptual, implementación y evaluación y mantenimiento de la base de datos. La planificación preliminar tiene lugar durante la planificación estratégica de la base de datos. La información del plan de la base de datos se revisa y se presenta como parte del CVDBD. Se lleva a cabo el estudio de viabilidad para determinar la viabilidad tecnológica, operacional y económica. La definición de requisitos determina los requisitos de información para la dirección, así como para aquellas áreas funcionales que se sirven de la base de datos. El diseño conceptual termina en un comprensivo modelo conceptual de datos, que nos da un esquema lógico detallado de toda la base de datos. Durante la implementación puede adquirirse un SGBD, el diseño conceptual se convierte a un diseño de implementación, se construye el diccionario de datos, se carga la base de datos, se desarrollan los programas de aplicación y se entrena a los usuarios. Después de la implementación se evalúa la base de datos como tal para determinar si verdaderamente cumple con las necesidades de los usuarios. Se llevan a cabo las mejoras necesarias para perfeccionar la forma de usar la base de datos y para atender las necesidades de los cambios que se produzcan en los negocios.

1. Defina con sus propias palabras cada uno de los términos siguientes:
   a. Base de datos
   b. Procesamiento electrónico de datos
   c. Sistema de información de gestión
   d. Base de datos centralizada
   e. Ciclo de vida del desarrollo de la base de datos
   f. Administración de la base de datos
   g. Diseño físico de la base de datos
   h. Restricción de valor
   i. Arquitectura tres-niveles
   j. Nivel interno
   k. Planificación preliminar
   l. Viabilidad tecnológica

m. Definición de requisitos
n. Implementación de la base de datos

2. Analice las ventajas y desventajas de compartir los datos para:
   a. Diferentes áreas funcionales
   b. Diferentes locaciones geográficas

3. ¿Cómo se usa el nivel de transacciones de datos para soportar las necesidades de información de los niveles de gestión y ejecutivos? ¿Qué otras clases de información pueden ser necesarias para la toma de decisiones estratégicas?

4. Analice los aspectos siguientes del proyecto de planificación estratégica de la base de datos:
   a. Soporte para la dirección experimentada
   b. Composición del equipo del proyecto
   c. Longitud del proyecto
   d. Alcance del proyecto
   e. Resultado del proyecto
   f. Interrelación del proyecto con el ciclo de vida del desarrollo de la base de datos

5. ¿Cuál es la responsabilidad de la administración de la base de datos con respecto a:
   a. Diseño conceptual de la base de datos
   b. El centro de Información
   c. Integridad de los datos
   d. Rendimiento del sistema de base de datos

6. Indique todos los inconvenientes potenciales que pueda imaginar asociados a la implementación y utilización de un sistema de base de datos. Clasifíquelos en función de si son arriesgados o caros. Explique por qué los clasificó así.

7. Explique las diferencias entre los niveles conceptual y externo en la arquitectura de tres-niveles ANSI/SPARC.

8. Explique el significado de la oración: "En realidad la base de datos sólo existe al nivel interno [ANSI/SPARC]".

9. Analice las diferencias entre el desarrollo de sistemas orientados a funciones y el desarrollo de sistemas orientados a datos. ¿Por qué es el sistema orientado a datos más propicio para permitir un amplio rango de funciones?

10. Indique y describa brevemente cada uno de los seis pasos en el desarrollo del ciclo de vida de la base de datos.

1. Haga corresponder los términos siguientes con sus definiciones:

—*sistema de apoyo a la toma de decisiones*	a. Donde los usuarios pueden encontrar facilidades de cómputo.
—*acceso sólo para recuperación*	b. Porción restringida de la base de datos.
—*planificación de la base de datos*	c. Múltiples sistemas de bases de datos conectados por líneas de comunicación.
—*viabilidad operacional*	d. Centrarse en el análisis de los datos usados por las funciones.
—*nivel externo*	e. Creación de un esquema de nivel conceptual.
—*diseño conceptual de base de datos*	f. Actualización no permitida.

—*integridad de los datos*	g. Esfuerzo estratégico para determinar las necesidades de información a largo plazo.
—*vista*	h. Combinar los datos para uso común.
—*integración de los datos*	i. Identifica los elementos, interrelaciones y restricciones.
—*ciclo de vida del desarrollo del sistema*	j. Nivel estructural que define el esquema lógico.
—*enfoque orientado a funciones*	k. Nivel estructural que define vistas de usuario.
—*sistema de base de datos distribuida*	l. Precisión y consistencia de los valores.
—*esquema*	m. Provee información estratégica.
—*centro de información*	n. Define la estructura lógica de la base de datos completa.
—*diseño conceptual*	o. Estudio costo-beneficio.
—*enfoque orientado a datos*	p. Un proceso para el desarrollo de sistemas.
—*nivel conceptual*	q. Determina la disponibilidad de experticidad y personal.
—*estudio de viabilidad*	r. Vistas de un sistema desde la perspectiva de las funciones que debe realizar.
—*viabilidad económica*	s. Determina la factibilidad tecnológica, operacional y económica.

2. Las áreas funcionales de IPD incluyen inventario, marketing, ventas, compra, contabilidad, procesamiento de pedidos y evaluación del producto. Identifique las áreas funcionales que (1) necesiten datos o (2) pudieran usar los datos contenidos en cada uno de los siguientes documentos. Explique en cada caso por qué se necesitan los datos o cómo podría usarlos:
   a. Un pedido a un departamento en Canadá de 100 vasos de cristal.
   b. Un informe que muestre las ventas y devoluciones de productos para cada uno de los representativos de ventas en la región de América Latina.
   c. Un informe que muestre los cambios en la cantidad en mano de cada producto en un período de tres meses.
   d. Un informe que muestre la cantidad en mano actual de cada producto, junto con el costo promedio y el precio actual por cada producto.

3. Clasifique los tipos siguientes de información según sea más significativa al nivel operacional, dirección intermedia o nivel ejecutivo:
   a. La población de Clark County es de 500.000 y se ha duplicado en cada una de las últimas dos décadas.
   b. La deuda de los Smith está pasada en dos meses.
   c. Becky Daines se casó y ahora su segundo apellido es Martínez.
   d. Galen ....
   e. ¡El cheque de Wilson fue de nuevo incobrable!
   f. Los costos de reparación para el equipamiento viejo de la fábrica de Charlesville están por los aires, es imposible encontrar nuevos empleados y los costos de transporte desde y hacia Charlesville comienzan a ser exorbitantes.

4. En una organización geográficamente distribuida, ¿qué tipo de información sería necesario centralizar y cuál necesario localizar?
   a. Las leyes sobre impuestos salariales de un país.
   b. El pago total del salario mensual de una fábrica.
   c. Las ventas mensuales de una oficina de ventas por línea de producto.
   d. Las ventas semanales de una oficina de ventas por representativo de ventas.

5. Analice las estrategias siguientes de proyectos de planificación de bases de datos:
   a. El equipo del producto gastó un año completo entrevistando al menos a un oficinista por cada unidad de procesamiento en la compañía.

b. El equipo del proyecto investigó cuidadosamente la información que se necesita de la oficina matriz, pero no gastó tiempo en las oficinas ramales porque estaban sólo interesados en la información de alto nivel.

6. Clasificar en cada uno de los puntos siguientes, dentro de las tareas de administración de bases de datos de diseño, formación, seguridad/integridad y rendimiento.
   a. Mostrarle a Helen Blomquist cómo construir una consulta para identificar el porcentaje promedio de devoluciones de cada producto.
   b. Reducir el tiempo de respuesta a la entrada de un producto en dos segundos.
   c. Rehacer las contraseñas para cada usuario del sistema.
   d. Relacionar directamente cada venta con el representante de ventas que la hizo.

7. Identificar las dificultades potenciales en las situaciones siguientes:
   a. Un área funcional está reacia a que otras áreas accedan a sus datos.
   b. Un directivo experimentado inicia un proyecto de base de datos sin estar completamente convencido de su valor.
   c. Se ha caído la computadora que controla una base de datos centralizada.
   d. Un sistema en línea necesita un tiempo de respuesta más rápido que el que puede proveer el SGBD.
   e. El único administrador de base de datos de una compañía en una pequeña ciudad ha pasado a trabajar para una compañía en otro Estado.

8. Identifique los niveles ANSI/SPARC de cada uno de los puntos siguientes:
   a. Un índice que nos da la dirección en disco de cada registro en un archivo PERSONAL.
   b. Una versión parcial del archivo PERSONAL que contiene sólo el nombre y la dirección de cada empleado.
   c. Una combinación de los datos de las transacciones de ventas y los datos de los clientes que muestra el número de producto del producto vendido y el número de cliente, su nombre y su dirección, pero no muestra el número del representante de ventas.
   d. Un archivo que da la dirección en disco del registro de un empleado junto con la direcciones en disco de los registros de todos los empleados dependientes.
   e. Datos que nos dan los nombres de todos los archivos y campos en la base de datos junto con las definiciones de las interrelaciones entre los campos en los diferentes archivos.

9. Identifique las funciones del sistema de información que para llevarse a cabo pueden usar los conjuntos siguientes de elementos de datos:
   a. Para una tienda por departamentos: número del producto, precio regular, precio de venta, costo, cantidad vendida, departamento, representativo de ventas, tarifa de comisión, supervisor. (Ejemplo: Usando el precio regular, el precio de venta y el costo, se puede calcular la ganancia y el precio de venta.)
   b. Para una firma consultante: consultante, rango del consultante, la tarifa horaria por rango, cliente, tipo de cliente, horas que un consultante trabaja para un cliente particular, fecha en las que se aplican las horas facturadas, tipo de proyecto, líder de proyecto. (Ejemplo: Usando consultante, rango, tarifa, horas trabajadas y fechas, se puede calcular el total facturado por el trabajo de un consultante para un cliente en un mes.)

10. ¿En qué paso(s) del CVDBD podrían llevarse a cabo cada uno de los puntos siguientes?
    a. Obtener un cálculo razonable del costo de operación de un sistema de base de datos.
    b. Determinar los archivos que se utilizan para el informe de ventas trimestral.
    c. Determinar si la organización tiene gente que está técnicamente calificada para diseñar y administrar una base de datos.
    d. Determinar las funciones que se llevan a cabo por el sistema de cuentas pagables.
    e. Introducir los datos en la base de datos.
    f. Identificar la información que necesita el administrador del departamento de adquisición relativa a la calidad del servicio de los vendedores.

g. Determinar los cambios necesarios para hacer más efectiva la base de datos.
h. Especificar la estructura conceptual completa de la base de datos.
i. Escribir y verificar programas de aplicación de base de datos.
j. Determinar qué funciones el sistema de base de datos realizará en cada área.
k. Determinar cuándo la tecnología de las comunicaciones es suficientemente poderosa para hacer que un sistema distribuido por todo el mundo sea práctico.

11. Explique cómo la información concerniente a las aplicaciones actuales y a los archivos que se han conformado durante la planificación preliminar pueden usarse para auxiliarnos a determinar los requisitos futuros de información.

12. Indique los formularios preimpresos que pudieran usarse en cada uno de los siguientes casos:
a. Una compañía de seguros.
b. Una compañía de electricidad.
c. Una tienda por departamentos.
d. Una universidad.

1. Entreviste a un ejecutivo de una gran organización que haya conducido recientemente un estudio de planificación estratégica de una base de datos. Determine las respuestas a cada una de las siguientes preguntas:
a. ¿Cuál fue la composición del equipo del proyecto? ¿Cuántos miembros tuvo el equipo y cuál fue su experiencia profesional? ¿Fue designado un consultante como asesor del equipo?
b. ¿Cuál fue la naturaleza del director general encargado del proyecto? ¿Creía éste en el éxito potencial del proyecto o estaba escéptico? ¿Qué influencias tuvo su cometido en la actitud de otros empleados?
c. ¿Cuál fue el resultado del proyecto? ¿Se compró un SGBD? ¿Se diseñó e instaló una nueva base de datos? ¿A cuántas áreas funcionales y niveles de dirección da servicio la base de datos y sus programas de aplicación? ¿Fueron consideradas bases de datos distribuidas? ¿Cuáles son los planes de bases de datos futuros de la organización?

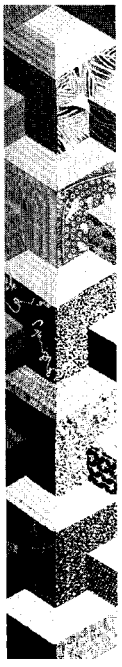
2. Escriba un artículo de investigación sobre la diferencia entre un administrador de datos y un administrador de base de datos. Determine si en la práctica la mayoría de las organizaciones distinguen entre los dos.

3. Escriba un artículo de investigación sobre una base de datos que haya fracasado. Trate de determinar por qué fracasó el proyecto. ¿Qué circunstancias existieron en este caso que pudieran no existir en otros casos? ¿Qué lecciones se pueden extraer de la experiencia de la gente en el proyecto?

4. Escriba un artículo de investigación de los aspectos que conducen al informe ANSI/SPARC sobre arquitectura de tres-niveles. ¿Por qué el comité recomendó estos niveles? ¿Cuál fue el contexto histórico de este informe? ¿Se dejaron algunos aspectos sin resolver?

5. Examine los archivos y las funciones de un sistema de aplicación instalado. Discuta el sistema con algunos de sus usuarios y determine si ellos querrían que el sistema llevase a cabo otras funciones. Identifique qué elementos de datos del sistema actual y cuáles elementos de datos nuevos se requerirían para las nuevas funciones.

6. Investigue sobre el concepto de ciclo de vida del desarrollo de la base de datos en libros y revistas actuales. Trate de identificar versiones del CVDBD que sean diferentes a las que se han dado en este capítulo. ¿Qué tienen estas versiones en común? Trate de sintetizar las diferentes versiones en un único y genérico CVDBD.



PARTE

DOS

DISEÑO DE BASE DE DATOS

En la parte II se tratan los problemas de definición de requisitos, diseño lógico y diseño de implementación en el ciclo de vida del desarrollo de la base de datos. En la medida que se vaya avanzando en el material de esta parte, el usuario irá desarrollando habilidades en el análisis de requisitos, diseño conceptual de base de datos, normalización e implementación del modelo relacional de datos.

En el Capítulo 4 se introducirá el diseño conceptual y se verá cómo se pueden usar los conceptos de interrelaciones entre objetos, atributos y generalización y especialización para crear modelos de información. También se aplican estos principios para diseñar soluciones de problemas más complejos. Se discuten y aplican técnicas sofisticadas de análisis y diseño a una variedad de ejemplos prácticos en una amplia gama de situaciones de negocios.

El Capítulo 5 introduce el modelo relacional y explica su significación en la historia del procesamiento de base de datos. Este es el modelo predominante en los entornos de negocios de hoy día para la implementación de nuevos sistemas. También se define e ilustra el proceso de normalización de la base de datos y la conversión del modelo conceptual.

CAPÍTULO

4

PRINCIPIOS DEL DISEÑO CONCEPTUAL DE BASE DE DATOS

Realidad, definición de requisitos y modelado conceptual de datos
- Realidad y modelos
- Modelos conceptuales de datos

Fundamentos
- Objetos
- Especialización y generalización
- Relaciones
- Cardinalidad
- Atributos
  - Claves
  - Especialización/Generalización y Atributos

Ejemplos
- Ejemplo 1: El modelo de datos de un banco
  - Cardinalidades
  - Especialización de los clientes del banco
- Ejemplo 2: Huerto frutícola de Stratton
- Ejemplo 3: Un problema de lógica

Construir modelos conceptuales de datos a partir de los informes existentes
- Caso: Servicios de Consultoría Manwaring
  - Estimados
  - Recepción de efectivo
  - Desembolso de efectivo
- Un modelo de datos para las compras
- Un modelo de datos para facturación de proyectos

Agregación
- Ejemplo 4: Compañía constructora Premier
- Ejemplo 5: Huerto frutícola de Stratton (continuación)
- Ejemplo 6: Servicios de Consultoría Manwaring (continuación)

Modelado conceptual de objetos contra objetos físicos
- El problema de una biblioteca
- Crear el modelo de datos de una Biblioteca
- Rastrear las piezas fabricadas
- Objetos conceptuales para los servicios de consulta

Integración de vistas: Un ejemplo

Resumen

Preguntas de comprobación

Problemas y ejercicios

Proyectos y cuestiones profesionales

¿Cuántas cuentas corrientes tenemos? ¿Cuántas cuentas de ahorro? ¿Cuántos clientes? ¿Cuántos clientes tienen ambos tipos de cuenta? ¿Qué porcentaje de nuestras cuentas de ahorro tienen un saldo superior a $1.000? ¿Qué tipo de clientes tiende a tener el mayor saldo promedio en sus cuentas? ¿Cuántos clientes tienen préstamos por encima de sus cuentas corrientes o de ahorro?"

Robert Goldthumb, presidente del Alchemical Bank and Trust (ABT), está frustrado porque no puede tener diariamente las respuestas a este tipo de preguntas. Mientras tanto, Anita Short, jefa de contaduría, necesita estar segura de que cada cliente recibe mensualmente el estado de cuenta, y Elliot Tigh, jefe de Préstamos a Clientes, necesita un informe semanal de los pagos vencidos de los préstamos y necesita también una aplicación que le genere automáticamente las cartas de recordatorio.

Claramente cada una de estas necesidades de los usuarios puede satisfacerse con un sistema de base de datos. También está claro que estos tres usuarios tienen necesidades diferentes. Es fácil ver que hay bastante solapamiento en los tipos de datos que los tres usuarios requieren. La tarea durante la definición de los requisitos y el diseño conceptual es identificar las necesidades básicas de datos y crear los modelos conceptuales de los datos que nos aseguren registrar los datos necesarios y las relaciones entre éstos. Después de leer este capítulo, el usuario debe ser capaz de:

- Usar las técnicas básicas de modelado conceptual de datos para la captura de los datos y las relaciones entre ellos inherentes a consultas simples de los usuarios y a los informes y formularios existentes.
- Mostrar cómo se pueden crear conjuntos de objetos agregados a partir de las relaciones existentes y cómo éstas funcionan como conjuntos de objetos que tienen atributos y que participan en las relaciones.
- Demostrar cómo las necesidades de información que ocurren con frecuencia en los negocios se pueden satisfacer usando estos conceptos de modelado de datos.

▼ Realidad, definición de requisitos y modelado conceptual de datos

Los procesos de definición de los requisitos y del diseño conceptual exigen identificar las exigencias de la información de los usuarios y representar éstos en un modelo bien definido. Para llevar a cabo esto necesitamos observar cuidadosamente la naturaleza de las condiciones de los usuarios y el significado preciso de la representación lógica de los mismos.

Realidad y modelos

modelo. Una representación de la realidad que retiene sólo detalles seleccionados.

¿Qué es un modelo? Un **modelo** es una representación de la realidad que conserva sólo los detalles relevantes. Por ejemplo, consideremos una transacción bancaria tal como un depósito en una cuenta corriente. Contaduría desea conservar ciertos detalles (número de la cuenta, monto del depósito, tiempo, fecha, número del cajero) e ignorar otros (las palabras que se han intercambiado durante la transacción, el número de gente en el banco, el número de personas que estaban esperando en la cola, la música que se estaba tocando en el audio local, las condiciones del clima fuera del banco, etc.). La realidad involucra un sinnúmero de detalles, pero contaduría considerará a la mayoría de ellos irrelevantes para la transacción. De modo que un modelo, desde el punto de vista de Contaduría, conservará sólo aquellos detalles que ésta considere relevante.

Por supuesto, algunos detalles considerados irrelevantes por un usuario pueden ser muy importantes para otros usuarios. Imaginemos, por ejemplo, que se está desarrollando un sistema de base de datos para un restaurante de comida rápida. Las condiciones del clima pueden ser aspectos significativos de la realidad del administrador del restaurante, puesto que un día frío debe ofertar un conjunto de ensaladas bastante diferente del de un día cálido. Como resultado, el administrador debe desear poder tener en cuenta estos cambios de clima y las órdenes que han sido suministradas de acuerdo con esto. El número de personas esperando en la cola puede ser otro aspecto importante para la realidad del administrador, puesto que éste necesita esta información para poder planificar la cantidad de cajeros y minimizar el tiempo de espera de un cliente. Por lo tanto, diferentes usuarios pueden tener diferentes modelos de la realidad.

Una base de datos incorpora un modelo de la realidad. El SGBD administra la base de datos de modo que cada usuario pueda registrar, acceder y manipular los datos que constituyen su modelo de la realidad. Manipulando los datos en una amplia variedad de formas, los usuarios pueden obtener la información necesaria para conducir una empresa con éxito. Por lo tanto, los modelos son herramientas poderosas para eliminar los detalles irrelevantes y comprender la realidad de los usuarios individuales.

Modelar la realidad es en muchas maneras similar a resolver un problema de una historieta. Ambos requieren que nos desprendamos de los detalles para crear un modelo "correcto" de una porción de la realidad. Esto significa que se debe asociar, o **identificar**, elementos de la realidad con elementos en el modelo. Si esta asociación se hace correctamente, entonces el modelo se puede usar para resolver el problema. De lo contrario, el modelo no puede producir una solución correcta. Mucha gente encuentra que el problema de las historietas es difícil porque no se sienten cómodos con el proceso de asociación en sí mismo. De hecho, la Figura 4.1 puede representar su propio enfoque de los problemas de historietas. Si así fuese, esperamos ayudarle a sentirse más cómodo con el modelado e identificación de la realidad. Se comenzará con conceptos de modelado simples y básicos y se mostrará cómo éstos pueden usarse para construir, paso a paso, una solución poderosa a lo que puede parecer un problema complejo. En la medida que se estudien los ejemplos, y se trabaje a través de los ejercicios y los casos, se podrán desarrollar habilidades significativas para el modelado de datos.

asociar (map). Asociar elementos de una esfera con elementos en otra esfera.

EL LADO LEJANO, por GARY LARSON

La Biblioteca del Diablo

Fuente: Copyright 1987 Universal Press Syndicate.
Reimpreso con permiso.
Todos los derechos reservados.

MODELOS EN LOS DIFERENTES NIVELES

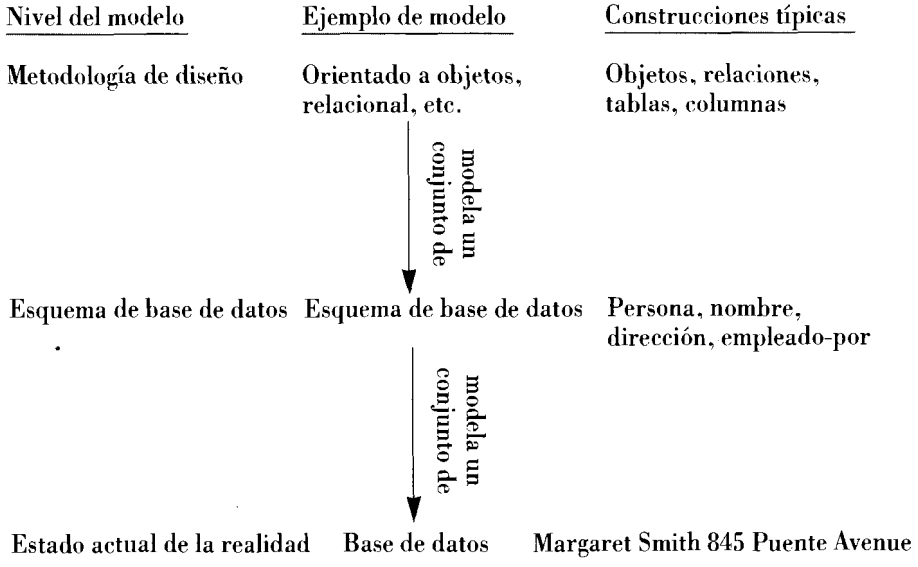
Aunque puede no ser obvio, se ha estado usando el término *modelo* a tres niveles diferentes en nuestro examen. Estos niveles (no relativos a los tres niveles de la arquitectura ANSI/SPARC) se ilustran en la Figura 4.1S.

En el nivel más bajo se dice que el estado en curso de una base de datos en particular es un modelo de la realidad porque es un registro de hechos seleccionados sobre la realidad que son verdaderos. Por ejemplo, la base de datos puede registrar el hecho: "Margaret Smith vive en el 845 Puente Avenue." Si Margaret cambia de dirección, entonces el estado de la base de datos debe cambiar si quiere continuar como un modelo preciso de la realidad.

En el próximo nivel, el esquema se describe en la estructura de la base de datos como un modelo de un conjunto de modelos (esto es, es un modelo de un conjunto de estados de la base de datos). El esquema modela un enorme rango de estados de la base de datos, definiendo aquellas características que todos estos estados tienen en común. Así, "Nombre" y "Dirección" se registran en el modelo como características que se aplican a diferentes personas y que cambian de tiempo en tiempo.

Al nivel más alto, la metodología de diseño describe las construcciones y reglas que pueden ser utilizadas en la formulación de un esquema . Por lo tanto, este nivel es también un modelo de un conjunto de modelos (posibles esquemas de bases de datos). Una metodología dada de diseño, tal como el modelo conceptual de datos o el modelo relacional, es un modelo al nivel más alto y describe en términos generales un conjunto potencialmente enorme de esquemas.

En resumen, se habla del modelo conceptual de datos como de una metodología para la creación de esquemas de bases de datos para situaciones particulares de aplicación. Estos esquemas de bases de datos son en sí mismos modelos que proveen una estructura lógica para capturar hechos sobre una porción particular de la realidad. Cuando estos hechos son capturados y registrados en un sistema de base de datos, entonces la base de datos en sí misma es un modelo del estado actual de la realidad. Cada uno de los dos niveles superiores de la Figura 4.1S es un modelo del nivel que tiene debajo.



Modelos conceptuales de datos

modelo orientado a objetos. Un modelo que representa las entidades del mundo real como objetos en lugar de como registros.

modelo semántico. Un modelo que captura los significados de las entidades del mundo real y sus interrelaciones.

Una metodología de modelado de datos que se estudiará y utilizará en este capítulo podría llamarse **modelo orientado a objetos** porque considera la representación en la computadora de las entidades del mundo real como "objetos" que tienen su propia identidad y atributos y que participan en las relaciones, en lugar de la forma tradicional de considerarlas como registros en un sistema orientado a archivos. Es generalmente reconocido que las representaciones orientadas a objeto son más precisas para expresar la esencia lógica de las aplicaciones del mundo real que las representaciones basadas en registros. Por esta razón, esta metodología también puede llamarse **semántica** porque proporciona un medio poderoso para asociar los *significados* de las cosas en la realidad con las construcciones en el modelo. Desde principios de los setenta han sido propuestos varios modelos conceptuales y semánticos de datos. Se usará una metodología genérica de modelado, la cual tiene aspectos en común con las metodologías propuestas. Por simplicidad, a lo largo del libro se le llamará a este modelo *modelo conceptual de datos*.

En el Capítulo 2 se introdujo un ejemplo que mostraba un modelo de datos conceptual sencillo. Ahora estamos listos para definir estas ideas de modo más preciso y adicionalmente introducir conceptos más poderosos, los cuales nos permitirán definir modelos aún más sofisticados.

¿MODELADO SEMÁNTICO U ORIENTADO A OBJETOS?

Las bases de datos orientadas a objetos son el resultado de la convergencia de dos disciplinas de investigación: el modelado semántico de datos y los lenguajes orientados a objeto. Estas disciplinas se desarrollaron de manera independiente, pero en los ochenta comenzaron a mezclarse con importantes implicaciones en el procesamiento de bases de datos.

El *modelado semántico de datos* fue originalmente desarrollado con el propósito de incrementar la efectividad y la precisión del diseño de bases de datos (Hull and King, 1987). Los métodos de modelado semántico fueron considerados apropiados para muchos problemas de usuario y podrían ser convertidos con facilidad a modelos con realizaciones basadas en registros tales como los modelos jerárquico, de redes y relacional. Abrial introdujo el modelo binario semántico de datos en 1974, y éste fue continuado durante los años siguientes por el modelo de entidad-relación de Chen (Chen, 1976), el modelo semántico de datos (SMD [1]) de Hammer y McLeod (1981) y el modelo de datos funcional (Shipman, 1981). Estos y otros modelos de datos, así como extensiones de estos modelos, enfocan el problema del modelado de datos, con el propósito de diseño de base de datos, desde varias perspectivas. Éstas tienen en común el objetivo de proporcionar un medio de capturar la *semántica* o significado del área de aplicación que se está modelando. El modelo de entidad-relación de Chen (E-R) ha sido el modelo semántico más popular y comúnmente se puede encontrar en libros sobre modelado conceptual de datos y diseño de bases de datos.

Mientras que aquellos dedicados al modelado semántico de datos tienen que ver primariamente con las estructuras de los datos, los que desarrollan los *lenguajes de programación orientados a objetos* están más interesados en el comportamiento de los datos objetos. Esto es, están buscando formas de manipular los datos que se centran en los datos y en las capacidades de manipulación del lenguaje (consulta, cómputo, actualización). La estructura de los datos pasa a un plano secundario.

[1] Siglas del inglés *Semantic Data Model* (N. del T.).

La convergencia de estas dos áreas surge cuando los investigadores comienzan a aplicar conceptos de los lenguajes orientados a objetos a las estructuras de datos semánticas. El resultado es la noción de una *base de datos orientada a objetos*. En esta mezcla de disciplinas, la terminología orientada a objetos tiende a predominar y por ello se habla más de objetos que de entidades, que es lo que usaríamos en la terminología semántica. Adicionalmente, los lenguajes orientados a objeto enfatizan varios conceptos que no aparecen en el modelo original E-R de Chen: identidad de un objeto, jerarquía de supraconjunto, subconjunto de objetos y herencia. Éstos se estudian más adelante. Así pues, la metodología que se utiliza combina el modelo E-R de Chen con conceptos del modelado orientado a objetos. El modelo E-R constituye la base de nuestro modelo conceptual de datos y el modelado orientado a objetos contribuye con varias mejoras significativas.

▼ Fundamentos

Los elementos principales de un modelo conceptual de datos son los *objetos* y las *relaciones*. Los objetos se conciben a menudo como *nombres*, y las relaciones se ven como *verbos*. Aunque algunos modelos conceptuales de datos proporcionan algunas construcciones adicionales, los objetos y las relaciones son suficientemente poderosos para los problemas que se considerarán.

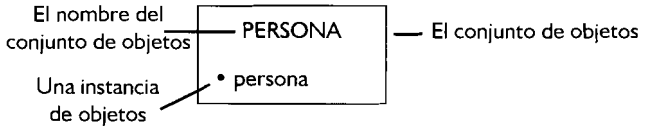
Objetos

Los objetos representan cosas que son importantes para los usuarios en el segmento de la realidad que queremos modelar. Ejemplos de objetos son las personas, los automóviles, los árboles, las máquinas lavaplatos, las casas, los martillos, los libros. Estos son objetos concretos. Objetos conceptuales son compañías, oficios, organizaciones, diseños de productos, transacciones de negocios y clasificaciones de los trabajos.

De lo anterior puede no quedar claro cuándo un objeto es una cosa en particular (una persona individual, un automóvil en particular, un banco en específico) o un *conjunto* de cosas (todas las personas, todos los automóviles, todos los bancos). Para evitar ambigüedad se usará el término **conjunto de objetos** para referirnos a un conjunto de cosas de la misma clase e **instancia (ejemplar) de objeto** para referirnos a un simple elemento (o elemento) de un conjunto de objetos. Como muestra la Figura 4.2, se usarán rectángulos para representar conjuntos de objetos y puntos para representar las instancias. El nombre de un conjunto de objetos todo en mayúsculas es la versión singular del objeto. De esta manera, "PERSONA" es el nombre del conjunto de objetos representando a la gente. Una "persona" (en minúscula) es una instancia del conjunto de objetos PERSONA. Se escribe "persona IN PERSONA" para indicar que persona es una instancia de PERSONA o que la instancia "persona" está en el conjunto de objetos "PERSONA".

conjunto de objetos. Un conjunto de cosas de la misma clase.

instancia de un objeto. Un miembro particular de un conjunto de objetos.



conjunto de objetos léxico. Un conjunto de objetos que consta de las instancias que se pueden imprimir.

conjunto de objetos abstractos. Un conjunto de objetos que consiste de las instancias que no se pueden imprimir.

Los conjuntos de objetos son **léxicos** o **abstractos**. Las instancias en conjuntos de objetos léxicos se pueden imprimir, mientras que las instancias de un objeto abstracto no. Así, por ejemplo, NOMBRE sería un conjunto de objetos léxicos, puesto que las instancias de NOMBRE son nombres formados por cadenas de caracteres que pueden imprimirse. FECHA, CANTIDAD y NÚMERO DE SEGURIDAD SOCIAL son otros ejemplos de conjuntos de objetos léxicos, puesto que las fechas, cantidades y números de seguridad social también se pueden imprimir.

Por otra parte, PERSONA es abstracto porque una persona no se puede imprimir. Aunque es cierto que una persona puede *representarse* por un objeto léxico, tal como un nombre o número de seguridad social, se insiste, sin embargo, en que una persona *no* es un nombre ni un número de seguridad social. Por ejemplo, el nombre de una persona o su número de seguridad social pueden cambiar, pero la persona continúa siendo la misma. Por lo tanto, para lograr un modelo más preciso de la realidad, nosotros distinguimos entre conjuntos de datos abstractos y conjuntos de datos léxicos.

En una realización computacional de un modelo conceptual, una instancia de un objeto léxico estaría representada por una cadena de caracteres imprimibles. Una instancia de un objeto abstracto estaría representada por un número interno que no tendría significado fuera del sistema. Este número interno en ocasiones es denominado *identidad del objeto*, o una **clave subrogada**, que representa e identifica unívocamente el objeto instancia abstracta del mundo real.

clave subrogada. Un identificador único para una instancia de un objeto abstracto, no tiene significado fuera del sistema de computador.

Suponga que una persona llamada Juanita Pérez está en el conjunto de objetos PERSONA. En una representación concreta de este conjunto de objetos, Juanita estaría representada por alguna clave subrogada, digamos "13948226". Su nombre (Juanita Pérez), el número de seguridad social, la fecha de nacimiento, la altura, el peso y otras informaciones de ese tipo podrían registrarse como datos léxicos y estarían asociadas en la base de datos con la clave subrogada que la representa. Los usuarios sólo verían estos datos léxicos y nunca verían el 13948226 en asociación con Juanita. Pero el sistema utilizaría los subrogados en todas las posibles numerosas relaciones que son parte de la base de datos.

Las claves subrogadas resuelven problemas que surgen de los tipos tradicionales de las claves. Por ejemplo, en muchos sistemas es muy difícil cambiar un valor clave. El número de seguridad social a menudo se utiliza como una clave para acceder unívocamente a la información sobre una persona. ¿Qué pasa si el número de seguridad social es incorrecto? Puesto que este número tiene importancia legal fuera del sistema de base de datos *debe* ser corregido. Pero esto podría llevar a muchas dificultades *dentro* de la base de datos, puesto que podría haber muchas referencias a dicho número de seguridad social. Este problema se elimina utilizando claves subrogadas, ya que éstas son definidas por el sistema y no tienen significado fuera del sistema. Si el número de seguridad social de Juanita Pérez está incorrecto, simplemente lo cambiamos y nada en la base de datos se afectará, puesto que nadie se refiere al número de seguridad social. En cambio, todas las referencias a Juanita Pérez usan la clave subrogada.

Especialización y generalización

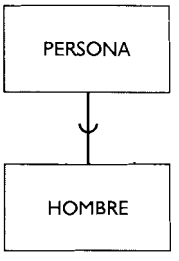
especialización. Un conjunto de objetos que es un subconjunto de otro conjunto de objetos.

generalización. Un conjunto de objetos que es un superconjunto de (o que contiene a) otro conjunto de objetos.

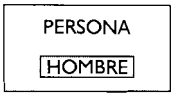
Algunos objetos están contenidos dentro de otros objetos. Por ejemplo, HOMBRE (conjunto de hombres) está contenido dentro de PERSONA. Esto significa que todo hombre (toda instancia del conjunto HOMBRE) es también una persona (una instancia del conjunto PERSONA). De modo similar, MUJER está contenida dentro de PERSONA. Se dice que HOMBRE es una **especialización** de (o un subconjunto de) PERSONA. Esto se puede representar escribiendo HOMBRE PERSONA. Por otro lado, PERSONA es una **generalización** o superconjunto de HOMBRE (y de MUJER). Especialización/generalización se representa gráficamente como se muestra en la Figura 4.3(a). La figura en forma de U indica la dirección de contenido de conjuntos. Si ponemos los conjuntos uno al lado del otro, la U debe estar de lado. Si PERSONA estuviera en el cuadrado del fondo y HOMBRE en el del tope, la U debería estar hacia arriba. También se puede mostrar a HOMBRE como si estuviera

dentro de PERSONA [Figura 4.3(b)]. Puesto que muchos conjuntos de objetos pueden estar contenidos dentro de un solo objeto, esta técnica tiende a causar múltiples diagramas.

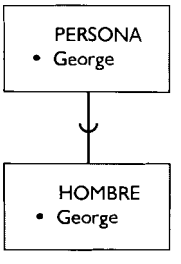
Suponga que George es un hombre, entonces George es también una persona. Esto se muestra gráficamente en la Figura 4.4. Obsérvese que dos puntos representan a la misma persona. Un punto representa a ésta como ejemplo en PERSONA y el otro punto la representa como un ejemplo en HOMBRE. Hay realmente un solo ejemplo, lo único que ésta se está mostrando como residiendo en dos objetos diferentes. La importancia de esto será ilustrada posteriormente.



(a) La interrelación de especialización-generalización



(b) Una representación alternativa de especialización



Interrelaciones

interrelación. Un enlace entre instancias de dos conjuntos de objetos.

Una **interrelación** enlaza a dos conjuntos de objetos. Considere el conjunto de objetos HOMBRE CASADO y MUJER CASADA. Se puede definir la interrelación ESTÁ-CASADO-CON entre estos dos conjuntos de objetos asociando cada hombre casado con su esposa (o inversamente cada mujer casada con su esposo). La interrelación ESTÁ-CASADO-CON consiste de un conjunto de parejas de casados, el marido que proviene del conjunto de objetos HOMBRE CASADO y la mujer que proviene del conjunto de objetos MUJER CASADA. Gráficamente se representa una interrelación entre dos conjuntos de objeto mostrando una línea (con opcionalmente un diamante incrustado) conectando los dos conjuntos [Figura 4.5(a)].

Una interrelación es en sí misma un conjunto de objetos consistente de pares de instancias tomadas de los dos conjuntos de objetos que relaciona. Esto es, cada instancia de la interrelación es un par de instancias tomadas de los dos conjuntos de objetos. Si

```
HOMBRE CASADO = { Adam, David, John}      y
MUJER CASADA = {Joan, Linda, Michelle}    y
Adam          está-casado-con      Joan
David         está-casado-con      Linda
John          está-casado-con      Michelle
```

entonces

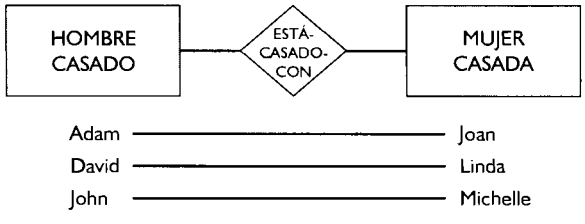
```
ESTÁ-CASADO-CON =
     { (Adam, Joan), (David, Linda), (John, Michelle) }
```

Las claves ({ }) se usan para indicar un conjunto. La Figura 4.5 (b) muestra esta información gráficamente. Se ve que la interrelación es en sí misma un conjunto de objetos cuyas instancias son parejas de casados. Un conjunto de objetos como ESTÁ-CASADO-CON, el cual se deriva de un interrelación entre otros dos conjuntos de objetos, se llama un **conjunto de objetos agregado.**

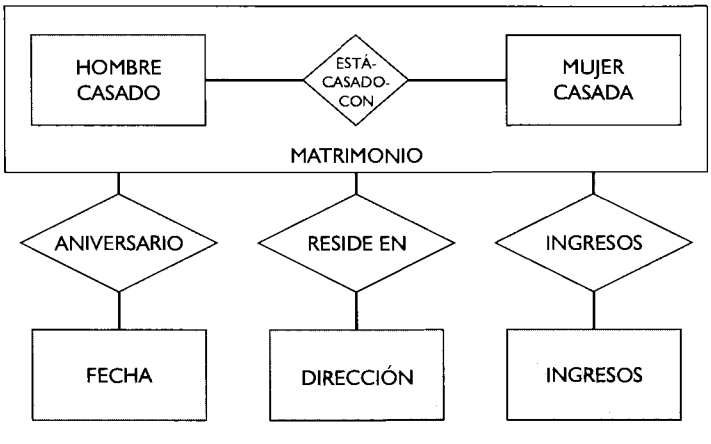
conjunto de objetos agregado. Una interrelación vista como un conjunto de objetos.

A los conjuntos de objetos agregados se les puede dar nombre y pueden participar en interrelaciones, al igual que conjuntos de objetos normales. En la Figura 4.5 (c) el agregado de ESTÁ-CASADO-CON se denomina MATRIMONIO y participa a su vez en varias interrelaciones. La interrelación ANIVERSARIO conecta a las parejas de casados con su aniversario de bodas; la interrelación RESIDE-EN conecta las parejas con sus direccio-

(a) La interrelación ESTÁ-CASADO-CON

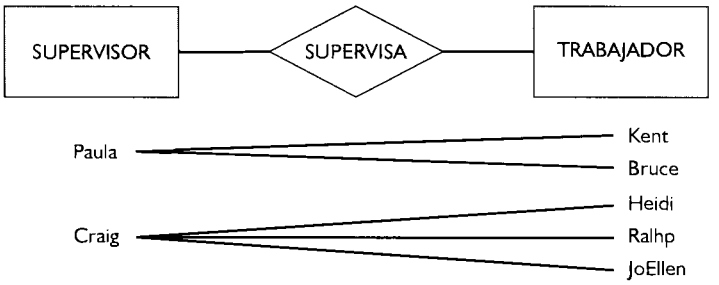


(b) Algunas instancias de ESTÁ-CASADO-CON



(c) El conjunto de objetos agregado MATRIMONIO participando en interrelaciones

(a) La interrelación SUPERVISA



(b) Instancias de la interrelación SUPERVISA

nes y la interrelación INGRESOS las conecta con el total de ingresos combinado de ambos.

Como otro ejemplo de una interrelación consideremos los dos subconjuntos SUPERVISOR y TRABAJADOR de una compañía de empleados. Se definen las instancias de TRABAJADOR como empleados que no supervisan a otros empleados. El conjunto SUPERVISOR consiste de los empleados que supervisan a los trabajadores. La interrelación SUPERVISA (obsérvese el uso del verbo) asociada a cada supervisor con los trabajadores que éste supervisa (Figura 4.6(a)). La Figura 4.6(b) ejemplifica instancias que pueden encontrarse en la interrelación SUPERVISA.

La generalización/especialización representa un tipo especial de interrelación. Recuérdese que en la Figura 4.4 dos puntos diferentes representaban a la misma persona George. El punto que en HOMBRE representa a George está relacionado por medio de esta interrelación de subconjunto con el punto que en PERSONA representa a George. En efecto, todo punto en HOMBRE está relacionado con exactamente un punto en PERSONA. Sin embargo, algunos puntos en PERSONA están relacionados con puntos en MUJER. Por tanto, todo punto en PERSONA está relacionado con cero o con un punto en HOMBRE. Este tipo de información sobre las interrelaciones se puede expresar añadiéndole cardinalidad a los diagramas.

Cardinalidad

cardinalidad. El número máximo de instancias de un conjunto de objetos que puede estar relacionado con una sola instancia del otro conjunto de objetos.

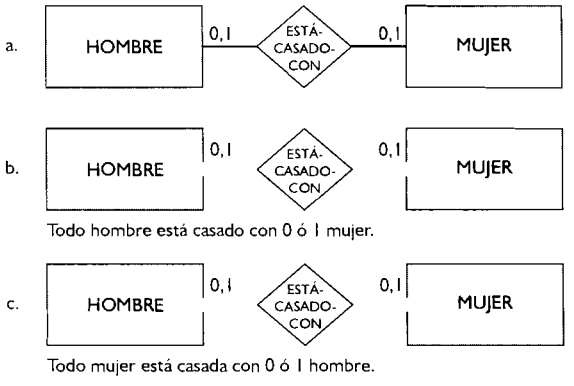
La **cardinalidad** de una relación se refiere al número máximo de instancias en un conjunto de objetos que está relacionado con una única instancia en el otro conjunto de objetos. Por ejemplo, si se asume que cada persona casada tiene sólo una esposa, la cardinalidad de la interrelación ESTÁ-CASADO-CON es 1 en cada dirección (Figura 4.7).

Aunque normalmente estamos interesados sólo en la máxima cardinalidad, a veces es útil especificar la cardinalidad mínima. Supongamos, por ejemplo, que restamos la interrelación ESTÁ-CASADO-CON a la interrelación que existe entre los conjuntos HOMBRE y MUJER [Figura 4.8(a)]. Puesto que muchos hombres y mujeres no están casados, la cardinalidad mínima es cero en ambas direcciones. Se escribirá "0,1" más cerca del conjunto de objetos MUJER para indicar que un hombre dado está casado con cero o con una mujer. Inversamente el 0,1 cercano al conjunto de objetos HOMBRE indica que cada mujer está casada con cero o con un hombre [Figuras 4.8(b,c)].

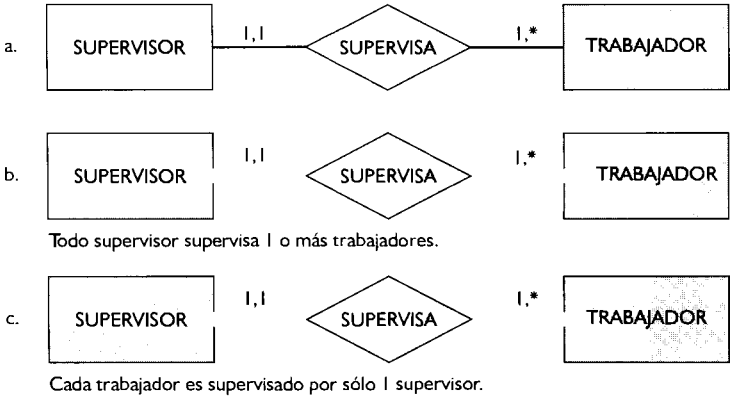
Algunas interrelaciones no tienen un valor específico para la cardinalidad máxima. Por ejemplo, un supervisor supervisa al menos a un trabajador, pero puede supervisar a muchos. Esta cardinalidad es 1,* donde el "1" indica la cardinalidad mínima y el "*" significa "muchos". Inversamente se asume que un trabajador tiene un y sólo un supervisor, por lo que la cardinalidad en la otra dirección es 1,1 (Figura 4.9).

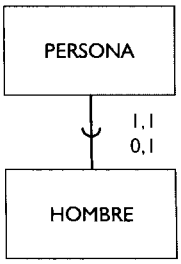
Las cardinalidades de la interrelación de especialización o subconjunto son siempre las mismas. Cada instancia en el conjunto de generalización está relacionada con cero o una instancia del conjunto de especialización, y cada instancia del conjunto de especialización está relacionada con exactamente una instancia en el conjunto de generalización (Figura 4.10).

La cardinalidad máxima es un concepto mucho más importante que el de la cardinalidad mínima. Por lo tanto, para simplificar los diagramas, la cardinalidad mínima sólo se



(Los diagramas de interrelación se pueden leer de izquierda a derecha y de derecha a izquierda)





indicará cuando sea necesario. Excepto en la interrelación de subconjunto (cuya mínima cardinalidad fue discutida anteriormente), las cardinalidades mínimas que se omiten serán asumidas como cero.

Una cardinalidad máxima de 1 en una dirección de una interrelación corresponde con el concepto matemático de función que determina una correspondencia uno-uno o muchos-a-uno entre dos conjuntos. Por lo tanto, una interrelación con una cardinalidad máxima de 1 en una dirección es llamada **funcional** en esa dirección. La interrelación supervisor/trabajador de la Figura 4.9 es funcional de trabajador a supervisor. Es decir, si se sabe que el trabajador es quien únicamente determina su supervisor. Esta interrelación *no* es funcional en la otra dirección, puesto que un supervisor puede tener muchos trabajadores.

interrelación funcional. Una interrelación que tiene una cardinalidad máxima de valor 1 en al menos una dirección.

Si la cardinalidad máxima en ambas direcciones de una interrelación es 1, se dice que la interrelación es **uno-uno**. Si el máximo es 1 en una dirección y muchos en la otra dirección, se dice que la relación es **uno-muchos**. Finalmente, si las cardinalidades máximas son muchos en ambas direcciones, se dice que la interrelación es **muchos-muchos**. La Tabla 4.1 resume las tres cardinalidades básicas de las interrelaciones.

uno-uno. Una cardinalidad de la interrelación que es 1 en ambas direcciones.

uno-muchos. Una cardinalidad de la interrelación que es 1 en una dirección y muchos en la otra.

muchos-muchos. Una cardinalidad de la interrelación que es mucho en ambas direcciones.

Atributos

Hemos representado los conjuntos de objeto como rectángulos y las instancias como puntos. Esto es muy abstracto. (¿Qué podría tener menos cosas que un punto?). Normalmente se piensa en las instancias de objetos como que tienen un cierto número de atributos que sirven para distinguirlos. Por ejemplo, una persona tiene nombre, fecha de nacimiento, número de seguridad social, estatura, peso, género, color de los ojos, color del pelo, padre, madre y, posiblemente, cónyuge. ¿Cómo representar estos atributos?

Cardinalidad	*Notación*	*Ejemplos*
Cardinalidad	Notación	Un marido tiene *una* esposa.
Uno-uno	1:1 ó 1-1	Una esposa tiene *un* marido.
Uno-muchos	1:* ó 1-*	(la interrelación de matrimonio es uno-uno.)
Muchos-muchos	*:* ó *-*	Un empleado está en *un* departamento. Un departamento tiene *muchos* empleados. (La interrelación de empleo es uno-muchos.)
		Un estudiante sigue *muchos* cursos. Un curso tiene *muchos* estudiantes. (La interrelación de matrícula es muchos-muchos.)

atributo. Una interrelación funcional de un conjunto de objetos en otro conjunto de objetos.

Un **atributo** de un objeto es realmente una interrelación funcional de ese conjunto de objetos con otro conjunto de objetos. Dos de los atributos enumerados anteriormente se muestran como interrelaciones en la Figura 4.11. Sin embargo, será conveniente representar algunos atributos de manera más simple, tal como se muestra en la Figura 4.12. Nótese que el nombre de la interrelación y que el nombre del objeto se combinan en cierto sentido para formar el nombre del atributo (especialmente FECHA DE NACIMIENTO). Escribir atributos de este modo no es más que una notación abreviada de escribir las interrelaciones. Generalmente, esta notación abreviada puede utilizarse siempre que no se intente usar el atributo como un objeto en otra interrelación.

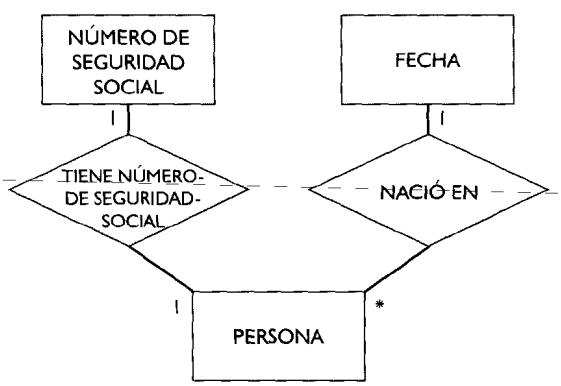
En el uso normal (que es el que se seguirá a continuación) los atributos son interrelaciones *funcionales* del conjunto de objetos al atributo, es decir, el valor del atributo está determinado unívocamente para cada instancia. Por ejemplo, cada persona tiene exactamente una fecha de nacimiento y, en esta base de datos, también un solo número de seguridad social. La cardinalidad máxima por parte del atributo en estas interrelaciones es siempre 1. Por esta razón siempre se omitirá la cardinalidad de los atributos en los diagramas. Si un objeto instancia en particular no tiene valor en alguno de sus atributos, se dice que el atributo tiene un valor **nulo** para esa instancia.

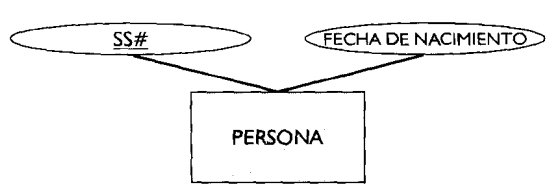
valor nulo de un atributo. Un valor de atributo que no existe en una instancia específica.

Es importante darse cuenta que los atributos pueden estar separados conceptualmente de los objetos que describen. Recuérdese la discusión inicial de que los valores de los atributos cambian frecuentemente, mientras que los objetos asociados a los atributos son los mismos. De este modo, una persona puede cambiar su estatura, su peso, su nombre y el color de su pelo, pero sigue siendo la misma persona. Esto no significa que todos los atributos deben cambiar los valores. De hecho siempre se trata de identificar cuáles atributos no cambian de valor, ya que éstos pueden utilizarse como claves externas.

clave. Un valor que siempre puede utilizarse para identificar unívocamente una instancia.

Claves. Una **clave** es un valor que siempre puede utilizarse de forma unívoca para identificar a un objeto instancia. Ya fueron mencionadas las claves subrogadas, las cuales se utilizan internamente en los sistemas para identificar conjuntos de objetos abstractos





(no léxicos). PERSONA, por ejemplo, es un conjunto de objetos abstractos. En una implementación de una base de datos conceptual, cada persona en el conjunto de objetos PERSONA tendría una clave subrogada para identificar a dicha persona dentro de la base de datos. Pero puesto que la clave subrogada no puede ser utilizada fuera del sistema de base de datos, los usuarios necesitan otra forma de identificar a las instancias de PERSONA. Esto se logra con las claves externas.

clave externa. Un conjunto de atributos lexicográficos cuyos valores siempre identifican una única instancia de objeto.

identificador. Una clave externa.

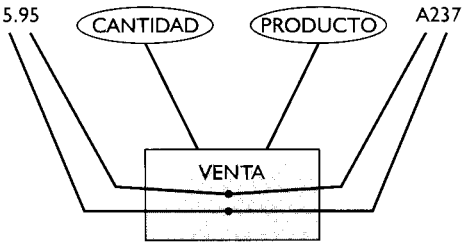
Una **clave externa** (también llamada un **identificador**) es un atributo léxico o un conjunto de atributos léxicos cuyos valores siempre identifican un único objeto. Un atributo léxico es un atributo formado ntilizando un conjunto de objetos léxico. De este modo, las claves externas se pueden imprimir y se pueden leer por los usuarios y sirven de medio por el cual instancias específicas pueden identificarse externamente a la base de datos. Usualmente nos referimos a las claves externas simplemente como claves. Por ejemplo, en la Figura 4.12, #SS podría ser una clave para PERSONA si se asume que cada número de seguridad social corresponde exactamente a nna única persona. Esto es, las cardinalidades mínima y máxima de #SS a PERSONA son 1,1. Por otra parte, la fecha de nacimiento no podría ser una clave, puesto que una determinada fecha puede ser la fecha de nacimiento de mnchas personas diferentes.

En ocasiones se necesita más de un atributo para formar una clave. Supongamos que PERSONA de la Figura 4.12 está siendo utilizada en una base de datos de genealogía, la cual expresa los árboles de las jerarquías familiares. Dado que muchas de las personas en PERSONA fallecieron antes de que se introdnjera el concepto de número de seguridad social, necesitamos algo diferente de #SS para la clave. Por supuesto que el nombre, la fecha de nacimiento y el lugar de nacimiento podrían ser suficiente. En este caso, la combinación de estos tres atributos formarían la clave para PERSONA. Si no fuese así entonces se necesitaría algo adicional. Si es necesario siempre se puede poner un número de identificación cuya unicidad pueda forzarse dentro del sistema. Se denotará que un atributo es nna clave externa, subrayando el nombre del atributo (Figura 4.12).

No todo conjunto de objetos necesita tener una clave. Por ejemplo, en una base de datos que registre transacciones de ventas, el usuario puede estar interesado solamente en registrar el total de las ventas y los productos vendidos. Obviamente, muchas ventas pueden tener los mismos valores en cantidad y en producto vendido. Sería poco razonable exigir que el usuario provea una clave única para cada venta. En este caso, la base de datos sólo registrará la información que el usuario desea de cada transacción, pero ésta quedará registrada como instancias separadas para cada transacción. La Figura 4.13 muestra dos ventas diferentes que tienen la misma cantidad de 5,95 y el mismo producto A237. El usuario verá dos transacciones, pero no tiene medios para distinguir éstas entre sí y además no está interesado en hacerlo. Por lo tanto, la ausencia de una clave no tiene consecuencia en este caso.

herencia. La propiedad de un conjunto de especialización que causa que éste tenga todos los atributos del conjunto de generalización.

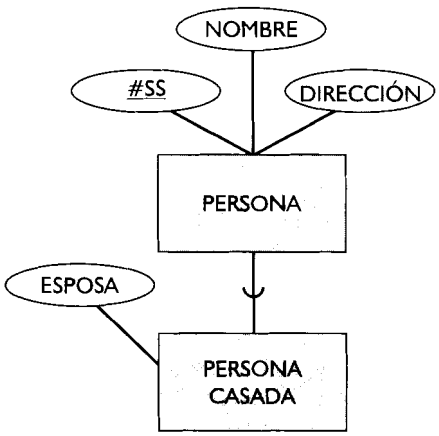
Especialización/Generalización y Atributos. Si un objeto es una especialización de otro objeto, entonces el objeto especializado **hereda** todos los atributos y las interrelaciones del objeto que especializa. PERSONA CASADA, por ejemplo, es una especialización

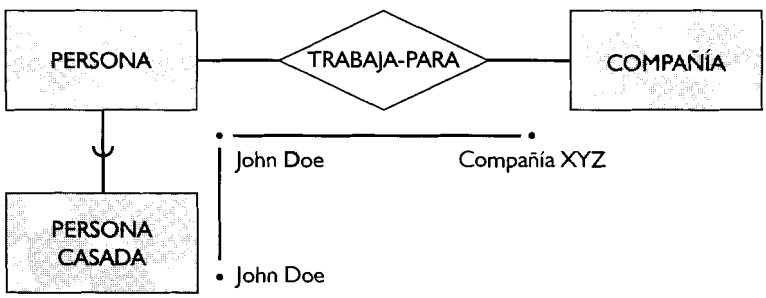


PERSONA. Por lo tanto, una persona casada, por su condición de ser una persona, tiene un nombre, un número de seguridad social, una dirección y otros. El conjunto de objetos PERSONA CASADA hereda esos atributos del conjunto de objetos PERSONA. Además, el conjunto de objetos especializado puede tener sus propios atributos. Por ejemplo, ESPOSA sería un atributo de PERSONA CASADA, pero no de PERSONA. En la Figura 4.14, se ilustran estos conceptos.

Una especialización no sólo hereda los atributos, sino que también hereda las interrelaciones. La Figura 4.15 muestra que PERSONA está relacionada con COMPAÑÍA por la vía de TRABAJA-PARA. PERSONA-CASADA al ser una especialización de PERSONA, también está relacionada con COMPAÑÍA por la vía de TRABAJA-PARA. Suponga que John Doe es una persona casada que trabaja para la Compañía XYZ. Hay aquí entonces un punto en PERSONA-CASADA representando a John Doe y un punto en COMPAÑÍA representando a la compañía XYZ. John Doe en PERSONA-CASADA está relacionado con John Doe en PERSONA, quien a su vez está relacionado con la compañía XYZ. Consecuentemente entonces John Doe en PERSONA-CASADA está relacionado con la compañía XYZ.

La herencia de atributos e interrelaciones es un concepto importante, puesto que permite definir subconjuntos de conjuntos de objetos, los cuales tienen atributos e interrelaciones propias, pero que conservan todos los atributos e interrelaciones de su superconjunto. Esto hace posible modelar la realidad de manera mucho más precisa que si no dispusiéramos del concepto de herencia.





▼ Ejemplos

A continuación se tratarán de crear los modelos conceptuales de datos para algunos problemas del mundo real.

Ejemplo 1: El modelo de datos de un banco

El primer ejemplo puede usarse para responder a algnnas de las preguntas que se hicieron al comienzo del capítulo. Estamos interesados en crear el modelo conceptual de datos del negocio bancario que reflejará la realidad de Robert Goldthumb, presidente de Alchemical Bank and Trust (ABT).

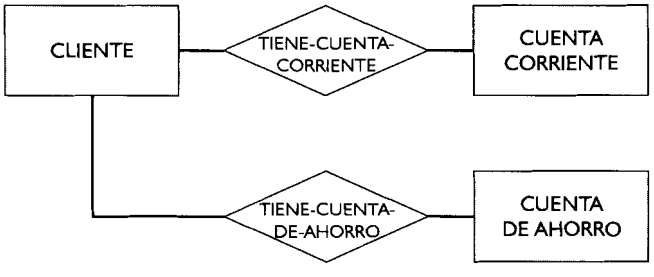
El banco tiene cuentas corrientes, cuentas de ahorro y clientes [Figura 4.16(a)]. Las interrelaciones apropiadas entre éstas se muestran en la Figura 4.16(b). Estamos ahora en posición de responder a las preguntas siguientes:

¿Cuántas cuentas corriente hay? ¿Cuántas cuentas de ahorro? ¿Cuántos clientes?

Las respuestas a estas preguntas se obtienen simplemente contando las instancias en cada uno de los tres conjuntos de objetos. Disponiendo del software apropiado, Goldthumb podría usar su computadora personal para plantear estas preguntas en cualquier momento, o podría recibir un informe periódico.

Obsérvese la forma tan limpia en que la base de datos maneja estas preguntas en comparación con un sistema tradicional basado en archivos. En un sistema basado en archivos, sin las conexiones interarchivos provistas por la base de datos, podría perfectamente haber sólo dos archivos —uno para las cuentas corrientes y otro para las cuentas de ahorro—. En cada uno de estos archivos, la información de los clientes estaría inmersa en un cierto número de campos (nombre del cliente, dirección y otros). La tercera pregunta —"Cuántos clientes"— sería difícil de responder, puesto que tendríamos que extraer los datos de los clientes de los dos archivos, ordenarlos y obviar los duplicados. Sin embargo, en una base de datos, estos clientes pueden mantenerse separadamente y preservando las conexiones deseadas con la información de las cuentas.

(a) Objetos de un Banco



(b) Interrelaciones simples entre los objetos de un Banco

¿Qué clientes tienen tanto cuenta corriente como cuenta de ahorros?

Esta pregunta sólo puede responderse mirando las interrelaciones. Un cliente tiene cuenta de ahorros sólo si está relacionado mediante TIENE-CUENTA-DE-AHORRO con una instancia en CUENTA-DE-AHORRO. Similarmente, un cliente tiene cuenta corriente si está relacionado mediante TIENE-CUENTA-CORRIENTE con una instancia de CUENTA-CORRIENTE. Por último, un cliente tiene *ambos* tipos de cuenta si está relacionado mediante estas interrelaciones con instancias de CUENTA-DE-AHORRO y CUENTA-CORRIENTE. Para responder a la pregunta anterior simplemente hay que contar a los clientes que están relacionados de las dos maneras.

Cardinalidades. La Figura 4.16(b) omite intencionalmente las cardinalidades. Éstas se verán a continuación. Supóngase que se indican las cardinalidades como se muestra en la Figura 4.17. Estas cardinalidades indican que un cliente puede tener no más de una cuenta de ahorro y una cuenta corriente. Para cada cuenta hay un cliente.

Estas cardinalidades pueden no ser un reflejo preciso de la realidad. Considérese la cardinalidad próxima a CUENTA-CORRIENTE. ¿No puede un cliente tener más de una cuenta corriente? ABT, como la mayoría de los bancos, permite que un cliente tenga más de una cuenta corriente, pero las cardinalidades de la Figura 4.17 no permiten esto.

Veamos las otras cardinalidades. ¿Es realista suponer que una cuenta no puede estar asignada a más de un cliente?. Esto tampoco es deseable, ya que las cuentas comunes, como entre esposo y esposa o entre padre e hijo, son muy frecuentes. Para reflejar más precisamente la percepción de la realidad se actualiza la Figura 4.17 y se obtiene la Figura 4.18.

El modelo de la Figura 4.17 es incorrecto porque no refleja nuestra percepción del banco ABT. Una percepción diferente de la realidad podría hacer que el modelo de la Figura 4.17 fuese correcto. Por ejemplo, un banco puede decidir que ningún cliente pueda tener más de una cuenta de un mismo tipo y que no haya cuentas comunes. En este caso, la

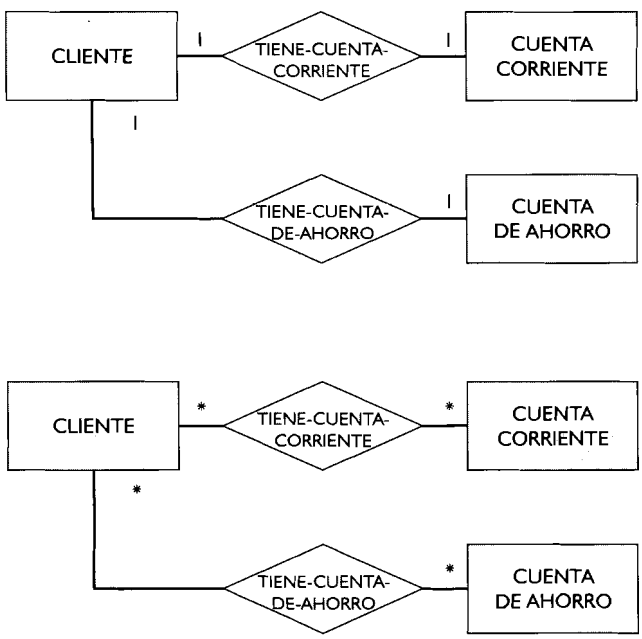


Figura 4.17 representaría correctamente la realidad de ese banco. Un modelo está bien o está erróneo sólo según represente correcta o incorrectamente la realidad en la que estamos interesados.

Retomando el modelo de la Figura 4.18, se puede ahora dar respuesta a preguntas adicionales:

¿Cuántos clientes tienen múltiples cuentas corrientes? ¿Cuántas cuentas corrientes compartidas existen? ¿Cuántos clientes con múltiples cuentas corrientes tienen una cuenta de ahorros?

Veamos la solución a la primera de estas preguntas. Un cliente tendrá múltiples cuentas corrientes si ese cliente está relacionado mediante TIENE-CUENTA-CORRIENTE con al menos dos instancias diferentes en CUENTA-CORRIENTE. Se puede responder a esta pregunta examinando cada instancia en CLIENTE para ver si está relacionada e ir contando las instancias con las que está. Se recomienda al lector tratar de ver cómo responder a las preguntas anteriores usando el diagrama de la Figura 4.18.

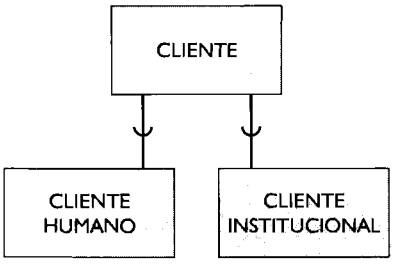
Especializando los Clientes de un Banco. ¿Los clientes de un banco siempre tienen que ser personas? Por supuesto, algunos clientes de los bancos son organizaciones: de negocios, no lucrativas, religiosas, agencias gubernamentales. ¿Desea Goldthmb distinguir entre los diferentes tipos de clientes? Sí, lo desea, ya que los diferentes tipos de clientes tienen atributos distintos. Además las cuentas de los distintos tipos de clientes pueden tener diferentes características. La Figura 4.19 muestra dos especializaciones de CLIENTE. CLIENTE HUMANO para aquellos clientes que son personas y CLIENTE INSTITUCIONAL para aquellos clientes que son organizaciones o empresas.

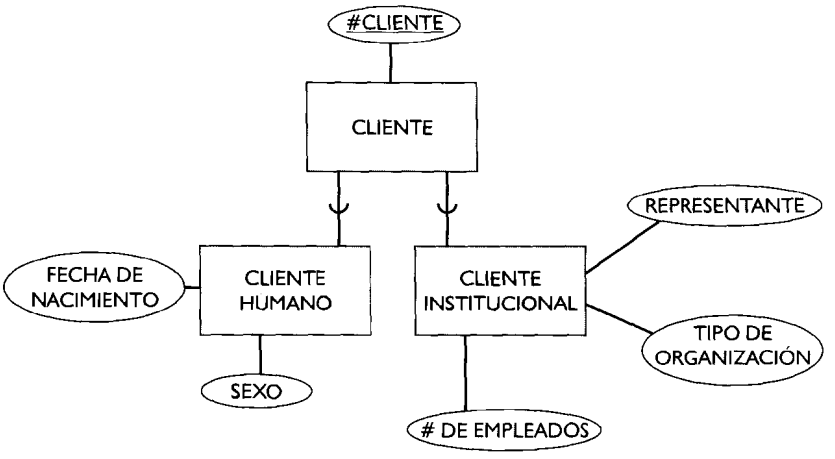
Una de las ventajas principales de usar la generalización y la especialización es que podemos crear diferentes atributos para las especializaciones de un conjunto de objetos, mientras que al mismo tiempo mantenemos los atributos comunes de su nivel más general. La Figura 4.20 muestra que cada cliente tiene un número de cliente, el cual puede ser utilizado como una clave, pero los clientes humanos tienen atributos diferentes que los clientes institucionales.

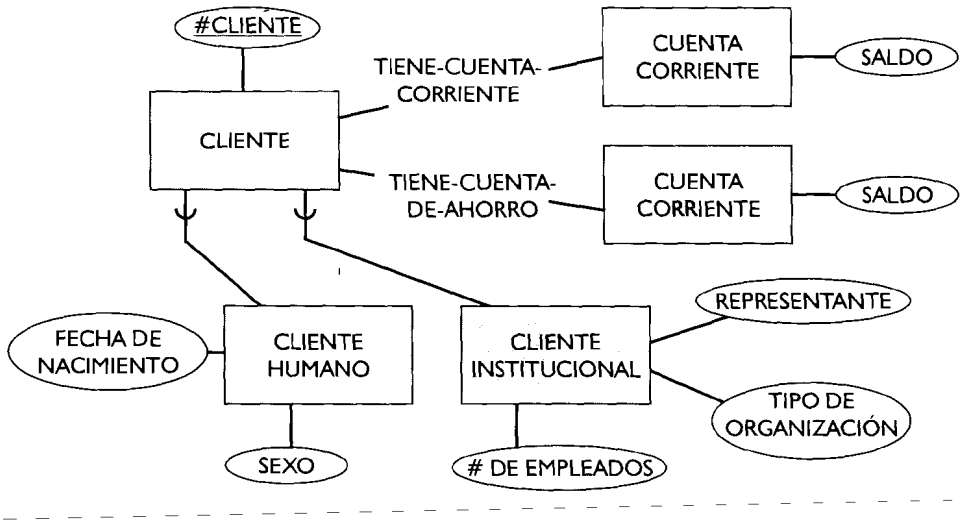
Revisemos la Figura 4.18 para reflejar las especializaciones de CLIENTE. Esta revisión se muestra en la Figura 4.21, que es una composición de las Figuras 4.18 y 4.20. (En esta figura se ha omitido la indicación del sentido en las interrelaciones, lo cual se hará usualmente a partir de ahora.) Se ha añadido un atributo SALDO a cada uno de los conjuntos de objetos cuenta. Ahora se puede responder a algunas preguntas más:

¿Cuál es el porcentaje de cuentas de ahorro que tienen un saldo de más de $1.000? ¿Qué tipos de clientes tienden a tener el mayor saldo promedio en sus cuentas corrientes?

La respuesta a la segunda pregunta depende de lo que signifique "tipos de clientes". El diseño de la base de datos permite distinguir entre clientes humanos e institucionales. Dentro de CLIENTE-INSTITUCIONAL se pueden hacer distinciones usando el atributo







TIPO-DE-ORGANIZACIÓN. Por ejemplo, el TIPO-DE-ORGANIZACIÓN podría ser Negocios, No lucrativa, Religiosa, o Agencia Gubernamental. Para responder a la segunda pregunta comenzamos con CLIENTE-HUMANO y recorremos, por ser a su vez CLIENTE, CUENTA CORRIENTE por la vía de TIENE-CUENTA-CORRIENTE. Esto se hace para cada cliente humano y se registra el saldo. Al terminar se computa el saldo promedio de los clientes humanos. Luego se sigue el mismo procedimiento con CLIENTE-INSTITUCIONAL. Para responder finalmente a la pregunta se comparan los dos promedios.

Ejemplo 2: Huerto frutícola de Stratton

Vern Stratton es un horticultor que está en el negocio de las frutas desde hace cincuenta años. Antes que él, su padre y su abuelo fueron dueños de sus huertos y previeron que al

menos uno de los nietos los heredaría. Ellos tienen excelentes registros de datos desde el siglo XIX que podrían constituir la base para un comprensivo sistema de información. Vern está ahora interesado en las respuestas a preguntas como:

> ¿Cuántas variedades de melocotones tenemos en el huerto Springtown? ¿Cuántos árboles murieron como promedio cada año en el huerto Lee's Valley? ¿Cuál es el promedio de edad de mis manzanos? ¿De cuántos árboles de ciruelo tengo más de una variedad?

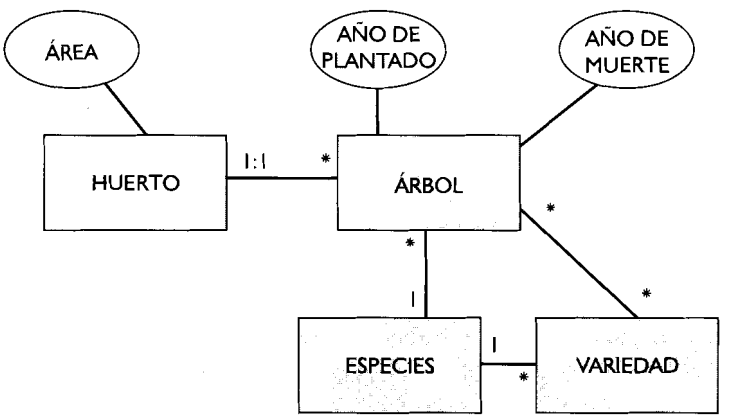
La Figura 4.22 muestra un simple modelo de datos que puede utilizarse para responder a estas preguntas. El conjunto de objetos HUERTO contiene una instancia para cada huerto. El atributo ÁREA describe el huerto. De este modo, ÁREA tendría valores tales como Springtown, Lee's Valley y otros. Cada huerto está relacionado con los árboles (instancias de ÁRBOL) que están en el huerto. Por lo tanto, las instancias de ÁRBOL tienen los árboles físicos en específico en lugar de tipos de árboles. Cada árbol fue plantado en un determinado año y puede o no haber muerto ya. Si el árbol murió, entonces AÑO DE MUERTE contiene un valor; de lo contrario, es nulo.

Los árboles tienen especies y las especies tienen variedades. Por ejemplo, manzana es una especie y Jonathan y Red Delicious son variedades. Puesto que los árboles pueden tener ramas injertadas, una misma especie de árbol pudiera soportar más de una variedad. De esta manera, un manzano que fue originalmente Red Delicious podría también tener Jonathan y Roman Beauty. Cada árbol tiene una sola especie, pero podría tener múltiples variedades.

Para responder a la primera pregunta comencemos con el conjunto de objetos HUERTO. Usando el atribnto ÁREA se identifica el huerto Springtown. Recorriendo el conjnuto de objetos ÁRBOL se identifican todos los árboles en el hnerto. De aquí seguimos a ESPECIES y descartamos todos aquellos árboles que no sean melocotones. Ahora desde ÁRBOL recorremos con todos los melocotones de Springtowu el conjunto de objetos VARIEDAD e identificamos las variedades de melocotones del huerto Spriugtown.

Ejemplo 3: Un problema de lógica

El lector debe de haber visto los problemas de lógica que aparecen en la sección de entretenimieutos de los periódicos y revistas. Usualmente esos problemas no se pueden resolver completamente con los principios de este capítulo, pero se pneden simplificar definiendo

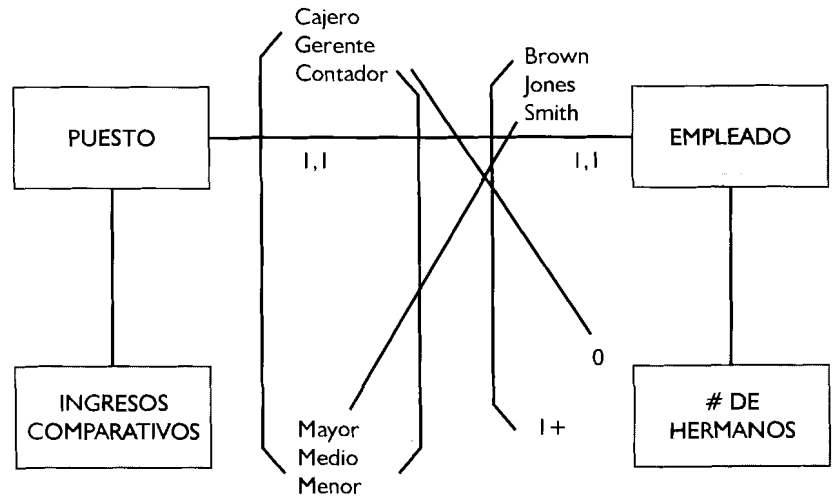


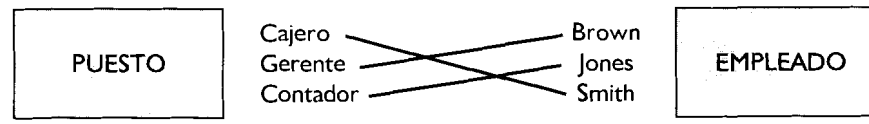
apropiadamente los objetos y las interrelaciones. El caso siguiente ha sido tomado de Wylie (1957, p. 1).

> En un cierto banco, los puestos de cajero, gerente y contador están ocupados, aunque no respectivamente, por Brown, Jones y Smith.
>
> El contador, que es quien menos gana, es hijo único.
>
> Smith, quien está casado con la hermana de Brown, gana más que el gerente.
>
> ¿Qué puesto ocupa cada hombre?

La Figura 4.23 representa este problema gráficamente. Tenemos cuatro objetos y sus correspondientes interrelaciones. Note que las cardinalidades mínima y máxima para la interrelación entre POSICION y EMPLEADO son 1,1 en ambas direcciones. Al hacer las asignaciones para esta interrelación se tendrá la solución a este problema. Se trata de asignar a cada empleado su puesto. Están listados los tres puestos y los tres empleados cerca de sus conjuntos de objetos. Algunos indicios del problema nos hablan de la comparación de los ingresos de los empleados y de cuántos hermanos éstos tienen. Esta información se indica en el diagrama.

La Figura 4.24 da la solución a este problema. Ésta se deduce en la forma siguiente: Puesto que Smith es el que más gana, no puede ser ni el gerente ni el contador. Por lo tanto, Smith es el cajero. Brown tiene un hermano (una hermana casada con Smith), por tanto no puede ser el contador porque éste no tiene hermanos. Luego Brown es entonces el gerente. Consecuentemente, Jones debe ser el contador.





▼ Construir modelos conceptuales de datos a partir de informes existentes

Los modelos desarrollados en este capítulo están basados en la información implícita que los *administradores* podrían hacer. Consecuentemente, estos sistemas forman la base para los *sistemas de gestión de información*. Sin embargo, también hay interés en los modelos que podrían usarse en los *sistemas de procesamiento de datos*, los cuales procesan las transacciones que ocurren a diario en la mayoría de los negocios. En esta sección se examinarán dos formas de informes que se utilizan en las transacciones por mucha gente de los negocios y muestran cómo los modelos conceptuales de datos se pueden deducir de estas formas. Estas formas se usan en el Sistema de Consultoría Manwaring, un caso de estudio que será introducido ahora y que se usará en varios capítulos futuros para ilustrar los pasos en el diseño e implementación de bases de datos.

Caso: Servicios de Consultoría Manwaring

Joan Manwaring, CPA, ha operado los Servicios de Consultoría Manwaring durante los últimos diez años. Emplean a seis consultores, que llevan a cabo proyectos de consultoría a los clientes de Manwaring. Cada proyecto involucra a uno o más consultores y puede durar varias semanas o meses, dependiendo del alcance del proyecto.

Estimados. *Para cada contrato que asumen, Joan debe hacer una propuesta de servicios. La propuesta incluye, entre otras cosas, alcance, objetivos, estructura de la tarea y honorarios. Los honorarios que Joan establece pueden variar mucho según los diferentes tipos de contratos. Los honorarios se basan en las ganancias que se le proporcionen al cliente, y en el tiempo y el esfuerzo dedicados a completar el contrato. Toda la información que pertenece al contrato se conserva para referencias futuras. Cualquier ajuste que se le haga al estimado se le mostrará al cliente y se registrará.*

Recepción de efectivo. *Aunque muchos de los contratos pequeños se pagan en efectivo, la mayoría de los clientes pagan sobre cuenta. El pago debe hacerse al terminarse el contrato, a menos que se hayan hecho arreglos de crédito. Los clientes usualmente pagan sus créditos a tiempo, pero Joan a veces les envía notificaciones a los clientes para que realicen un pago.*

Desembolsos de efectivo. *Aunque muchos suministros se cargan directamente a un contrato específico, algunos suministros y equipamiento se asocian con múltiples contratos o gastos generales. Todos los suministros se compran a cuenta.*

Un modelo de datos para compras

La forma de la orden de compra de Manwaring que se usa para ordenar los suministros se muestra en la Figura 4.25. Esta incluye el nombre y dirección del vendedor, la fecha, el número de orden y el número del vendedor. También da el número de inventario, el producto y el precio de cada producto. El total, incluyendo impuestos, se muestra en la parte de abajo de la figura.

A partir de este formulario se pueden deducir los siguientes conjuntos de objetos: VENDEDOR, ORDEN y PRODUCTO. Los atributos de estos tres conjuntos, así como las interrelaciones entre éstos, se muestran en la Figura 4.26. Note las cardinalidades de las

SERVICIO DE CONSULTORÍA
MANWARING
950 MAIN
EASTON, PA 11111

ORDEN DE COMPRA

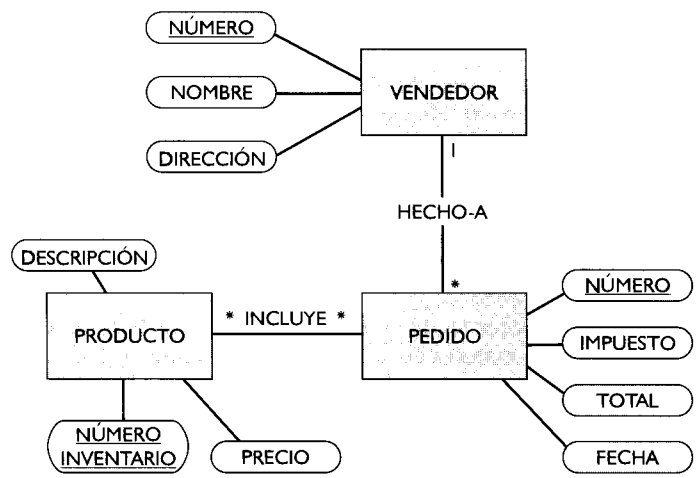
Fecha	Número de orden	Número de vendedor
19/3	384	23

#Inventario	Descripción del producto	Precio
3821	Caja de lápices #2	4,00
4919	Caja de blocs de papel	8,90
	Impuesto	0,77
	Total	13,67

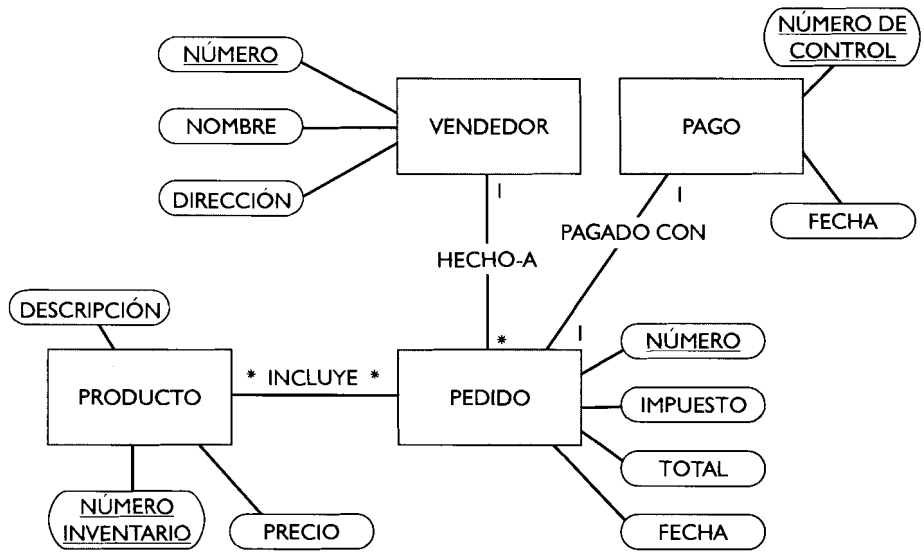
Vendedor:

Oficina Consolidada de Suministros
414 S. Choctaw Drive
Flagship, PA 12345

interrelaciones. La interrelación entre ORDEN y VENDEDOR tiene una cardinalidad uno-muchos porque cada orden se le hace a *un* vendedor, pero un determinado vendedor puede recibir *muchas* órdenes. La interrelación INCLUYE entre ORDEN y PRODUCTO es muchos-muchos porque una orden incluye *muchos* productos y un producto puede aparecer en *muchas* órdenes.



Eventualmente se puede hacer el pago de una orden. Para registrar esta información se aumentará el modelo al que se muestra en la Figura 4.27. Se ha añadido el conjunto de objetos PAGO con los atributos NÚMERO DE CONTROL y FECHA. La cardinalidad uno-uno indica que cada orden se pagará con un cheque, y que cada cheque pagará una sola orden.



Un modelo de datos para la facturación de proyectos

Cuando se completa un proyecto, Manwaring a menudo envía una factura por los servicios prestados y por los suministros utilizados en el proyecto. En la Figura 4.28 se muestra un ejemplo de factura. La factura incluye fecha, número de factura y título del proyecto, así como una pormenorización de los servicios y suministros cargados al proyecto.

En este formulario se pueden identificar los conjuntos de objetos CLIENTE, PROYECTO y CARGO con sus atributos. En la Figura 4.29 se muestran éstos junto con sus interrelaciones. Nótese que hay dos *tipos* diferentes de cargos: los cargos por servicio de consultoría y los cargos por suministros. Puesto que los cargos por servicios de consulta incluyen la identificación del consultor, el conjunto de objetos CARGO se ha dividido en dos subconjuntos: SERVICIO y CARGO POR SUMINISTRO. El conjunto de objetos CARGO tiene dos atributos, CANTIDAD y DESCRIPCIÓN, que son heredados por ambos subconjuntos. Además, la especialización SERVICIO del conjunto CARGO tiene el atributo CONSULTANTE.

Una orden de compra y los formularios de facturas ilustran el proceso de cómo usar los informes existentes para definir el modelo conceptual de los datos. El analista de la base de datos obtiene los datos necesarios para el diseño de la base de datos usando la amplia gama de informes existentes que contribuyen a la creación del modelo conceptual. Además, el analista utiliza un proceso de entrevistas para precisar los requisitos de información adicional mediante la identificación de los tipos de preguntas que los administradores necesitan para responder a las preguntas. Los datos que se obtienen de estos informes y de las entrevistas se combinan en el diseño de un modelo conceptual comprensivo de los datos. Este modelo será implementado eventualmente como un esquema de base de datos y será la base tanto para el procesamiento de los datos como para la gestión del sistema de información.

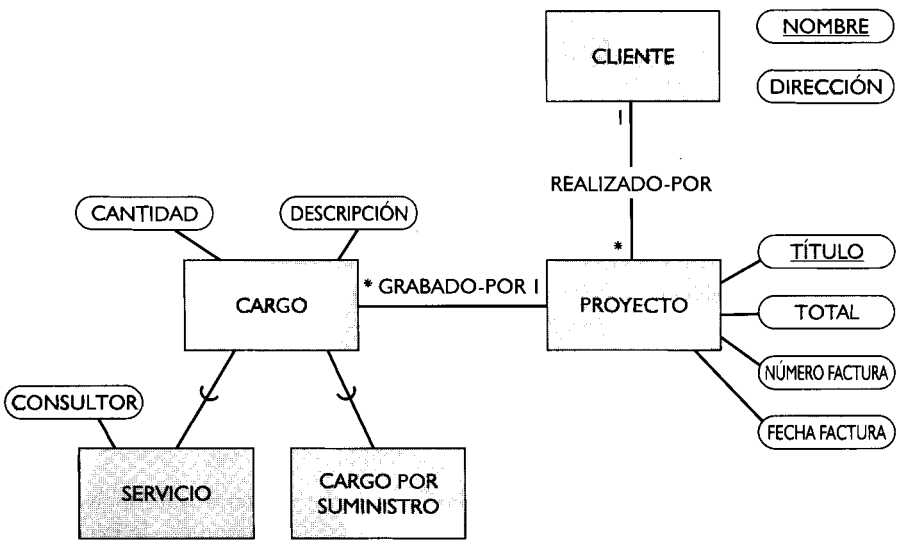
SERVICIO DE CONSULTORÍA MANWARING
950 MAIN
EASTON, PA 11111

FACTURA

Fecha	Número de factura	Proyecto
27/10	342	Sistema de control de Inventario

Consultor	Descripción de cargo	Cantidad
Rodríguez	80 horas de análisis y diseño	4.800,00
Harris	200 horas de programación	8.000,00
Farasapoulos	30 horas de entrenamiento al usuario	900,00
	Suministros (papel, fotocopias, etc.)	40,00
	Total	13.740,00

Cliente:
Sterohouse Markets
318 Elm Street
Morgantown, PA 11222



▼ Agregación

El modelo conceptual de datos desarrollado anteriormente aplica los conceptos básicos del modelado conceptual de datos. En el resto de este capítulo se ampliarán estas técnicas, relativamente simples, para afrontar situaciones más complejas, las cuales surgen natu-

ralmente en los negocios. En esta sección se centrará la atención en los conjuntos agregados de objetos y en la próxima en los conjuntos conceptuales de objetos. Estos dos conceptos ocurren a menudo en las situaciones prácticas. Comprender estas técnicas permite incrementar las habilidades para llevar a cabo la definición de los requisitos y los pasos del diseño conceptual del ciclo de vida del desarrollo de una base de datos.

Aunque los modelos que han sido creados con los conceptos básicos de modelado son relativamente sencillos, es fácil apreciar su poder y utilidad. Sin embargo, la mayoría de los problemas que realmente se encuentran en los negocios son considerablemente más complejos y a menudo involucran el uso de un **agregado**, una interrelación vista como un conjunto de objetos, o como una **interrelación de alto nivel**, que involucra a tres o más conjuntos de objetos.

agregado. Una interrelación vista como un conjunto de objetos.

interrelación de alto nivel. Una interrelación entre tres o más conjuntos de objetos.

Ya se ha visto que una interrelación se puede usar como un conjunto de objetos. Por ejemplo, cuando se definieron y discutieron anteriormente las interrelaciones, se pudo apreciar que cada hombre y cada mujer que estaban relacionados por la interrelación ESTÁ-CASADO-CON constituyen un *matrimonio*, que es en sí un objeto. Como tal, el matrimonio puede tener sus propios atributos, tales como el aniversario de bodas, el total de ingresos, la dirección. Es más, puede participar en otras interrelaciones, tales como DUEÑO-DE AUTOMÓVIL y SON-PADRES-DE. De esta manera, la interrelación ESTÁ-CASADO-CON puede verse como un conjunto de objetos cuyos elementos son los matrimonios.

Esto es cierto para cualquier interrelación. Las interrelaciones pueden ser vistas como objetos y pueden tener atributos y participar en otras interrelaciones. Como se dijo, tales interrelaciones se llaman agregados. Gráficamente se representará un agregado dibujando una caja alrededor de la interrelación y sus conjuntos de objetos participantes (Figura 4.30). Algunas veces, por conveniencia, se le dará al agregado un nombre de objeto —un nombre— además del nombre de la interrelación. Por ejemplo, en la Figura 4.30, MATRIMONIO es el nombre del conjunto de objetos que se le da a la interrelación ESTÁ-CASADO-CON. Esto es razonable si la interrelación se usa como un conjunto de objetos.

Todas las interrelaciones que se han considerado hasta ahora involucran a dos conjuntos de objetos. Tales interrelaciones se llaman **binarias**. Sin embargo, las interrelaciones también pueden involucrar tres o más conjuntos de objetos. Estas interrelaciones de *alto-nivel* se denotan como **interrelaciones n-arias**, donde n es el número de conjuntos de objetos que se están relacionando. Una 3-aria interrelación se llama *ternaria*. Sin embargo, para utilizar una terminología más comprensible, a menudo nos referiremos a las interrelaciones 3-aria o 4-aria como interrelaciones de tres-vías o de cuatro-vías.

interrelación binaria. Una interrelación entre dos conjuntos de objetos.

interrelación n-aria. Una interrelación entre n conjuntos de objetos.

Estos conceptos se ilustran con el ejemplo siguiente. Supongamos que Dick Greenberg de International Product Distribution (IPD) desea revisar las ventas de una línea de productos por países. Para auxiliarle se crea el conjunto de objetos PRODUCTO y el conjunto de objetos PAIS y se establece una interrelación SE-VENDIÓ-EN entre ellos [Figura 4.31(a)]. Una instancia en PRODUCTO, digamos "detergente lavaplatos #5", está relacionada con una instancia de PAIS, digamos Inglaterra, si el detergente lavaplatos #5 se vende en Inglaterra. Si se trata a la interrelación SE-VENDIÓ-EN como un conjunto de objetos, entonces podemos ponerle el atributo CANTIDAD para indicar cuántos productos se han vendido en cada país.

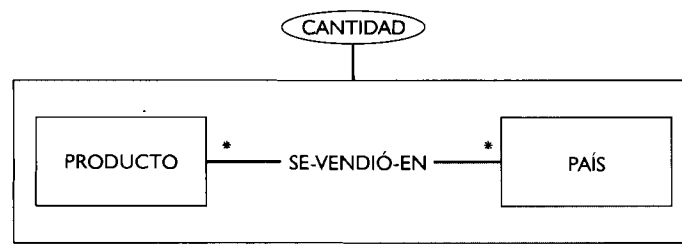
Nótese que el atributo CANTIDAD depende *tanto* de producto *como* de país. Esto es, no se puede determinar el valor de cantidad a partir de producto por separado ni de país por separado —se necesitan ambos—. De ahí por qué CANTIDAD es un atributo de

la *interrelación* entre producto y país en lugar de un atributo de producto o de país por separado. Por esta razón, los modelos de las Figuras 4.31(b) y 4.31(c) son ambos incorrectos. En el caso (b), el modelo no distingue entre cantidades vendidas en los diferentes *países*, y en el caso (c), el modelo no distingue entre las cantidades vendidas de diferentes *productos*.

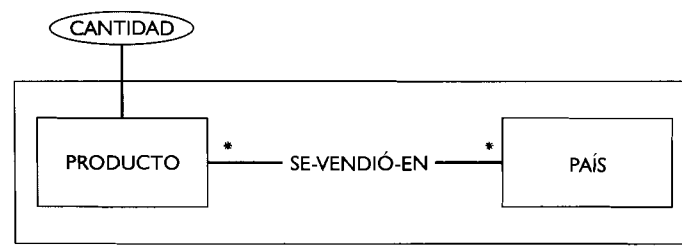
El modelo de la Figura 4.31(a) le permite a Dick recorrer las ventas de productos por países. Supongamos, no obstante, que éste desea una información más refinada de las ventas que la que este modelo puede dar, pues quiere registrar también la cantidad de cada producto vendido en cada país y *en cada día*. Se relaciona entonces SE-VENDIÓ-EN con FECHA y se le da a esta nueva interrelación el atributo CANTIDAD [Figura 4.32(a)]. Una vez más, el atributo se aplica a la interrelación más externa porque las instancias de los tres conjuntos de objetos —PRODUCTO, PAÍS, FECHA— son necesarias para determinar la cantidad.

La Figura 4.32(a) da la solución a este problema en forma de dos interrelaciones binarias, la primera de las cuales (SE-VENDIÓ-EN) es un conjunto de objetos en la segunda interrelación (VENDIDA-EL). Podría considerarse que es más conveniente expresar este modelo como una única interrelación de tres-vías, como se muestra en la Figura 4.32(b). De nuevo podemos ver que CANTIDAD es un atributo de la interrelación entre los tres conjuntos de objetos.

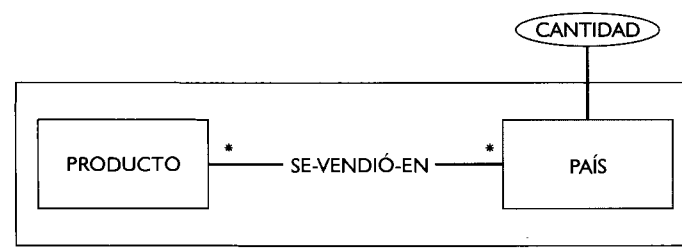
Cualquier interrelación de alto-nivel puede dividirse en una serie de interrelaciones binarias anidadas. Sin embargo, algunas de estas interrelaciones binarias pueden no tener sentido para nosotros si tratamos de relacionarlas con algo del mundo real. Por lo tanto, en ocasiones se usan interrelaciones para expresar los conceptos que se tratan de capturar en un modelo particular de datos, ya que éstas se pueden relacionar con el problema.



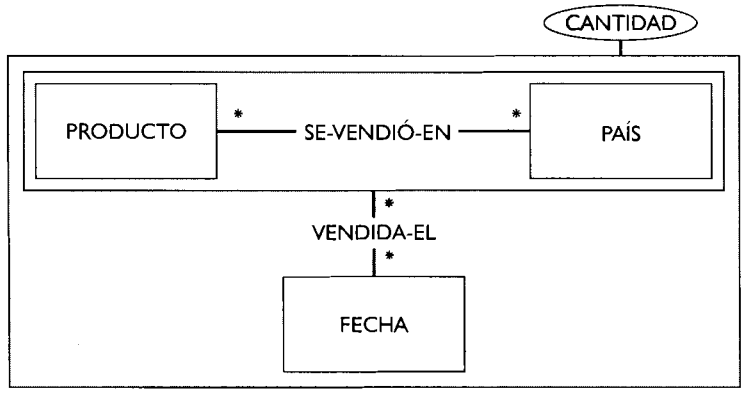
(a) Un modelo correcto para el control de ventas



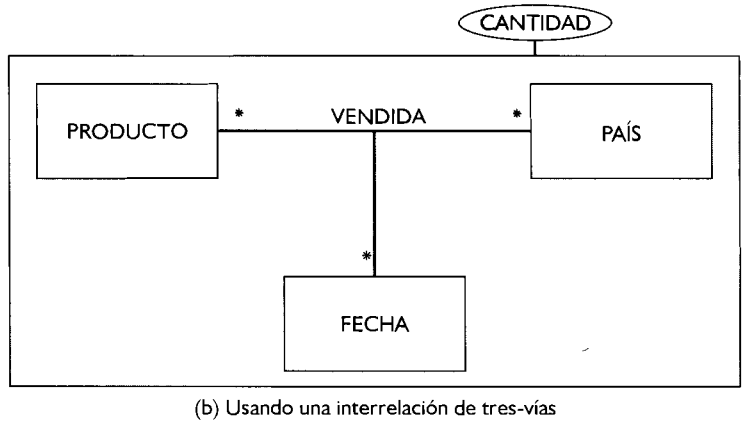
(b) Un modelo incorrecto para el control de ventas



(c) Un modelo incorrecto para el control de ventas



(a) Usando dos interrelaciones binarias



(b) Usando una interrelación de tres-vías

Con respecto al máximo de cardinalidades en las interrelaciones de alto-nivel, se asumirá que todas las interrelaciones binarias que componen la interrelación de alto-nivel son interrelaciones binarias muchos-muchos. Este supuesto viene bien en la práctica la mayoría de las veces.

El poder de estos conceptos se ilustrará a continuación considerando otros ejemplos, todos ellos algo más complejos que aquellos que se han considerado hasta ahora.

Ejemplo 4: Compañía Constructora Premier

La Constructora Premier construye edificios en una gran variedad de sitios. Cada edificio requiere un número de tipos diferentes de materiales en cantidades que varían por edificio. Diferentes cuadrillas llevan a cabo diferentes partes del proyecto. Por ejemplo, puede haber una cuadrilla para la estructura, una para el techo, una para fontanería, una para la albañilería y así sucesivamente. Al planificar la construcción de un edificio Premier asigna diferentes cuadrillas a diferentes fechas. Los trabajadores se asignan a las diferentes cuadrillas según su especialidad. De este modo, Hank Brigman pnede hacer trabajo de carpintería y albañilería, por lo que es asignado a brigadas de estructura, techado y albañilería. El tamaño de una cuadrilla varía de acuerdo al tamaño de los requisitos del edificio. Para una edificio en particular las cuadrillas se irán formando según sea necesario. También se asigna un capataz a cada cuadrilla de cada edificio. Un trabajador puede ser

el capataz en una cuadrilla y un simple trabajador en otra cuadrilla. Marcus Brown, dueño de Premier, desea conocer cuáles de sus trabajadores fueron asignados a cuadrillas para varios edificios, qué materiales están siendo utilizados en los edificios, y para cuándo está planificado el trabajo en cada edificio. A continuación se diseñará un modelo conceptual de datos que puede ofrecer la información que Brown está buscando.

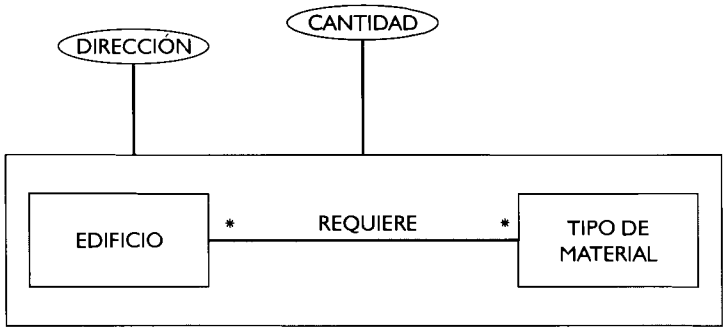
La Figura 4.33(a) modela la interrelación entre edificios y materiales. El conjunto de objetos EDIFICIO contiene una instancia para cada edificio en la base de datos. El conjunto de objetos TIPO DE MATERIAL representa los tipos de material tales como "maderos de 2 × 4 × 10' ", "clavos #10", y otros. Las cardinalidades de la interrelación entre EDIFICIO y TIPO DE MATERIAL indican que cada edificio requiere muchos tipos de materiales y que cada tipo de material se usa en muchos edificios. Obsérvese que el atributo DIRECCIÓN es aplicable sólo a EDIFICIO. La DIRECCIÓN puede usarse como clave para identificar a cada edificio en particular.

El rectángulo alrededor de la interrelación REQUIERE indica que se desea considerar esta interrelación como un conjunto agregado de objetos. A este conjunto de objetos se le da entonces el atributo CANTIDAD. Las instancias de este conjunto agregado de objetos consiste de pares: edificio y tipo de material. Así, por ejemplo, el par formado por el edificio del 610 Fifth St. y maderos de 2 × 4 × 10' puede ser una instancia en la interrelación REQUIERE. A este par se le asigna también una cantidad —digamos 500 piezas—, que es la cantidad de maderos de 2 × 4 × 10' que se requieren para este edificio [ver Figura 4.33(b)].

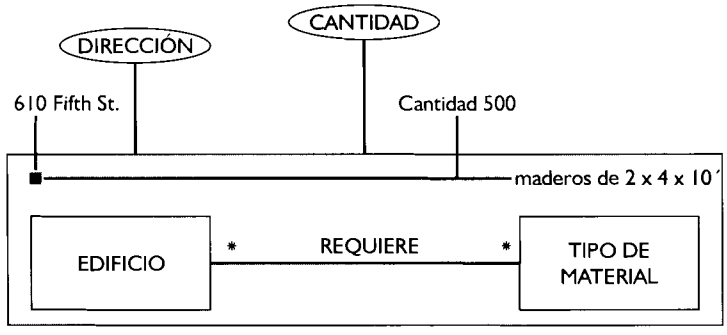
objeto conceptual. Un objeto que representa un tipo de cosa.

objeto físico. Un objeto que representa una cosa física específica.

Es importante notar que el conjunto de objetos TIPO DE MATERIAL en este ejemplo representa a un **objeto conceptual** en lugar de un **objeto físico**. Es decir, cada instancia en TIPO DE MATERIAL representa un *tipo* de material en lugar de una pieza específica de material. Esta noción de objetos conceptuales en lugar de objetos físicos tiene aplicación frecuente en el modelado conceptual de datos. En algunos casos se necesita modelar conjuntos de objetos separados para los objetos físicos.



(a) La interrelación de edificios y materiales



(b) Cantidad de un tipo de material usado en un edificio

Ahora se muestra cómo representar la formación de brigadas y la asignación de trabajadores y capataces a las brigadas. La Figura 4.34 muestra una interrelación entre los conjuntos de objetos TIPO DE CUADRILLA y EDIFICIO. TIPO DE CUADRILLA es otro ejemplo de un conjunto de objetos conceptuales. Esto es, las iustancias de TIPO DE CUADRILLA no representan brigadas *particulares*, sino *tipos* de brigadas, tales como albañilería o techado. La interrelación entre un tipo de brigada y un edificio representa a una brigada en particular —la brigada asignada a ese edificio para realizar la tarea asociada con el tipo de la brigada—. Por lo tanto, se puede ver a esta interrelación como un objeto y darle entonces el nombre CUADRILLA.

Cada cuadrilla, como una instancia en el conjunto de objetos CUADRILLA, tiene planificado trabajar en un número de fechas diferentes. Por ejemplo, los fontaneros requieren de un cierto número de días para hacer la fontanería de un edificio dado. Por tanto, se tiene una interrelación PLANIFICADA-PARA muchos-muchos entre CUADRILLA y FECHA.

La Figura 4.35 muestra la asignación del capataz y de los trabajadores a las brigadas. Nótese que la interrelación ES-CAPATAZ-DE es uno-muchos. Esto es debido a que una brigada tiene un solo capataz, pero un trabajador puede ser capataz de diferentes brigadas. La Figura 4.36 nos da un diagrama compuesto que muestra el modelo completo de los datos para la Compañía Constructora Premier.

Ejemplo 5: Huerto frutícola de Stratton (continuación)

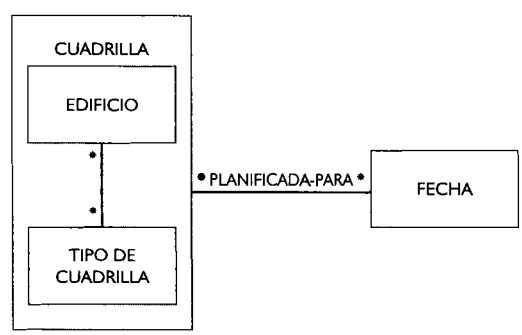
El ejemplo 2 nos da cierta información sobre el negocio frutícola de Vern Stratton. A continuación se dará información adicional que se usará para obtener el modelo de datos correspondiente, que será más complejo que el modelo inicial. Este modelo será más poderoso y permitirá dar la información necesaria para planificar a los trabajadores y la plantación de nuevos árboles, y para obtener otras informaciones de decisión, necesarias para la marcha del negocio.

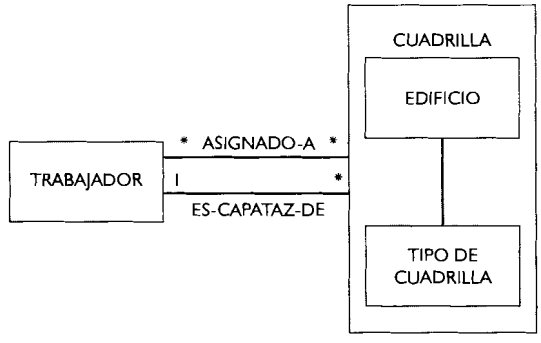
Los árboles en los huertos de Vern se siembran en filas y columnas. Las filas y las columnas están separadas entre sí por 20 pies. Cuando un árbol muere éste se extirpa y otro árbol se siembra en su lugar.

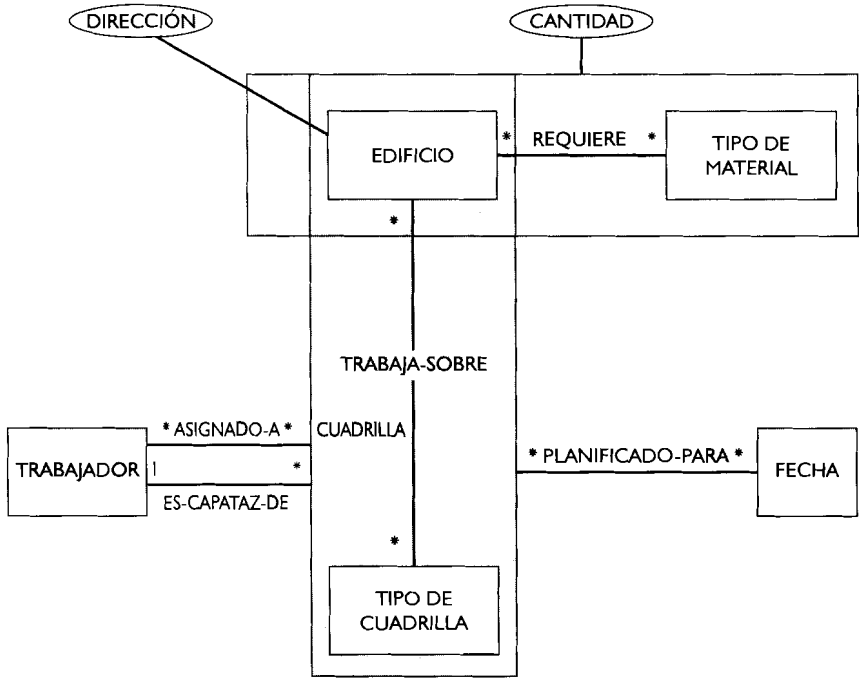
Recuérdese que las especies representan a una amplia categoría de frutas, como manzanas, melocotones o cerezas, y variedades de subcategorías de éstas, como Jonathan y Red Delicious (para las manzanas). Dependiendo de las condiciones climáticas durante los meses iniciales, las variedades retoñan en diferentes momentos. La cosecha comienza una cierta cantidad de días a partir del florecimiento completo de una variedad determinada.

Además de poder manejar esta nueva información, el modelo de datos debe construirse de modo tal que se pueda responder las siguientes preguntas:

¿Cuántos arbustos de manzanas Red Delicious se extrajeron del huerto de Paynesville el año pasado? ¿Cuál fue la fecha promedio de cosecha en todos los huertos para una determinada varie-



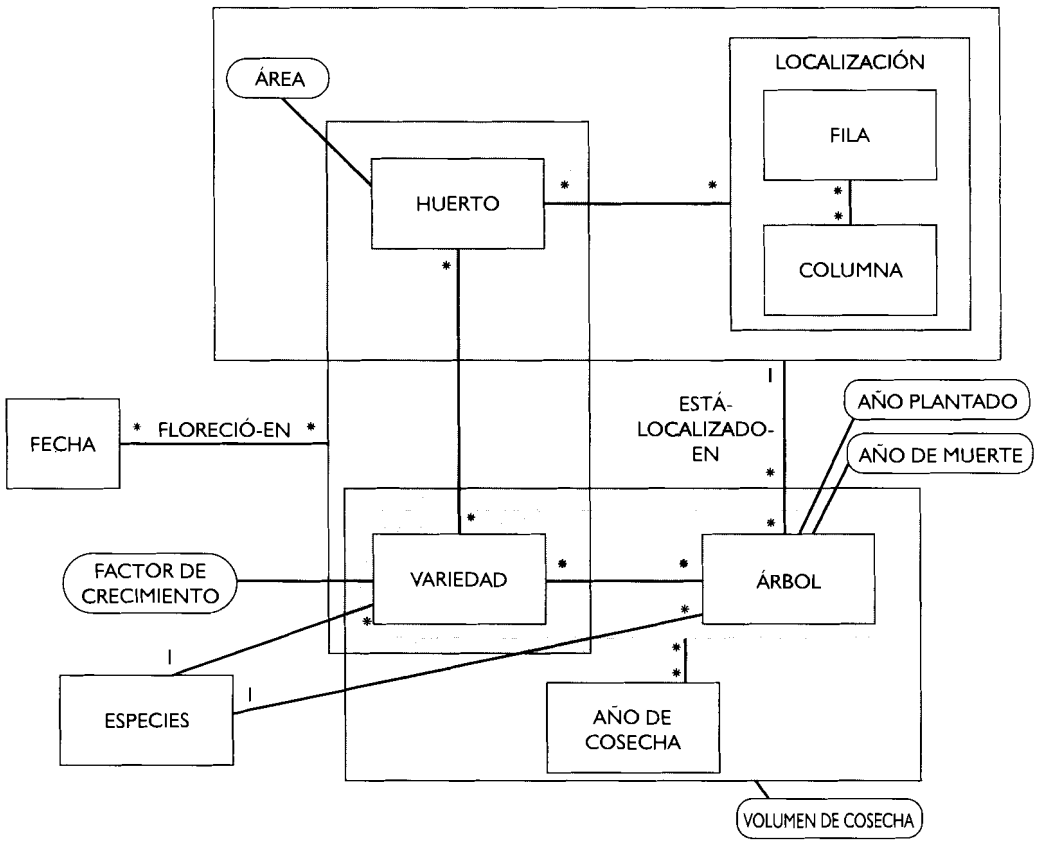




dad de melocotones en los últimos diez años? ¿Cuándo deberán estar listas para la cosecha de este año las manzanas Jonathan del huerto Lee Valley? ¿Cuántos espacios hay libres en cada huerto para plantar nuevos árboles? ¿Cuántos quedarían libres si taláramos aquellos árboles cuyo promedio de producción de los últimos cinco años están por debajo de un cierto rendimiento?

La Figura 4.37 muestra la versión mejorada de la Figura 4.22. El conjunto de objetos LOCALIZACIÓN es otro ejemplo de objeto conceptual. Éste no representa una localización específica, sino que nos da la fila y la columna que puede estar en cualquier huerto. De este modo, una instancia de LOCALIZACIÓN como puede ser (10, 17) representa a la fila 10 y la columna 17 de un huerto *no* especificado. Cuando una instancia como ésta se relaciona con un huerto en específico —digamos por ejemplo el huerto Springtown—, entonces el *triplete* (Springtown, 10, 17) nos da la fila 10, la columna 17 del huerto Springtown.

En la Figura 4.22 sólo se relacionan los árboles con sus huertos. Ahora se pueden



identificar los huertos y las localizaciones específicas (fila y columna) dentro del huerto donde está plantado un árbol. Esto permite dar respuesta a la pregunta sobre el número de espacios vacíos en cada huerto:

La interrelación que liga los árboles con sus localizaciones específicas dentro de un huerto se llama ESTÁ-LOCALIZADO-EN. ¿Por qué es uno-muchos? Está claro que un árbol puede tener una sola localización, pero ¿por qué una localización puede tener muchos árboles? Recuérdese que se conserva el año en que un árbol murió, la base de datos conserva el rastro de todos los árboles que han estado en una localización específica en el transcurso de los años. Lo que se tiene ahora que hacer es identificar todos los árboles asociados con un espacio en particular dentro del huerto. Si todos han muerto, entonces se puede asumir que el espacio está disponible para un nuevo árbol.

Esta información también se puede usar para determinar dónde están las áreas más fértiles dentro de un huerto. Podemos rastrear el número de florecimientos que produce cada árbol en el año. Con el paso de los años, los árboles en las áreas más fértiles producirán más frutas.

Para determinar cuándo es la época de cosecha para una variedad dada en un huerto dado se debe registrar cuándo ha tenido lugar un total florecimiento para dicha variedad en el huerto. Por lo tanto, se deben conectar HUERTO con VARIEDAD en una interrelación y luego conectar estas dos con FECHA en la interrelación FLORECIÓ-EN. Esto nos dice cuándo ha ocurrido un florecimiento total para cada variedad en cada huerto. El atributo FACTOR DE CRECIMIENTO de VARIEDAD nos da el número de días desde que terminó el florecimiento hasta el comienzo de la cosecha. Usando el atributo FAC-

TOR DE CRECIMIENTO en combinación con la interrelación FLORECIÓ-EN podemos obtener cuándo deberá ser la cosecha de cada variedad en cada huerto. Esta información es esencial para planificar a los trabajadores durante la época de cosecha.

Para registrar el volumen total cosechado de cada variedad cada año y para cada árbol, debemos ligar cada instancia de la interrelación (ÁRBOL, VARIEDAD) con AÑO DE COSECHA. Esta interrelación de tres-vías tendrá el atributo VOLUMEN DE COSECHA, que nos dice cuánto se ha cosechado de cada variedad para cada árbol en cada año de cosecha.

Ejemplo 6: Servicios de Consultoría Manwaring (continuación)

Al principio del capítulo se crearon modelos de datos para órdenes de compra y facturas de los Servicios de Consultoría Manwaring. Los formularios utilizados se simplificaron para que se ajustaran a los modelos conceptuales básicos que estaban disponibles. Usando conceptos más avanzados se pueden crear modelos de datos para formatos de informes más sofisticados. Se verán ahora versiones más sofisticadas de las órdenes de compra y de las facturas, y se crearán modelos para las mismas.

La Figura 4.38 muestra una orden de compra mejorada de los Servicios de Consultoría Manwaring. Si se compara este formulario con el de la Figura 4.25 debe notarse que se incluyen nuevas columnas para *Descripción del Producto*, *Cantidad*, *Precio Unitario* y *Precio Total*, mientras que el original sólo tenía Descripción del Producto y Precio. En el formulario original, la cantidad que se ordenaba estaba incluida dentro de la *Descripción del Producto*, mientras que ahora se separan. El *Precio Unitario* no aparecía. El *Precio* del formulario original es lo que ahora es el *Precio Total*.

En este nuevo formulario tiene dos ventajas: (1) Puesto que *Precio Unitario* está en función del producto que se está ordenando, el *Precio Total* se puede calcular automáticamente a partir de *Cantidad* y de *Precio Unitario*. En el antiguo formulario se requería que

SERVICIO DE CONSULTORÍA MANWARING
950 MAIN
EASTON, PA 11111

ORDEN DE COMPRA

Fecha	Número de orden	Número de vendedor
29/3	388	23

#Inventario	Descripción del producto	Cantidad	Precio unitario	Cantidad
3821	Lápices #2	3	4,00	12,00
4919	Blocs de papel legalizado	4	8,90	35,60
	Impuestos			2,86
			Total	50,46

Cliente:

Consolidated Office Suplies
414 S. Choctaw Drive
Flagship, PA 12345

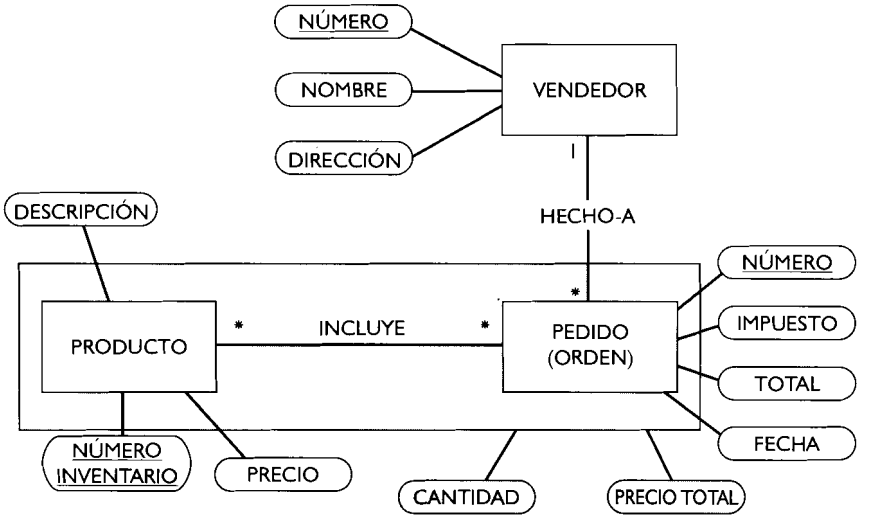
esto se hiciese manualmente. (2) Puesto que *Cantidad* se lista por separado es posible llevar a cabo cálculos con ella tanto en la orden de compra en sí, como en la determinación de la cantidad total ordenada para cualquier producto a lo largo de un período de tiempo. Tales cálculos pueden usarse para responder a preguntas como:

¿Cuántos blocs de papel legal se utilizaron el año pasado?

La Figura 4.39 muestra el modelo de datos que se obtiene de este formulario para órdenes de compra. Obsérvese que se ha añadido la interrelación entre PRODUCTO y ORDEN. CANTIDAD y PRECIO TOTAL son atributos del *agregado* puesto que dependen tanto de PRECIO como de ORDEN. Esto es, la *cantidad* es el número de unidades de un *producto* que están siendo ordenadas en una *orden* en particular. El PRECIO TOTAL es un atributo que se calcula y que se aplica a PRODUCTO y a ORDEN en la misma forma que CANTIDAD. Nótese también que DESCRIPCIÓN, NÚMERO DE INVENTARIO y PRECIO UNITARIO son todos atributos de PRODUCTO, ya que dependeu sólo de PRODUCTO y no de ORDEN. En el nuevo modelo, DESCRIPCIÓN tiene un significado diferente que en el modelo de la Figura 4.26, ya que en este último la DESCRIPCIÓN incluía la cantidad que estaba ordenándose.

La Figura 4.40 muestra una versión mejorada de las facturas. Si se compara esta factura con la de la Figura 4.28 se puede ver que los cargos se han separado en *Cargos de Consultoría* y *Otros Cargos*. En la factura mejorada se muestran *Actividad* y *Horas* en lugar de *Descripción de Cargo*, como en la original. *Descripción de Cargo* era un campo libre en el cual el usuario podía escribir la información que considerase apropiada. Por el contrario, *Actividad* y *Horas* son mucho más precisas. *Actividad* incluye sólo un número específico de actividades predefinidas, tales como análisis de sistema, diseño de sistema, programación y entrenamiento al usuario, con las que los Consultores estarían a cargo. *Horas*, por supuesto, debe ser numérico. Este enfoque hace mucho más fácil que un sistema automatizado calcule el número de horas que cada consultor ha dedicado a cada tipo de actividad para cada cliente.

El modelo de datos para esta factura se muestra en la Figura 4.41. Se han añadido las interrelaciones entre CONSULTOR y ACTIVIDAD, así como la interrelación entre este agregado y PROYECTO. El más grande de los agregados tiene, por lo tanto, los atributos HORA y CANTIDAD. Esto es debido a que el valor del atributo HORAS depende de los



SERVICIO DE CONSULTORÍA
MANWARING
950 MAIN
EASTON, PA 11111

FACTURA

Fecha	Número de factura	Proyecto
27/12	349	Sistema de control de gastos

Consultor	Actividad	Horas	Tarifa	Cantidad
Rodríguez	Análisis de sistema	30	$60/hr	1.800,00
Rodríguez	Diseño de sistema	30	$60/hr	1.800,00
Rodríguez	Programación	20	$60/hr	1.200,00
Chatman	Programación	60	$40/hr	2.400,00
			Total de consulta	7.200,00

OTROS CARGOS

Descripción	Cantidad
Suministros (Papel, Fotocopias, etc.)	35,00
Otro total	35,00
Factura total	7.235,00

Cliente:

Robespierre Manufacturing
1793 Bonaparte Road
Bastille, PA 10000

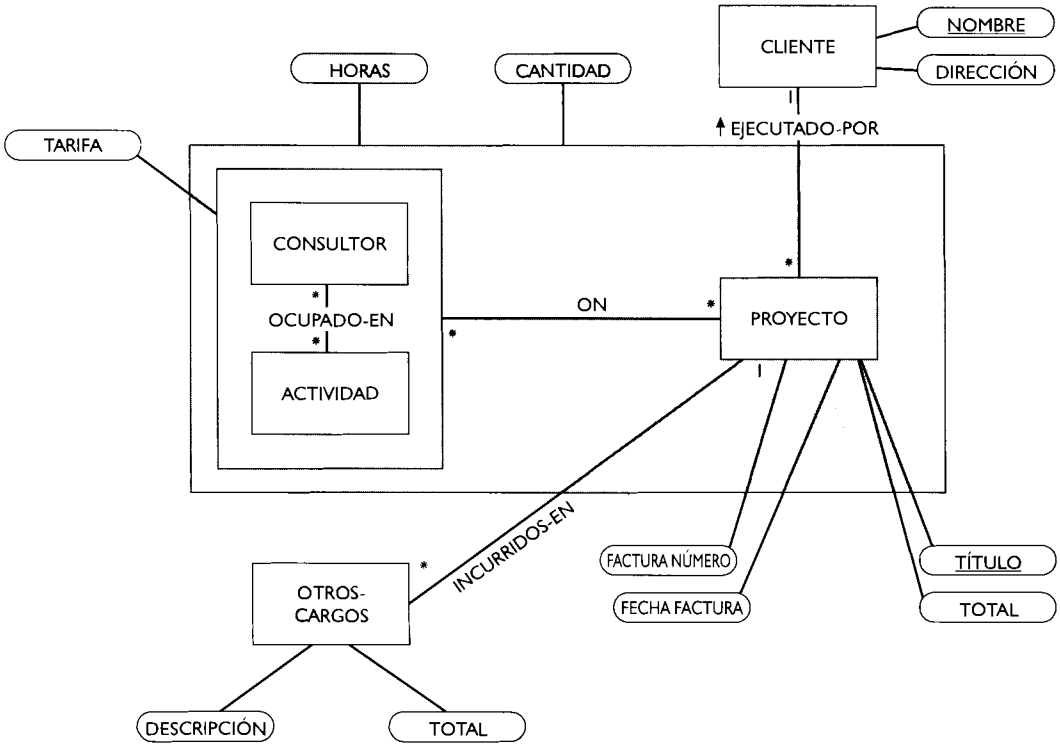
tres factores: consultor, actividad y proyecto. Es decir, el atributo HORAS nos dice cuánto tiempo ha estado un determinado *consultor* a cargo de una *actividad* para un determinado *proyecto*.

Obsérvese que el atributo TARIFA está conectado directamente con el conjunto de objetos CONSULTOR , ya que sólo depende del consultor. Esto es, Manwaring cobra la misma tarifa por horas para un determinado consultor independientemente del tipo de actividad de la que esté a cargo. Esto se muestra en la factura mejorada de la Figura 4.40, se puede ver aquí que la tarifa de Rodríguez siempre es de $60 por hora.

CANTIDAD indica el cargo por el trabajo de un consultor en un actividad y para un proyecto. Esto se calcula multiplicando la *tarifa* (que se toma del atributo TARIFA del consultor) por las *horas* (que se toma del atributo HORAS) para la actividad de dicho *consultor* en el *proyecto*.

Al comienzo del capítulo se vio que Joan Manwaring estaba interesado en un sistema que relacionase consultores, actividades y clientes, de modo que pudiera obtenerse información sobre sus interrelaciones. La Figura 4.41 nos da el modelo de datos necesario. Los datos soportados por este modelo de datos se pueden manipular para crear un gran número de informes, dos de los cuales se muestran en las Figuras 4.42(a) y 4.42(b).

El informe de la actividad de consulta de la Figura 4.42(a) muestra cuántas horas ha invertido cada consultor en cada actividad durante el pasado año. Por ejemplo, Chatman



SERVICIOS DE CONSULTORÍA MANWARING

INFORME DE LA ACTIVIDAD DE CONSULTORÍA
Para el año que termina en diciembre 31, 19__

CONSULTOR	ACTIVIDAD	HORAS
Chatman	Programación	950
	Entrenamiento al usuario	600
	Análisis de oficina	450
Farasopoulos	Entrada de Datos	30
	Conversión de archivos	1.400
	Entrenamiento al usuario	350
	Análisis de oficina	220
Harris	Conversión de archivos	1.140
	Programación	500
	Diseño de sistemas	120
	Análisis de oficina	240
Rodríguez	Programación	150
	Diseño de sistemas	800
	Análisis de sistemas	750
	Entrenamiento al usuario	100
	Análisis de oficina	200

(a) Un informe que relaciona consultores con actividades

SERVICIOS DE CONSULTORÍA MANWARING

INFORME CONSULTOR-CLIENTE
Para el año que termina en diciembre 31, 19__

CONSULTOR	CLIENTE	HORAS
Chatman	Robespierre	60
	Statten	400
	Sunderman	950
	Universal	140
Farasopoulos	Storehouse	30
	Sunderman	1.100
	Watanabe	650
Harris	Goldman	950
	Martino	425
	Storehouse	200
	Universal	185
Rodríguez	Goldman	800
	Martino	840
	Robespierre	80
	Storehouse	80

(b) Un informe que relaciona consultores con clientes

invirtió 950 horas en programación, 600 horas en entrenamiento a los usuarios y 450 horas en las actividades de oficina que no podrían facturarse a los clientes. El informe consultor-cliente de la Figura 4.42(b) muestra cuántas horas ha gastado cada consultor en actividades facturables para cada cliente.

El modelo de datos de la Figura 4.41 podría usarse para obtener una variedad de informes similares. Por ejemplo, podría generarse un informe para mostrar precisamente qué actividades lleva a cabo cada consultor para cada cliente y sobre cuál proyecto. Por supuesto, también podría mostrarse la cantidad de horas que ellos gastan en cada actividad. Otro informe podría ser el del porcentaje medio para cada proyecto de las horas facturables empleadas en cada actividad. Por ejemplo, si el informe muestra que, como promedio, los analistas de sistema ocupan sólo el 5 por 100 del tiempo del proyecto, entonces podrían planificarse entrenamientos adicionales para desarrollar en los consultores mejores habilidades de análisis de sistema.

La agregación y las interrelaciones de alto nivel son herramientas poderosas que tienen aplicación frecuente en el modelado de sistemas complejos en los negocios. En verdad, prácticamente todos los problemas de negocios tienen suficiente complejidad para requerir la aplicación de estos conceptos. Los ejemplos dados en este capítulo ilustran el poder de la agregación y la rica variedad de situaciones en la cual se puede aplicar.

▼ Modelado conceptual de objetos contra objetos físicos

Aunque la agregación y las interrelaciones de alto nivel son herramientas muy útiles en la solución de una amplia variedad de problemas de modelado, hay ciertos problemas en los que los aspectos más difíciles pueden resolverse con herramientas más básicas. En esta sección se verán algunos problemas que surgen debido a las ambigüedades de nuestro lenguaje cotidiano. Como se podrá apreciar una vez que se comprendan y se aíslen los conceptos

involucrados en tales ambigüedades, se podrán resolver los problemas del modelado de datos simplemente definiendo los conjuntos de objetos apropiados. Los agregados y otros conceptos se pueden usar para introducir construcciones adicionales en el modelo de datos según sea necesario.

conjunto de objetos conceptuales. Un conjunto de objetos cuyas instancias son objetos conceptuales.

En la sección anterior se pudieron notar varias instancias de **conjuntos de objetos conceptuales**. Por ejemplo, TIPO DE MATERIAL y TIPO DE CUADRILLA, en el modelo de datos de la Compañía Constructora Premier, son conjuntos de objetos conceptuales, puesto que sus instancias representan *tipos* de cosas en lugar de ejemplos específicos y concretos de tales tipos. Un tipo de material puede ser "maderos de $2 \times 4 \times 10'$ " en lugar de un pedazo de madero en específico. Un *tipo de brigada* podría ser "de techo" o "eléctrica", mientras que una *brigada* en específico pudiera ser "la que está poniendo el techo del edificio en 320 Main Street".

conjunto de objetos físicos. Un conjunto de objetos cuyas instancias son objetos físicos.

A menudo es necesario distinguir entre los conjuntos de objetos conceptuales y los **conjuntos de objetos físicos** que corresponden a éstos, ya que ambos tipos de conjuntos de objetos se necesitan representar en el mismo modelo de datos. Esto se ilustra en el ejemplo siguiente.

El problema de una biblioteca

Un estudiante llama a una biblioteca y pregunta:

ESTUDIANTE: ¿Tienen *The Pickwick Papers* de Charles Dickens?
BIBLIOTECARIO: (Introduce la consulta en el catálogo en línea). No, no lo tenemos.
E: ¿Y *Bleak House*?
B: (Introduce la segunda consulta). No.
E: ¿Cuántos libros tienen de Dickens?
B: (Introduce una tercera consulta). Tenemos doce.
E: ¿De verdad? ¿Cuáles son?
B: Tenemos *A Tale of Two Cities*, copia 1; *A Tale of Two Cities*, copia 2; *A Tale of Two Cities*, copia 3, y así hasta la copia 12.
E: ¡Todos son el mismo libro! No tienen doce libros de Dickens, tienen sólo uno.
B: No, no son todos el mismo. Uno es la Edición Clásica, otro es una traducción al alemán, otro es una traducción al francés, uno es una versión condensada, y así sucesivamente.
E: Pero el hecho cierto es que todos son *verdaderamente* el mismo libro. No importa lo que se pueda haber hecho para ponerlo en diferentes ediciones, sigue siendo *A Tale of Two Cities*. Realmente tienen sólo un libro de Dickens.

Esta conversación, basada en Kent (1978), nunca podría ocurrir puesto que ningún bibliotecario daría los argumentos que el bibliotecario está dando. Sin embargo, sirve para destacar un problema significativo que tenemos con el lenguaje natural que los humanos usan en sus conversaciones corrientes. En este ejemplo, ¿qué es lo que entendemos por *libro*? Sin profundizar demasiado, y fuera del contexto de esta conversación, podemos pensar que "un libro es un libro", y que no debe haber ninguna ambigüedad en el uso de esa palabra. Pero aquí el estudiante y el bibliotecario están usando la palabra libro de dos maneras muy diferentes. Por un lado —para el estudiante—, un libro es algo conceptual que puede tener muchas versiones físicas diferentes. De este modo, *A Tale of Two Cities* es realmente el mismo libro, independientemente si es la copia 1 o la 8, independientemente de si está en inglés o en francés, e independientemente de si es una versión completa o una versión condensada. Por otro lado, el bibliotecario está utilizando (al menos inicialmente) el otro sentido: Un libro es algo físico que podemos tener en nuestras manos, que podemos hojear y poner en un estante. El bibliotecario necesita llevar la cuenta de los libros *físicos* que tiene, independientemente de si es la primera o la decimosegunda copia de un libro *conceptual*.

Algunas veces se distingue entre estas dos formas de uso, insistiendo en que los libros físicos sean denominados *copias* o *volúmenes*. De este modo, se puede decir: "¿Cuántos volúmenes contiene la biblioteca?" Pero como observadores de las conversaciones entre los usuarios debemos reconocer que la gente frecuentemente no observa tales convenios. Ellos dicen simplemente "libro", con lo que algunas veces están queriendo decir "libro conceptual" y otras "libro físico". Para diseñar una base de datos se necesita ser capaz de detectar estas diferencias. En algunos casos, los usuarios se referirán a un *objeto conceptual*, que es una versión abstracta o generalizada de un objeto. En otros casos, los usuarios se referirán a un *objeto físico*, o una instancia específica de un objeto conceptual. Por lo tanto, si se quiere responder a las necesidades de todos los usuarios de bases de datos, los modelos de datos deben capturar esta distinción entre conceptual y físico.

Hay muchas otras distinciones sutiles que captar. En la discusión entre el estudiante y el bibliotecario, el bibliotecario eventualmente acepta que hay una diferencia entre un libro físico y uno conceptual, pero insiste en que, si son ediciones diferentes, los libros son conceptualmente diferentes. Esto es, la Edición Clásica Completa de *A Tales of Two Cities* es un libro conceptualmente distinto de la versión condensada. Sin embargo, el estudiante insiste en que la edición es irrelevante y que conceptualmente el libro es el mismo a través de sus diferentes ediciones.

Lo cierto es que ambas partes tienen puntos de vista legítimos. Puesto que lo que nos interesa es el diseño de bases de datos, no tenemos que determinar cuál de las partes "tiene la razón". Lo que se necesita es conocer a qué clase de preguntas desean los usuarios que el sistema pueda dar respuesta. Una vez que se haya identificado el tipo de información que se necesita, se pueden tomar decisiones sobre el diseño de los datos. Idealmente se quisieran satisfacer todos los puntos de vista, incluyendo *tanto* el del estudiante como el del bibliotecario.

Crear el modelo de datos de la biblioteca

Durante la fase de definición de los requisitos del ciclo de vida del desarrollo de la base de datos (CVBD), como analistas tenemos que realizar entrevistas a los usuarios para determinar sus necesidades y sus expectativas con relación al sistema de base de datos. Durante esta fase es muy importante que se identifiquen correctamente los objetos y las interrelaciones que forman la actividad cotidiana de los usuarios. De este modo, si hay diferencias sutiles en el significado de los diferentes términos que ocurren de manera natural en las transacciones de negocios, tenemos que ser capaces de identificarlas de manera que podamos modelar las interrelaciones con precisión.

Para crear el modelo al problema de la biblioteca hay que considerar las siguientes preguntas:

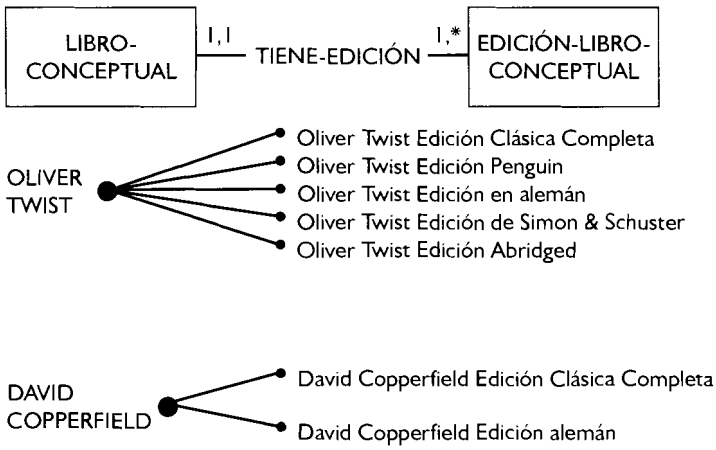
> ¿Cuántos libros de Charles Dickens tiene la biblioteca? ¿Cuántos libros diferentes tiene la biblioteca en la Edición Clásica Completa? ¿Cuántos libros tiene la biblioteca que están en segunda edición? ¿Cuántas copias tiene la biblioteca de "Pride and Prejudice"[2]?

Para estas preguntas se pueden identificar tres tipos de "libros":

Un libro conceptual
Una edición de un libro conceptual
Un libro físico

A partir de las dos primeras se pueden construir dos conjuntos de objetos y una interrelación (Figura 4.43). Fíjese en las cardinalidades mínima y máxima para LIBRO-CONCEPTUAL. Estas cardinalidades muestran que el conjunto de objetos EDICIÓN-

[2] *Orgullo y Prejuicio* (N. del T.).



conjunto de objetos dependiente. Un conjunto de objetos cuyas instancias deben estar relacionadas con al menos otra instancia de otros conjuntos de objetos.

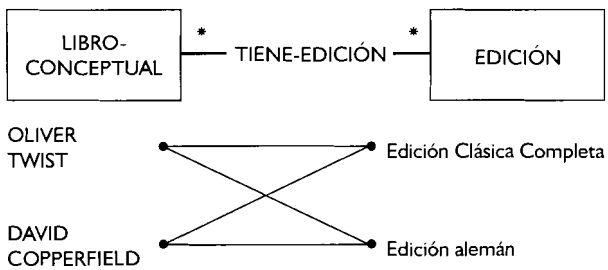
LIBRO-CONCEPTUAL es **dependiente** del conjunto de objetos LIBRO-CONCEPTUAL. Esto es, cada edición de libro conceptual es una edición de uno y sólo un libro conceptual.

Aunque da respuesta a alguna de las preguntas, en definitiva falla porque no es capaz de responder a preguntas tales como:

¿Cuántos libros diferentes tiene la biblioteca en la edición Clásica Completa?

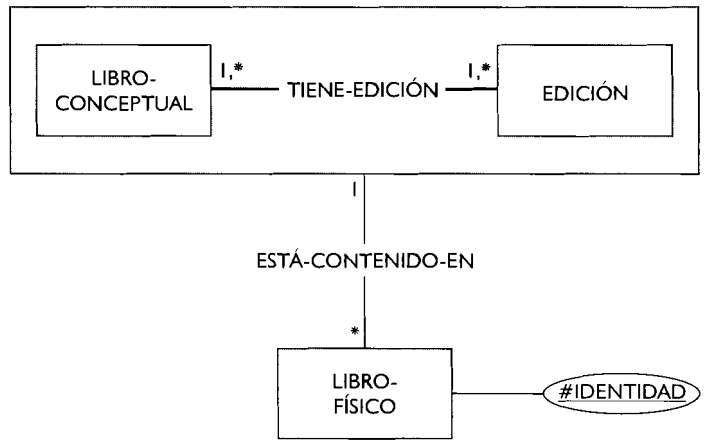
El problema tiene que ver con el conjunto de objetos EDICIÓN-LIBRO-CONCEPTUAL. Dado que cada instancia es una edición de un libro particular, no se pueden comparar ediciones idénticas de libros diferentes. Un problema adicional con esta solución es que requiere que la EDICIÓN-LIBRO-CONCEPTUAL contenga considerablemente más instancias de objetos que las que son realmente necesarias.

La Figura 4.44 da una mejor solución. En este caso, EDICIÓN es un conjunto de objetos independiente. Puesto que un libro conceptual puede tener muchas ediciones, EDICIÓN no puede ser un atributo de LIBRO-CONCEPTUAL. Por tanto, la interrelación entre LIBRO-CONCEPTUAL y EDICIÓN es muchos-muchos. Con este modelo se pueden responder las preguntas sobre ediciones y no hay duplicaciones innecesarias de las ediciones conceptuales. Por ejemplo, Edición Clásica Completa aparece sólo una vez en el conjunto de objetos EDICIÓN, mientras que aparece dos veces incrustado en "Edición Clásica Completa de Oliver Twist" y "Edición Clásica Completa de David Copperfield" en el conjunto de objetos EDICIÓN-LIBRO-CONCEPTUAL de la Figura 4.43. Puesto que podrían haber muchos libros en la Edición Clásica Completa, este nuevo enfoque elimina una gran cantidad de duplicaciones potenciales.



Usando la Figura 4.44 se puede añadir a nuestro modelo la noción de "libros físicos" (Figura 4.45). Una instancia de LIBRO-FÍSICO representa un volumen concreto que se puede marcar con un número de identidad y el cual se puede detectar con un solo patrón de biblioteca a la vez. En este ejemplo se asume que el número de identidad incluye *toda* la información necesaria para identificar de forma única a un libro físico en particular. Por lo tanto, la *clave externa* para cada libro físico es este número, o *número de identificación física*, mediante el cual puede ser rastreado con propósitos de control de inventarios. El número de identidad puede incluir información tal como *número de copia*, que distingue una copia de un determinado libro conceptual dado de otra copia del mismo libro.

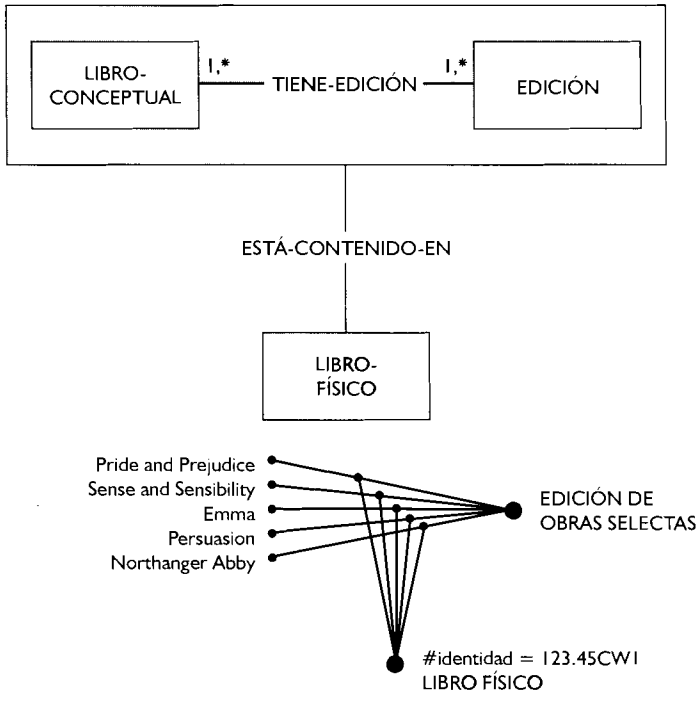
Obsérvese la cardinalidad uno-muchos de la interrelación ESTÁ-CONTENIDO-EN de la Figura 4.45. Tal cantidad afirma que una determinada combinación libro-edición puede estar contenida en muchos libros físicos diferentes. Esto corresponde con nuestra comprensión de la realidad. Pero la cardinalidad también asevera que un libro físico dado puede contener sólo un libro-edición. ¿Es esto preciso?

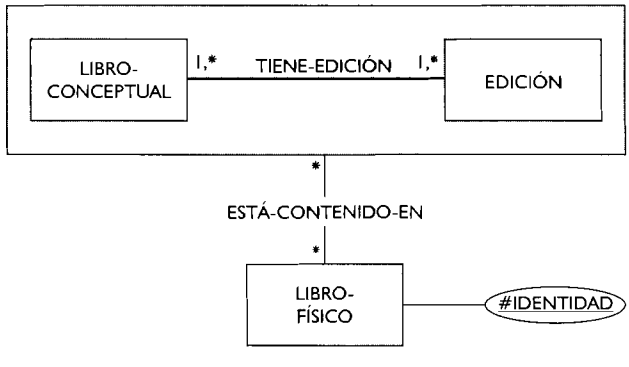


Considere un libro que contiene obras seleccionadas de Jane Austen. Un tal libro contiene diferentes libros conceptuales, aunque se puede decir que todos tienen la misma edición. Todos estos libros conceptuales están contenidos en el mismo libro físico, como se muestra en la Figura 4.46. Puesto que esta situación no es inusual, para ganar en precisión vamos a corregir la cardinalidad de la Figura 4.45 de uno-muchos a muchos-muchos (Figura 4.47). Esto es, un sencillo libro físico puede estar relacionado con múltiples libros conceptuales.

Nuestro modelo de datos puede aún no ir suficientemente lejos. Si los usuarios de la biblioteca necesitan identificar los libros por el idioma en que fueron publicados, se debería tener a IDIOMA como un objeto aparte. El idioma puede ser un atributo de una combinación libro-edición (asumiendo que una edición de un libro puede estar en un solo idioma) o pudiera ser un conjunto de objetos aparte que tiene una interrelación muchos-muchos con edición-libro. Esto es, una edición dada de un libro pudiera contener porciones en Italiano, Francés, Español, Inglés y otros. La Figura 4.48 muestra IDIOMA como un conjunto de objetos relacionado a través de la interrelación ESTÁ-EN-IDIOMA con el agregado de TIENE-EDICIÓN. Un libro físico puede entonces corresponder con un objeto instancia consistente de un libro conceptual, una edición y un idioma, el cual está en el agregado de ESTÁ-EN-IDIOMA.

La distinción entre libros conceptuales y libros físicos es crucial para la solución de este problema. Lo que es aún más importante, esta distinción conceptual-física es útil en la solución de muchos problemas similares en el modelado de datos. Éstos se encuentran en

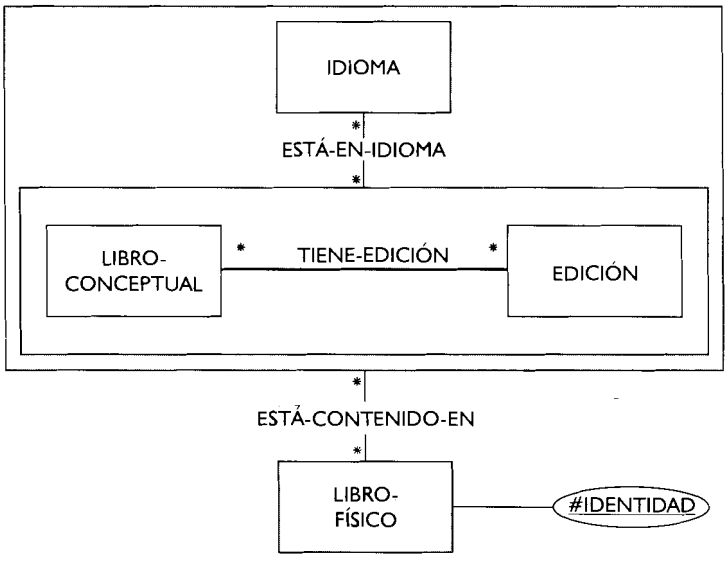




muchos tipos de situaciones de negocios. Cada vez que una palabra se utilice ambiguamente, el problema potencial existe. Sin embargo, como se ha demostrado, la solución es bastante simple. Definiendo conjuntos de objetos por separado, uno para cada uno de los significados del término ambiguo, y definiendo las interrelaciones apropiadas entre estos conjuntos de objetos, se puede construir un modelo de datos que provea toda la información que los usuarios requieren. Algunos ejemplos adicionales ayudarán a aclarar esto.

Fabricación de piezas

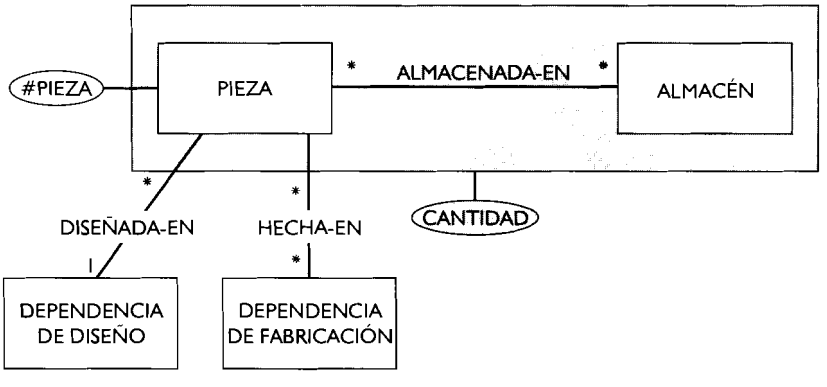
La Empresa Robespierre tiene talleres de diseño, talleres de fabricación y almacenes. Estos talleres de diseño producen y almacenan piezas. Una pieza se diseña sólo en un taller,



pero puede ser elaborada y almacenada por varios talleres. Después de entrevistarse con los dueños, Louis y Marie Blades, y con varios de los oficinistas y gerentes de Robespierre, los analistas de sistemas determinaron que las siguientes son preguntas típicas:

¿Qué piezas fueron diseñadas en qué taller? Si una pieza en particular falla, ¿podemos seguir el rastro hasta el taller que la diseñó y el taller que la confeccionó? ¿Qué cantidad de la pieza A235 hay almacenada en el almacén Lexington?

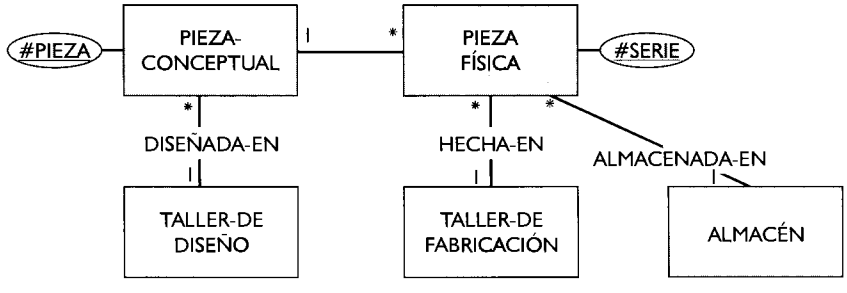
Está claro que la mayoría de estas preguntas tienen que ver con seguir el rastro de piezas específicas que fueron diseñadas en un taller, confeccionadas en otro y luego guardadas en un almacén. La Figura 4.49 es el resultado de un primer intento de un modelo de datos para este problema. Note las cardinalidades. Se puede responder a la primera pregunta —¿Qué piezas fueron diseñadas en qué taller?— porque cada pieza se diseña en



un único taller. Si una pieza falla, se puede usar el número de pieza para identificar ésta y determinar dónde fue diseñada. Pero no podemos decir dónde fue fabricada, puesto que una pieza puede fabricarse en muchos talleres diferentes. La tercera pregunta —¿Qué cantidad de piezas A235 hay en el almacén Lexington?— se puede responder, puesto que el modelo captura la cantidad de piezas almacenadas en cada almacén para cada combinación pieza/almacén. De este modo, el modelo de datos de la Figura 4.49 provee la mayoría de la información que se necesita para responder a las preguntas.

Un modelo de datos mejor, que responde a todas las preguntas, se muestra en la Figura 4.50. Aquí se distingue entre piezas conceptuales y piezas físicas. Una pieza conceptual representa un *tipo* de pieza y tiene un número de pieza, que se le asigna al diseñarla. Esto es precisamente lo que se entiende por PIEZA en la Figura 4.49. Una pieza física es una instancia particular de su correspondiente pieza conceptual. Por lo tanto, la interrelación entre PIEZA CONCEPTUAL y PIEZA FÍSICA es uno-muchos: Una pieza física corresponde a sólo una pieza conceptual, pero una pieza conceptual corresponde a muchas piezas físicas. Una pieza física tiene un número de serie que la identifica. Más aún, ésta fue fabricada en sólo un taller, y en un momento dado está guardada en un solo almacén.

Este modelo de datos responde a todas las preguntas listadas anteriormente. Se ha omitido CANTIDAD como atributo de este modelo de datos. La cantidad de piezas en un almacén puede determinarse buscando la instancia deseada de almacén en el conjunto de objetos ALMACÉN y luego contando el número de piezas físicas relacionadas con este almacén en la interrelación ESTÁ-ALMACENADA-EN. Puesto que la computadora puede contar fácilmente el número de tales instancias es innecesario crear un atributo superfluo CANTIDAD.



Objetos conceptuales para los servicios de consulta Manwaring

A lo largo de un período de varios años, Manwaring ha desarrollado una serie de sistemas de computación para sus clientes. Después de trabajar con muchos clientes diferentes, el equipo de Manwaring ha encontrado que con frecuencia éstos tienen necesidades similares, por lo que el mismo software se puede usar para estas necesidades. Por ejemplo, Statten necesita un sistema de contabilidad por cobrar, de contabilidad por pagar, un sistema de contabilidad de costos y un sistema de nóminas. Crear sistemas generales para la contabilidad por cobrar, por pagar, de costos, para nóminas, para control de inventarios y otros, Manwaring puede satisfacer las necesidades de muchos clientes a un costo reducido. A partir de esta experiencia surge la decisión de crear sistemas básicos en cada una de estas áreas.

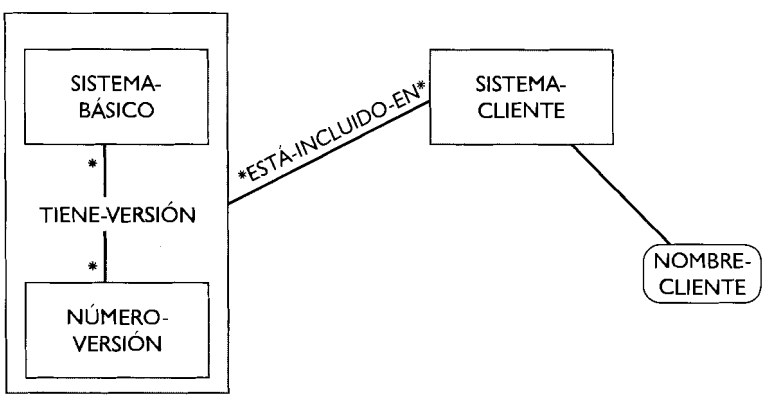
La Figura 4.51(a) muestra un modelo de datos que da la interrelación entre los sistemas básicos y los sistemas de los clientes que usan éstos. Los sistemas básicos tienen *números de versión* para indicar las diferentes versiones del sistema. Por ejemplo, la primera versión de un sistema de contabilidad por pagar pudiera haber tenido el número de versión 1.0. La segunda y la tercera versión pudieran tener los números 1.1 y 2.0, respectiva-

mente. Puesto que cada sistema básico puede tener muchos números de versiones y cada número de versión se puede aplicar a muchos sistemas básicos, la interrelación entre SISTEMA BÁSICO y NÚMERO DE VERSIÓN es la de muchos-muchos.

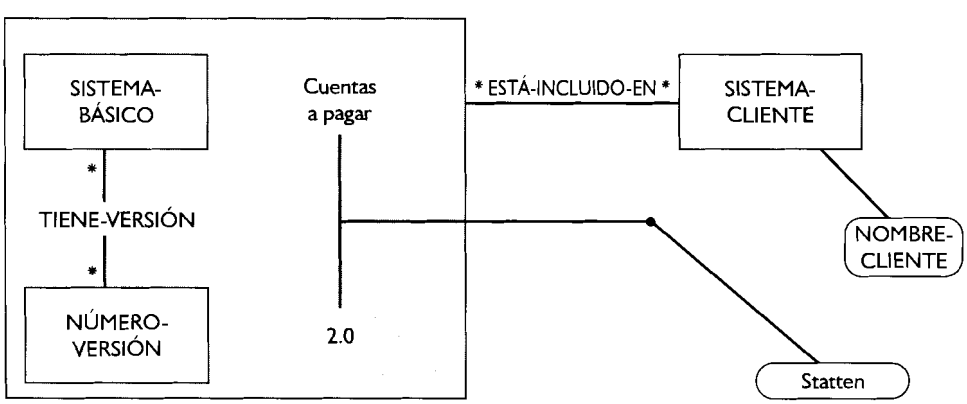
Cada sistema del cliente está relacionado al sistema(s) básico a partir del cual fue constituido. Sin embargo, puesto que el cliente siempre recibirá una *versión* específica del sistema básico, el sistema del cliente está relacionado *tanto* con el sistema básico como con el número de versión. Esto es, la interrelación ESTÁ-INCLUIDA-EN es entre SISTEMA CLIENTE y el *agregado* de SISTEMA BÁSICO y NÚMERO DE VERSIÓN. Esta interrelación ESTÁ-INCLUIDA-EN es muchos-muchos porque un sistema de cliente dado incluirá muchas combinaciones de sistema-básico/número-de-versión, y una combinación dada de sistema-básico/número-de-versión se incluirá en muchos sistemas de cliente diferentes.

La Figura 4.51(b) muestra las instancias para este modelo de datos. El sistema para el cliente Statten se muestra como un punto del rectángulo SISTEMA CLIENTE. Este sistema incluye las cuentas por pagar, versión 2.0, y por ello en el diagrama está conectado al par (cuentas por pagar, 2.0). Si el sistema de Statten incluyese otras versiones de sistemas básicos estarían ilustradas otras instancias como ésta.

Este modelo de datos ilustra además la distinción conceptual-físico. El conjunto de objetos SISTEMA-BÁSICO es un conjunto de objetos conceptuales y el conjunto de objetos SISTEMA CLIENTE es un conjunto de objetos físicos. De hecho, este ejemplo es muy similar al ejemplo anterior de la biblioteca. Si se comparan las Figuras 4.51(a) y 4.47 se puede ver la correspondencia siguiente:



(a) Un modelo de datos para los sistemas instalados



(b) Un modelo de datos con instancias

```
LIBRO CONCEPTUAL ———————————— SISTEMA BÁSICO
EDICIÓN ———————————————— NÚMERO DE VERSIÓN
LIBRO FÍSICO ———————————— SISTEMA CLIENTE
```

El propósito de este ejemplo, así como el de todos aquellos que le han precedido en esta sección, es ilustrar las ambigüedades que acechan en el lenguaje natural que se utiliza para describir los requisitos de los usuarios de las bases de datos. Para estar seguro que los modelos de datos son precisos y completos se deben analizar cuidadosamente las circunstancias de cada aplicación y la clase de información que desean los usuarios del sistema de base de datos.

▼ Integración de vistas: Un ejemplo

Los ejemplos que se han estado usando en los últimos tres capítulos tienden a unificarse en la creación de un único modelo que satisfaga los requisitos de los usuarios con los que se ha estado trabajando. En una gran organización un enfoque tan simple no es posible, y el desarrollo de un proyecto de base de datos puede requerir la creación de varios modelos de datos diferentes creados por los equipos de analistas que trabajan con los usuarios en las diferentes áreas. Estos modelos separados se llaman **vistas**, puesto que cada uno de ellos representa la forma en que un usuario ve a la base de datos. Para crear una única e integrada base de datos, estas diferentes vistas deben integrarse en un único modelo de datos.

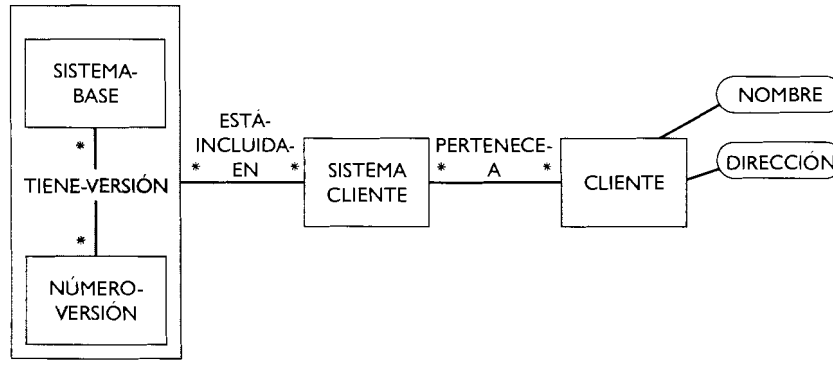
vista. Una definición de una porción restringida de la base de datos.

Como ejemplo, tomar dos de los modelos de datos de los Servicios de Consulta Manwaring y ver cómo éstos se pueden integrar en un único modelo. El enfoque será preservar cada vista en su estado original lo más posible y conectar los conjuntos de objetos en las diferentes vistas mediante la creación de nuevas interrelaciones entre ellos.

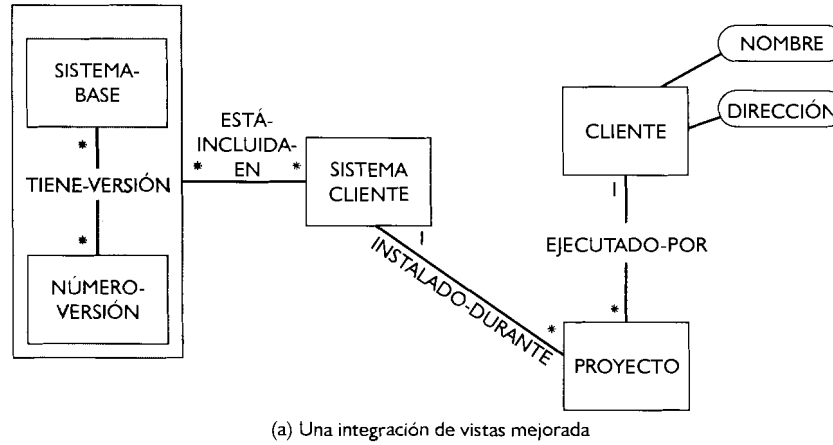
Considere las Figuras 4.41 y 4.51 (a). La Figura 4.41 contiene un objeto CLIENTE con un atributo NOMBRE, y la Figura 4.51(a) contiene un conjunto de objetos SISTEMA CLIENTE con un atributo NOMBRE CLIENTE. Puesto que el atributo NOMBRE CLIENTE de la Figura 4.51(a) y el atributo NOMBRE de la Figura 4.41 representan el mismo atributo, un modelo integrado hará redundante a uno de ellos. Una solución posible sería relacionar SISTEMA CLIENTE a CLIENTE y eliminar NOMBRE CLIENTE como atributo de SISTEMA CLIENTE [como se muestra en la Figura 4.52(a)].

Sin embargo, esta solución falla al considerar otras partes de la Figura 4.41. Por ejemplo, los sistemas cliente se desarrollan durante los proyectos. De esta manera parece más razonable relacionar el conjunto de objetos SISTEMA CLIENTE al conjunto de objetos PROYECTO, como se muestra en la Figura 4.52(b). La interrelación INSTALADO-DURANTE indica que un sistema cliente fue creado durante una serie de proyectos, todos ellos para el mismo cliente. De este modo, podemos recorrer un *sistema cliente* a través de los *proyectos* utilizados para instalarlo y pasar a los clientes para los cuales se llevaron a cabo los proyectos. Esta solución integra dos vistas y nos da el simple y unificado modelo de datos que estamos buscando.

La integración de vistas para una base de datos de una gran organización es un problema complejo que requiere del análisis de los conjuntos de objetos, los atributos y las relaciones de las vistas por parte de los analistas y de los usuarios que estén más familiarizados con ellas. Aquí solamente se ha mostrado un ejemplo sencillo que ilustra algunos de los conceptos básicos involucrados. En una situación concreta de negocios, el proceso de integración de vistas puede trabajar en algunos casos, pero en otros no. Como se ha dicho anteriormente, debido a la complejidad del desarrollo de un proyecto de base de datos algunas organizaciones optan por *no* tener una única base de datos para toda la información que necesitan. Escogiendo este enfoque, estas organizaciones han evitado algunos de los aspectos más difíciles de la integración de vistas. Por supuesto, las bases de datos más pequeñas son representativas de la variedad de vistas de los usuarios y cada una de estas bases de datos requerirá de un proceso de integración de vistas para su diseño exitoso.



(a) Un primer intento de integración de vistas



(a) Una integración de vistas mejorada

En este capítulo se han estudiado los fundamentos del modelado conceptual de datos. Se ha definido el concepto general de un modelo de datos, se ha descrito una metodología de modelado y se ha mostrado cómo esta metodología se puede usar para los conjuntos de consultas de los usuarios y para los informes existentes.

La palabra *modelo* se usa a menudo en tres niveles diferentes: como una metodología para la creación de modelos de segundo nivel, como un esqnema de base de datos que define los tipos de datos que se deben tener en una aplicación en particular, y como una base de datos implemeutada, que contiene hechos específicos estructurados al esquema de segundo-nivel.

Un modelo de datos conceptual también se denomina *semántico*, puesto que captura el significado de las cosas en el mundo real. Los modelos de datos conceptuales consisten de conjuntos de objetos, interrelaciones y sus agregados, atributos, conjuntos de especialización, indicadores de cardinalidad y claves. Los conjuntos de objetos pueden ser lexicográficos, que contienen instancias que se pueden imprimir, o abstractos, que contienen instancias que no se pueden imprimir. Las instancias en conjuntos de objetos abstractos están representadas por claves subrogadas, las cuales son identificadores internos sin ningún significado externo. Las interrelaciones establecen conexiones entre instancias de dos conjuntos de objetos. Los atributos son interrelaciones entre dos conjuntos de objetos que son funcionales en la dirección de uno de los conjuntos de objetos. La especialización de conjuntos de objetos, como subconjuntos de otro conjunto de objetos, nos dan un medio para definir atributos para algunas instancias sin necesidad de definir éstos para otras instancias.

Las cardinalidades de una interrelación indican cuántas veces un conjunto de objetos está relacionado bajo dicha interrelación con un elemento simple en el otro conjunto. Las cardinalidades son uno-uno, uno-muchos y muchos-muchos. Las claves identifican de forma única a los objetos. Las claves subrogadas son identificadores internos. Las claves externas son subconjuntos de atributos léxicos que juntos identifican un elemento en un conjunto de objetos. Un agregado es una interrelación vista como un conjunto de objetos. Usando agregados es posible abordar problemas de modelado más complejos. Los agregados pueden tener atributos y participar en otras interrelaciones. Estas nuevas interrelaciones también pueden ser agregadas y se pueden usar como conjuntos de objetos en otra interrelación.

Los conjuntos de objetos conceptuales representan entidades que son tipos de cosas. Por ejemplo, un libro en un conjunto de objetos conceptuales no es un libro físico en concreto, sino que representa a una entidad conceptual compleja desarrollada por el autor. El libro conceptual puede tener muchas ediciones diferentes y cada edición puede tener una impresión de muchas copias físicas del libro. Las copias físicas son libros físicos, los cuales podrían representarse en un conjunto propio de objetos. Existen muchos otros ejemplos de la distinción entre conceptual y físico. Comprendiendo cómo una simple palabra como libro se puede usar de manera ambigua, el analista está mejor preparado para crear estructuras de modelado para manipular los diferentes significados posibles.

Mediante el análisis de las preguntas que los usuarios quieren responder y de los informes de organización que estos usuarios necesitan, se pueden construir los modelos conceptuales. En el proyecto de desarrollo de una gran base de datos diferentes analistas trabajarán con diferentes grupos de usuarios para crear los diferentes modelos de datos o vistas, las cuales deberán integrarse. Este proceso implica quitar sólo aquellos conjuntos de objetos, interrelaciones y atributos que son redundantes entre las vistas y luego conectar las vistas, definiendo nuevas interrelaciones. Este proceso requiere que los analistas y los usuarios que trabajan en áreas diferentes se comuniquen para entender cómo es que deben integrarse las vistas de manera precisa.

1. Defina con sus propias palabras cada uno de los términos siguientes:
   a. modelo
   b. modelo orientado a objetos
   c. conjunto de objetos
   d. conjunto de objetos léxicos
   e. clave subrogada
   f. generalización
   g. conjunto de objetos agregados
   h. interrelación funcional
   i. cardinalidad
   j. uno-muchos
   k. atributo
   l. interrelación *n*-aria
2. Identifique y escriba seis construcciones utilizadas en el modelado conceptual de datos.
3. Discuta cómo las entrevistas y el análisis de los informes se utilizan con el modelado conceptual de datos en el proceso del diseño conceptual de base de datos.
4. Discuta cómo se analizan una serie de consultas potenciales de los usuarios para determinar las construcciones siguientes en un modelo de datos conceptual:
   a. conjuntos de objetos
   b. atributos
   c. interrelaciones
   d. especializaciones

5. Discuta cómo se analiza un informe para determinar las siguientes construcciones en un modelo de datos conceptual:
   a. conjuntos de objetos
   b. atributos
   c. interrelaciones
   d. especializaciones
6. ¿En qué situaciones se necesitan los agregados en el modelado de los datos? ¿Cuándo son apropiadas las interrelaciones de más alto nivel? Dé ejemplos de situaciones en el campo de los negocios.

Parte A

1. Conecte cada término con su definición:

—*muchos-muchos*	**a.** Valor usado para identificar unívocamente a una instancia objeto.
—*interrelación binaria*	**b.** Interrelación entre tres o más conjuntos de objetos.
—*agregado*	**c.** Atributos léxicos cuyos valores identifican a una única instancia.
—*modelo semántico*	**d.** Conjunto de objetos que consta de instancias que no se pueden imprimir.
—*conjunto de objetos abstractos*	**e.** Un enlace entre instancias de dos conjuntos de objetos.
—*especialización*	**f.** Miembro particular de un conjunto de objetos.
—*interrelación*	**g.** Cardinalidad de interrelación que es mucho en ambas direcciones.
—*uno-a uno*	**h.** Conjunto de objetos que es un subconjunto de otro conjunto de objetos.
—*función (map)*	**i.** Propiedad de tener todos los atributos del conjunto de generalización que especializa.
—*objeto instancia*	**j.** Asociar elementos en un área con elementos de otro área.
—*clave externa*	**k.** Una interrelación vista como un conjunto de objetos.
—*valor nulo*	**l.** Cardinalidad de interrelación que es uno en ambas direcciones.
—*clave*	**m.** Captura el significado de las entidades del mundo real y las interrelaciones.
—*herencia*	**n.** Valor de atributo que no existe para una instancia en específico.
—*interrelación de alto-nivel*	**o.** Interrelación entre dos conjuntos de objetos.

Parte B. Para cada uno de los problemas siguientes crear un modelo de datos conceptual consistente de conjuntos de objetos, interrelaciones, atributos, y otros, que puedan usarse para responder preguntas similares a las preguntas dadas. Indique las cardinalidades.

Asuma que estos modelos son para un entorno universitario:

2. ¿Cuántos miembros de la Facultad han sido asignados al departamento de matemáticas? ¿Cómo se llaman? ¿Quién es el encargado del departamento de música?

   (Nota: "matemáticas" y "música" son sólo ejemplos de departamentos. El modelo debe ser capaz de responder a las mismas preguntas si fuesen, digamos, sociología, ingeniería mecánica o ciencias políticas.)
3. ¿Cuáles son los estudiantes que están mejor en historia? ¿Y en español?
4. ¿Qué miembros de la Facultad están impartiendo cursos de sociología? ¿Qué cursos están impartiendo?

5. ¿Cuántos estudiantes realizan el curso Física 201? ¿Qué sección está llevando Andrea Edens?

6. ¿Cuántos alemanes están registrados formalmente en el programa de honor? Para aquellos que están, ¿quién es su consultor del programa de honor?

Asuma que estos modelos son para un comerciante.

7. ¿Qué productos tienen precios sobre los $200? ¿Cuáles cuestan menos de $150? ¿Cuáles provienen de fabricantes del medio oeste? ¿Cómo se llaman los fabricantes?

8. ¿Qué vendedor ha vendido productos con precio por encima de $200? ¿Cuáles son las fechas de estas ventas? ¿Cuál es el salario base para estos vendedores?

Asuma los modelos siguientes para un banco. Obtenga estos modelos ampliando el modelo de la Figura 4.21.

9. ¿Qué porcentaje de propietarios de cuentas corrientes son empleados del banco?

10. ¿Cuántos cajeros tienen cuentas de ahorro en el banco? ¿Y cuántos gerentes? ¿Cuántos cajeros no tienen?

11. ¿Cuántos gerentes que tienen cuentas de ahorro en el banco dirigen a empleados que también tienen cuentas de ahorro en el banco?

Parte C. Señale qué preguntas no se pueden responder por el modelo de datos de la Figura 4.21 y explique por qué no.

12. ¿Cuál es el saldo medio de las cuentas de ahorro de las fábricas que tienen más de 500 empleados?

13. ¿Cuántas mujeres abrieron cuentas corrientes el 5 de diciembre de 1988? (Figura 4.22).

14. ¿Cuántos espacios están disponibles para nuevos árboles en el huerto Heber City?

15. ¿Cuál es el promedio de vida de los manzanos Jonathan en el huerto Pleasantville? (recuerde que Jonathan es una variedad y manzana es una especie).

16. ¿Cuántos árboles de melocotón del huerto Springtown tienen más de dos variedades?

Parte D.

17. Obtenga un modelo conceptual analizando el informe que se muestra en la Figura 4.1E.

SERVICIOS DE CONSULTA MANWARING

INFORME DEL PERFIL DE LOS CONSULTORES

Nombre	NSS	Fecha de contratación	Código-Especialidad	Especialidad
Farasopuolas	539-88-4242	22/11/84	A	Entrenamiento al usuario
			B	Entrada de datos
			D	Conversión de archivos
Harris	560-43-1111	8/11/86	C	Programación
			D	Conversión de archivos
			F	Diseño de sistemas
Rodríguez	524-33-8119	3/7/85	A	Entrenamiento al usuario
			C	Programación
			E	Análisis de sistemas
			F	Diseño de sistemas

Parte E. Agregación.

18. Para cada una de las sentencias siguientes dibuje un modelo de datos que muestre la interrelación entre los conjuntos de objetos, una agregación de la interrelación y atributos del agregado.

a. Los estudiantes reciben clases y obtienen calificaciones en las clases.
b. Las secciones de cada curso se ofrecen en horarios específicos y en aulas y edificios.
c. Cada período escolar puede estar representado por una sesión (otoño, invierno, primavera y verano) y un año, y comienza y termina en fechas específicas.
d. Cada día los empleados trabajan un cierto número de horas.
e. La gente se suscribe a periódicos y las suscripciones tienen fechas de inicio y de terminación.
f. Los pilotos tienen un cierto número de horas de entrenamiento para cada tipo de avión.

Para cada uno de los problemas siguientes crear un modelo de datos conceptual, consistente en conjuntos de objetos, interrelaciones, atributos, y así sucesivamente, que pueda utilizarse para responder a preguntas similares a las preguntas dadas. Use agregados e interrelaciones de alto nivel según necesite. Indique las cardinalidades.

19. ¿Cuántos estudiantes están cursando Física 201? ¿Cuál es la sección que Andrea Edens está tomando? ¿Cuántas veces ha cursado Jim Hardy Contabilidad 201, cuándo, quiénes fueron sus profesores, qué calificaciones obtuvo?

Dustin Tomes, profesor de historia, desea utilizar la base de datos para elaborar preguntas sobre la historia europea. Crear un modelo de datos separado para cada uno de los problemas siguientes.

20. ¿Cuántos reyes de Prusia tenían nombre Frederick? ¿Cuándo vivieron y cuándo reinaron? ¿Gobernaron en algún otro país durante sus vidas? ¿Hubo países de Europa gobernados por mujeres durante el siglo XVII? ¿Cuáles?

21. ¿Fue el abuelo de María Antonieta gobernante de algún país? ¿De cuál y cuándo? ¿Quién fue su madre? ¿Qué gobernantes de países diferentes se casaron entre sí? ¿Cuántos hijos de Enrique VIII fueron monarcas de Inglaterra? ¿Quiénes fueron sus madres?

Brick Wall Communications tiene un grupo de estaciones de televisión. Estas estaciones televisan seriales, anuncios comerciales y eventos deportivos en directo. Crear un modelo separado para cada problema a continuación.

22. ¿Qué cadenas transmiten series de *Batman*? ¿Ha retransmitido el año pasado la Brick Wall alguno de los episodios del período de 1988 del *Show de Cosby*? ¿Pasaron el quinto episodio? ¿Cuándo y qué cadenas?

23. ¿Cuántos juegos de baseball transmitieron el año pasado? ¿En qué fechas transmitieron partidos entre los Dodgers y los Mets? ¿Qué equipos fueron transmitidos más a menudo? ¿Qué hay con los juegos de fútbol? ¿Y con los de baloncesto? ¿Y los partidos de tenis? ¿Torneos de golf? ¿Y otros deportes? ¿Ha jugado tenis Steffi Graf por algna de las cadenas de la Brick Wall? ¿Qué cadenas y cuándo?

24. ¿Qué anuncios se han mostrado más de tres veces en una misma hora en una misma cadena? ¿Cuándo ocurrió esto? ¿A qué hora, qué día y en qué cadena? ¿Cuánto cobra la Brick Wall por la transmisión de cada uno de estos anuncios?

Frank Howe, director representante de la firma de abogados Dewey, Kleenem, Outt y Howe, ha decidido que la firma se beneficiaría sustancialmente teniendo una

base de datos que sea directamente aplicable a asuntos legales. Para cada uno de los problemas siguientes crear nn modelo de datos separado.

25. ¿Qué casos tienen opiniones vertidas sobre la Sección 411.3c del código federal? ¿Qué cortes fueron involucradas? ¿Cuándo se atendieron estas opiniones? ¿Qué secciones del código federal fueron interpretadas como el caso de *Black* vs. *Williams?*

26. ¿Qué firmas han representado en la corte a General Continental durante los últimos diez años? ¿Cuáles fueron los casos; en cuáles el veredicto fue favorable; y cuál fue el importe de las recompensas? ¿Cuáles fueron las firmas opositoras? ¿Qué otras grandes compañías fueron representadas por estas firmas en la misma época?

Parte F.

27. Como parte de un proyecto para Acme Insurance Company, uno de los analistas de Manwaring creó un informe para medir la productividad del personal de esta compañía en la entrada de datos. Este informe da para cada día del mes el número de transacciones de cada tipo que han sido introducidas por cada empleado. Deduzca el modelo de datos conceptual que podría usarse como base para el informe que se muestra en la Figura 4.2E.

Parte G. Use conjuntos de objetos conceptnales y físicos para crear modelos de datos para los problemas siguientes:

28. Una aerolínea desea contestar a preguntas como las siguientes sobre sus aeronaves: ¿Cuál es la capacidad de asientos del Boeing 727? ¿Cuántos motores tiene éste? ¿Cuál es la media de edad del parque de 727s? ¿Quién es el mecánico jefe responsable del servicio del avión número 1388? ¿Qué compañía fabricó dicho avión?

ACME INSURANCE COMPANY

INFORME DE PRODUCTIVIDAD MENSUAL
Para el mes que termina en marzo 31

Empleado núm.	Nombre	Fecha	Tipo de transacción	Cantidad terminada
3855	J. Perkins	1/3	Nueva póliza	15
			Pago	75
			Reclamación	22
		2/3	Nueva póliza	18
			Cambio de póliza	53
			Reclamación	25
		• • •		
3921	S. Stallone	1/3	Pago	45
			Cambio de póliza	83
			Reclamación	10
		2/3	Nueva póliza	8
			Pago	63
			Cambio de póliza	35
		• • •		

29. La División de Servicios Administrativos de una gran ciudad desea tener la información de su equipamiento computacional. También desea responder a preguntas sobre los modelos de computadoras que tiene. Cree un modelo de datos para las preguntas siguientes:

¿Cuál es la memoria máxima que un PC IBM puede tener? ¿Y los PC-XT y los PC-AT? ¿Cuál es la memoria máxima posible para un Macintosh II? ¿Cuáles de nuestros empleados tienen PCs IBM en sus oficinas? ¿Quién tiene el computador con número de serie 4538842? ¿Cuánta memoria principal tiene?

Parte H. Integración de vistas.

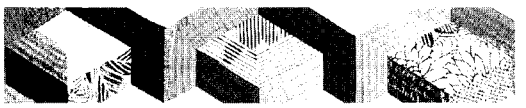
30. Crear un modelo de datos integrado a partir de las vistas (o modelos de datos) que fueron creadas en:

a. Problemas 2, 3, 4, 5 y 6.
b. Problemas 7 y 8.
c. Problemas 20 y 21.
d. Problemas 22, 23 y 24.
e. Problemas 25 y 26.

1. Crear un modelo de datos conceptual para una parte de una organización. El modelo debe tener al menos cinco conjuntos de objetos con sus atributos e interrelaciones. Tratar de determinar las condiciones bajo las que se requiere de conjuntos de especialización.

2. Leer los artículos de cada uno de los siguientes modelos de datos y determinar cómo estos modelos tratan la agregación. Determinar cuándo estos artículos dan información suficiente para responder a estas preguntas: ¿Puede un agregado tener atributos en el modelo? ¿Puede éste participar en interrelaciones? ¿Pueden agregarse también estas interrelaciones?

a. El modelo de entidad-interrelación (Chen 1976)
b. El modelo de datos semántico (Hammer y McLeod, 1981)
c. El modelo de datos funcional (Shipman, 1981).



CAPÍTULO

5

EL MODELO DE DATOS RELACIONAL

- El modelo de datos relacional y el desarrollo de sistemas
- El modelo de datos relacional: Conceptos fundamentales
  - Relaciones
  - Valores nulos
  - Claves
  - Claves externas (ajenas)
  - Restricciones de integridad
    - Integridad de la entidad
    - Integridad referencial
- El proceso de normalización
  - Primera forma normal
  - Segunda forma normal
  - Tercera forma normal
  - Otras formas normales
    - Quinta forma normal
    - Forma normal dominio/Clave
- Transformar el modelo conceptual en modelo relacional
  - Transformar conjuntos de objetos y atributos
  - Transformar modelos sin claves externas
  - Transformar la especialización y la generalización de los conjuntos de objetos
  - Transformar interrelaciones
    - Interrelaciones Uno-Uno
    - Interrelaciones Uno-Muchos
    - Interrelaciones Muchos-Muchos
  - Transformar conjuntos de objetos agregados
  - Transformar relaciones recursivas
  - Ejemplos de transformaciones: Servicios de Consultoría Manwaring
- Comparación del modelado de datos conceptual y relacional
- Resumen
- Preguntas de repaso
- Problemas y ejercicios
- Proyectos y cuestiones profesionales

Marcus Brown, propietario de la compañía de construcción Premier, está discutiendo la implementación de bases de datos con Tony Melton, gerente de sistemas de información de la compañía. El diseño conceptual para el sistema de bases de datos se ha terminado y el grupo de proyectos está listo para adentrarse en el diseño de implementación para un sistema de gestión de bases de datos relacionales.

"Eso no está claro para mí, Tony. ¿Por qué fuimos a través del diseño conceptual como paso para producir un modelo de bases de datos conceptual si conocíamos desde el principio que tendríamos que convertirlo a un modelo relacional de todas formas?"

"El diseño conceptual es esencial para la creación de un esquema de bases de datos. eso suena lógico. Marcus, pero en este momento hay pocos sistemas que realmente pueden ejecutar una base de datos de modelo conceptual. Ahora que hemos creado un buen modelo lógico para nuestro negocio, necesitamos implementarlo sobre un sistema adaptado a nuestras necesidades de aplicación. Hemos elegido un sistema de gestión de bases de datos relacionales porque para nuestros propósitos es lo más avanzado disponible."

"Eso está bien, pero yo entiendo que es necesario normalizar una base de datos relacional antes de que esté lista para la implementación. No estoy seguro qué significa eso, pero ¿no te requerirá hacer un diseño de bases de datos adicional?"

"El proceso usado para convertir un modelo de datos conceptual en un modelo relacional es uno de los aspectos más poderosos del modelado de datos conceptual. Siguiendo un proceso de conversión mecánico y lineal, crearemos un diseño de implementación relacional que está completamente normalizado. En esencia tendremos lo mejor de ambos mundos."

El centro de atención de este capítulo es el modelo de datos relacional y su uso como un modelo de diseño de implementación de bases de datos. Se define la construcción del modelo, se discutirá el proceso de normalización y se mostrará cómo cualquier modelo conceptnal puede convertirse fácilmente en un modelo relacional equivalente. Después de la lectura de este capítulo debe ser capaz de:

- Explicar los conceptos fundamentales del modelo relacional, incluyendo relaciones, atributos, dominios, claves, claves foráneas, integridad de la entidad e integridad referencial.
- Demostrar cómo las relaciones pueden ser normalizadas. El proceso de normalización requiere un primer entendimiento a través de la cuarta forma normal, dependencias funcionales y dependencias multievaluadas.
- Transformar un modelo de datos conceptual en un modelo de datos relacional en la cuarta forma normal.

▼ El modelo de datos relacional y el desarrollo de sistemas

En 1970, la forma en que las personas veían las bases de datos cambió totalmente cuando E. F. Codd introdujo el modelo de datos relacional (Codd 1970). En ese tiempo, el enfoque existente para la estructura de las bases de datos usaba punteros físicos o direcciones a disco para relacionar registros en diferentes archivos. Suponga, por ejemplo, que se necesita relacionar un registro *A* con un registro *B* en uno de estos primeros sistemas. Para hacer esto, se adicionaría al registro A un campo que contenga la dirección en disco del registro *B*. Este campo añadido, o puntero físico, siempre señalaría desde el registro *A* al registro *B*. Codd demostró que estas bases de datos limitaban significativamente los tipos de manipulación de datos que pueden ser hechos fácilmente por un usuario. Además, estas bases de datos eran fácilmente vulnerables a los cambios del entorno físico. Cada vez que

se añadían los controladores de un nuevo disco a la configuración del sistema de cálculo y los datos eran movidos de una localización física a otra, se requería una conversión extensa de los archivos de datos. Si los campos fueron añadidos a un formato de registro en un archivo, todos los registros existentes en el archivo tendrían una nueva localización física, requiriendo una conversión adicional de los datos. Así, los usuarios y el software fueron restringidos por consideraciones físicas por estar usando datos en una gran variedad de formas que la estructura lógica habría permitido.

El modelo relacional, basado en las relaciones lógicas entre los datos, superó estos problemas. Esto permitió estar totalmente en desacuerdo, sin saberlo exactamente, con la estructura física de dato. Además, Codd propuso dos lenguajes de manipulación de datos basados en la lógica, los cuales prometieron más poder en el acceso y en el procesamiento de los datos. Estos lenguajes, el álgebra relacional y el cálculo relacional, se analizan en el capítulo 6. Hoy estos lenguajes proporcionan las bases para los lenguajes de bases de datos de relaciones comerciales usados en los más populares sistemas de gestión de bases de datos comerciales(SGBD). Se describirán estos SGBD en más detalle en la parte III.

En este capítulo se repasarán dos introducciones al diseño de bases de datos relacional. El primer enfoque es más tradicional. En este enfoque el diseño conceptual no incluye el modelo de datos conceptual, pero procede directamente a la creación de un esquema de bases de datos relacional consistente en definición de tablas relacionales. El diseño se completa entoces mediante la normalización de estas definiciones de tablas de acuerdo a un proceso bien definido.

El segundo asume la creación de un modelo de datos conceptual durante el diseño conceptual. Este modelo es entonces convertido mecánicamente a un modelo relacional. El proceso de conversión garantizará automáticamente la normalización de los resultados del modelo conceptual.

El primer enfoque fue usado tradicionalmente antes que el modelo conceptual se convirtiera en conocido y establecido. Hoy es todavía usado en situaciones que requieren un esquema de bases de datos relativamente simple. En tales casos, el analista debe encontrarlo más fácil que crear y normalizar definiciones de tablas relacionales directamente de la información del usuario. El segundo enfoque, usando modelos conceptuales, es válido en el diseño de grandes y complejos esquemas de bases de datos necesarios para los sistemas de bases de datos corporativas.

Después de introducir los conceptos del modelo relacional, discutiremos el proceso de normalización y el proceso de conversión de un modelo conceptual.

▼ El modelo de datos relacional: Conceptos fundamentales

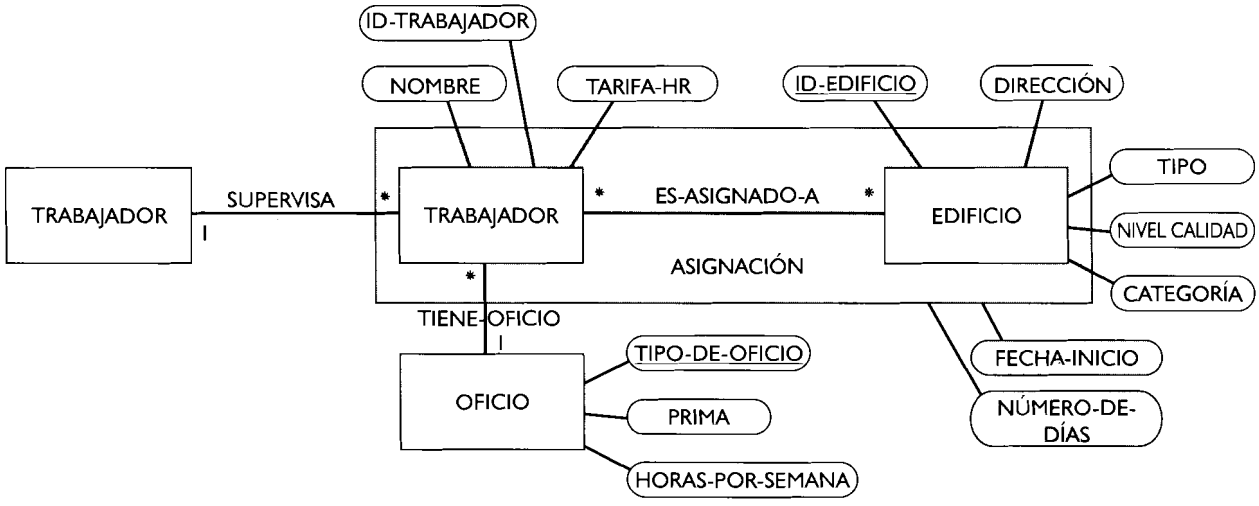
Relaciones

modelo de datos relacional. Un modelo de datos donde los datos se representan en forma de tabla.

El **modelo de datos relacional** organiza y representa los datos en forma de tablas o relaciones. **Relación** es un término que viene de la matemática y representa una simple tabla de dos dimensiones, consistente en filas y columnas de datos. Ejemplos de relaciones se encuentran en el desarrollo de bases de datos para la compañía de construcción Premier, una compañía mencionada anteriormente.

relación. Una tabla de dos dimensiones que contiene filas y columnas de datos.

La Figura 5.1 muestra un modelo de datos conceptual revisado que proporciona los fundamentos para la base de datos relacional de la compañía Premier. Observe que este modelo de datos contiene tres conjuntos de objetos. TRABAJADOR, EDIFICIO y OFICIO, y un conjunto de objetos agregado, ASIGNACIÓN. Aunque el conjunto de objetos TRABAJADOR aparece dos veces en el modelo, es el mismo conjunto de objetos ambas veces. Así, ambas copias de TRABAJADOR tienen los mismos atributos y participan en las mismas relaciones. El conjunto de objetos ASIGNACIÓN es una agregación de la relación ES_ASIGNADO_A entre TRABAJADOR y EDIFICIO. Eso es, que cada asignación con-



siste en un par —un trabajador y un edificio— y significa que el trabajador se asigna a trabajar en el edificio. Cada una de estas asignaciones tiene dos atributos: FECHA_INICIO, la fecha en la que el trabajador es asignado a comenzar a trabajar en el edificio, y NÚMERO_DE_DÍAS, que significa el número de días que el trabajador requiere para completar el trabajo en el edificio. Así, el conjunto de objetos agregado ASIGNACIÓN tiene dos atributos, FECHA_INICIO y NÚMERO_DE_DÍAS.

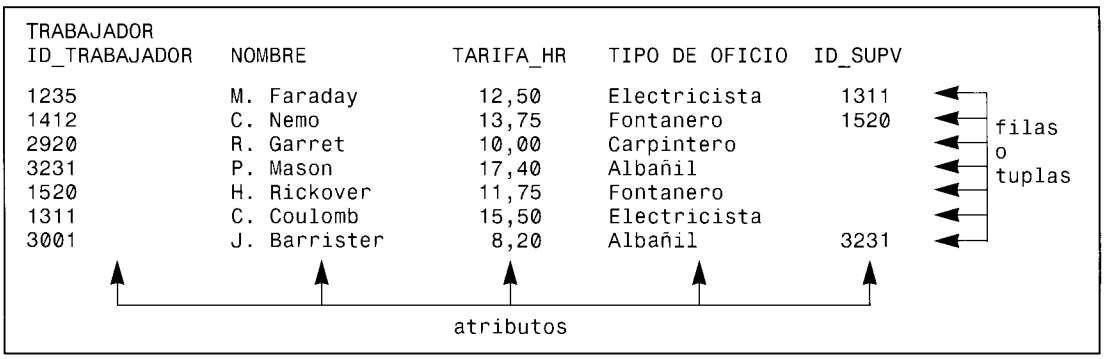
Además de la relación ES_ASIGNADO_A, el modelo contiene la relación TIENE_OFICIO y la relación SUPERVISA. La relación TIENE_OFICIO asocia un oficio, tales como fontanero o albañil, con cada trabajador. El oficio tiene una descripción, una prima y el número de horas por semana que un trabajador de ese tipo de oficio debe trabajar antes que sea efectiva la prima. Puesto que los trabajadores son supervisados por otros trabajadores, la relación SUPERVISA asigna a cada trabajador un supervisor del conjunto de objetos TRABAJADOR. Una relación como SUPERVISA, la cual relaciona objetos del mismo conjunto, es **recursiva**.

interrelación recursiva. Una interrelación que relaciona un objeto consigo mismo.

Usando un proceso que se discutirá posteriormente en este capítulo, este modelo de datos puede ser convertido en un modelo de datos relacional. La Figura 5.2 muestra una relación con un ejemplo de los valores de los datos, la cual representa el conjunto de objetos TRABAJADOR, sus atributos y dos de sus relaciones. Cada columna en la relación es un **atributo** de la relación. El nombre de la columna se llama *nombre del atributo*. Se usan los términos *atributo* y *nombre del atributo* más que los términos *columna* y *nombre de la columna* por ser consistentes con los convenios de bases de datos relacionales. Los nombres de los atributos de TRABAJADOR son ID_TRABAJADOR, NOMBRE, TARIFA_HR, TIPO_DE_OFICIO e ID_SUPV. Estos atributos se corresponden con los atributos y relaciones en el modelo de datos conceptual, tal como se muestra a continuación:

atributo de la relación. Una columna en una relación.

Modelo conceptual	*Atributo de la relación*
ID_TRABAJADOR (atributo)	ID_TRABAJADOR
NOMBRE (atributo)	NOMBRE
TARIFA_HR (atributo)	TARIFA_HR
TIENE_OFICIO (relación)	TIPO_DE_OFICIO
SUPERVISA (relación)	ID_SUPV



En la Figura 5.3 se muestra una base de datos relacional, equivalente al modelo de datos conceptual para la compañía Premier, junto con ejemplos de valores. Como se indicó, se explicará más adelante, en este capítulo, el proceso de conversión de un modelo de datos conceptual en un modelo relacional.

grado de la relación. El número de atributos en una relación.

El número de atributos en una relación se llama **grado de la relación**. El grado de TRABAJADOR es cinco. De este modo, el usuario no tiene que recordar el orden de los atributos en una relación, se asume que el orden en que se relacionan los atributos no es significativo. Por eso se considera que dos atributos en una relación no pueden tener el mismo nombre.

tupla. Una fila en una relación.

Las filas de una relación también se llaman **tuplas**. Se asume que no hay un orden preestablecido de las filas o tuplas de la relación y que dos tuplas no tienen idénticos conjuntos de valores.

Una anotación común que se usa para representar relaciones, tales como la relación de la Figura 5.2, es

```
TRABAJADOR (ID_TRABAJADOR, NOMBRE, TARIFA_HR, TIPO_DE_OFICIO, ID_SUPV)
```

Es decir, el nombre de la relación va seguido de los nombres de los atributos entre paréntesis.

dominio del atributo. El conjunto de valores que puede tomar un atributo.

El conjunto de todos los posibles valores que puede tener un atributo es su **dominio**. Dos dominios son idénticos sólo si tienen el mismo significado. Así, NOMBRE y TIPO_DE_OFICIO poseen diferentes dominios, aunque ambos dominios consisten en cadenas de caracteres. No es necesario que dos atributos con el mismo dominio tengan el mismo nombre. Por ejemplo, ID_SUPV tiene el mismo dominio que ID_TRABAJADOR. En ambos casos, el dominio consiste en los números de identificación del trabajador.

Valores nulos

Suponga un atributo que no sea aplicable en un caso específico. Por ejemplo, algunos empleados en la relación TRABAJADOR no tienen supervisores. Consecuentemente, no existen valores para el atributo ID_SUPV para estos empleados. Además, cuando se entran los datos para una fila de la relación, es posible no conocer los valores de uno o más de los atributos para esa fila. En ambos casos, no se entra nada y esa fila es guardada en la base de datos con **valores nulos** para esos atributos. Un valor nulo no es un espacio en blanco o cero, es simplemente un valor desconocido o inaplicable que puede ser reemplazado más tarde.

valor nulo. El valor dado a un atributo en una tupla si el atributo es inaplicable o su valor es desconocido.

Claves

Está claro que las filas de TRABAJADOR contienen información sobre los empleados individuales. En efecto, se espera que cada empleado se representará por una y sólo una fila

TRABAJADOR

ID_TRABAJADOR	NOMBRE	TARIFA_HR	TIPO DE OFICIO	ID_SUPV
1235	M. Faraday	12,50	Electricista	1311
1412	C. Nemo	13,75	Fontanero	1520
2920	R. Garret	10,00	Carpintero	
3231	P. Mason	17,40	Albañil	
1520	H. Rickover	11,75	Fontanero	
1311	C. Coulomb	15,50	Electricista	
3001	J. Barrister	8,20	Albañil	3231

ASIGNACIÓN

ID_TRABAJADOR	*ID_EDIFICIO*	FECHA_INICIO	NÚM_DIAS
1235	312	10/10	5
1412	312	01/10	10
1235	515	17/10	22
1412	460	08/12	18
1412	435	15/10	15
1412	515	11/05	8
1311	435	08/10	12

EDIFICIO

ID_EDIFICIO	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORÍA
312	123 Elm	Oficina	2	2
435	456 Maple	Comercio	1	1
515	789 Oak	Residencia	3	1
210	1011 Birch	Oficina	3	1
111	1213 Aspen	Oficina	4	1
460	1415 Beech	Almacén	3	3

OFICIO

TIPO_DE_OFICIO	PRIMA	HORAS_POR_SEM
Fontanero	3,00	35
Electricista	3,50	37
Carpintero	2,00	40
Albañil	5,00	35

en TRABAJADOR. Así, si algún atributo identifica unívocamente un empleado, se debe esperar que ese mismo atributo identifique unívocamente la fila para ese empleado en TRABAJADOR. Se asume que el atributo ID_TRABAJADOR identifica unívocamente a un empleado. Entonces, el valor del atributo ID_TRABAJADOR identifica unívocamente a una fila en TRABAJADOR y decimos que ID_TRABAJADOR es una clave en la relación TRABAJADOR.

superclave. Un conjunto de atributos que identifica unívocamente cada fila en una relación.

A cualquier conjunto de atributos que identifique unívocamente a cada tupla en la relación se le llama **superclave**. Una **clave** de una relación es un conjunto *minimal* de tales atributos, es decir, una clave es una superclave minimal. Por *minimal* se entiende que ningún subconjunto del conjunto de atributos claves identificará unívocamente tuplas en una relación. Una clave puede ser también descrita como un conjunto minimal de atributos que determina unívocamente, o **determina funcionalmente**, cada valor del atributo en una tupla.

clave. El conjunto mínimo de atributos que identifica unívocamente cada fila en una relación.

determinante funcionalmente. Determina unívocamente un valor.

En la discusión del modelo conceptual en el capítulo 4 nos referimos a *claves subrogadas*, las cuales son identificadores internos del objeto que no tienen significado fuera del sistema de cálculo, y *claves externas*, las cuales son atributos léxicos que tienen significado fuera del sistema. Las *claves* en el modelo relacional son claves externas en el sentido explicado en el capítulo 4. Es decir, son atributos significativamente externos y sus valores los asignan los usuarios.

Para ilustrar los conceptos de clave y superclave del modelo relacional, consideraremos la base de datos de la Figura 5.3. En la relación TRABAJADOR, los valores del conjunto de atributos

```
{ID_TRABAJADOR, NOMBRE}
```

identifican unívocamente a cualquier tupla de la relación. Por tanto, este conjunto es una superclave para TRABAJADOR. Sin embargo, este conjunto de atributos no es minimal y por tanto no es una clave. En este ejemplo, ID_TRABAJADOR por sí solo es una clave, puesto que cualquier fila en la relación está identificada unívocamente por ID_TRABAJADOR.

En la relación ASIGNACIÓN, la clave está compuesta de los atributos ID_TRABAJADOR e ID_EDIFICIO. Ni ID_TRABAJADOR ni ID_EDIFICIO por separados identifican unívocamente cada fila, pero los dos atributos unidos nos dan la identificación única requerida para una clave. Una clave compuesta de más de un atributo se llama **clave compuesta**.

clave compuesta. Una clave compuesta de más de un atributo.

En una relación dada, puede que más de un conjunto de atributos puedan ser elegidos como clave. Estos atributos se llaman **claves candidatas**. Es posible, por ejemplo, que NOMBRE sea una clave candidata de la relación TRABAJADOR. Esto sería posible si se asumiese que el nombre *siempre* fuera único. Si no se puede hacer esa suposición, entonces NOMBRE no es una clave candidata. Cuando una de las claves candidatas se selecciona como la clave de la relación, puede ser llamada **clave primaria**. La clave candidata que sea más fácil de usar diariamente en la entrada de los datos es normalmente seleccionada como la clave primaria. Generalmente se usa el término *clave* para nombrar a la clave primaria.

clave candidata. Cualquier conjunto de atributos que puede ser elegido como una clave de una relación.

Después de introducir el concepto de clave, ampliaremos la notación usada para identificar una relación, subrayando los atributos de la clave. La relación de la Figura 5.3 se diseña como sigue:

clave primaria. La clave candidata elegida como la clave de la relación.

```
TRABAJADOR (ID_TRABAJADOR, NOMBRE, TARIFA_HR, TIPO_DE_OFICIO, ID_SUPV)
ASIGNACIÓN (ID_TRABAJADOR, ID_EDIFICIO, FECHA_INICIO, NÚM_DÍAS)
EDIFICIO (ID_EDIFICIO, DIR_EDIFICIO, TIPO, NIVEL_CALIDAD, CATEGORÍA)
OFICIO (TIPO_DE_OFICIO, PRIMA, HORAS_POR_SEM)
```

Puesto que ASIGNACIÓN tiene dos atributos claves, ambos están subrayados. Eso significa que ID_TRABAJADOR e ID_EDIFICIO forman una clave para ASIGNACIÓN. Eso no significa que cada uno de ellos por sí solo sea una clave.

Claves externas (ajenas, *foreign*)

El esquema de base de datos dado anteriormente tiene instancias en diferentes relaciones que usan el mismo nombre en un atributo. Ejemplos de esto son TIPO_DE_OFICIO en las relaciones TRABAJADOR y OFICIO, e ID_EDIFICIO en las relaciones ASIGNACIÓN y EDIFICIO. Ambos atributos brindan ejemplos del concepto de una *clave externa*.

clave externa (*ajena*). Un conjunto de atributos en una relación que constituyen una clave en alguna otra (o posiblemente la misma) relación; usada para indicar enlaces lógicos entre relaciones.

Una **clave externa (ajena)** es un conjunto de atributos en una relación que es una clave en otra (o posiblemente la misma) relación. TIPO_DE_OFICIO en la relación TRABAJADOR e ID_EDIFICIO en la relación ASIGNACIÓN son ejemplos de claves externas, puesto que TIPO_DE_OFICIO es la clave de la relación OFICIO e ID_EDIFICIO es la clave de la relación EDIFICIO. Las claves externas son los enlaces esenciales entre las relaciones. Se usan para vincular datos en una relación con datos en otra relación. Así, TIPO_DE_OFICIO enlaza la relación TRABAJADOR con la relación OFICIO y los atributos ID_TRABAJADOR e ID_EDIFICIO en la relación ASIGNACIÓN muestran el enlace entre TRABAJADOR y EDIFICIO.

Los atributos que son claves externas no necesitan tener los mismos nombres que los atributos de la clave con la cual ellos se corresponden. Por ejemplo, ID_TRABAJADOR e

ID_SUPV en la relación TRABAJADOR tienen diferentes nombres, pero ambos toman sus valores del dominio de los números de identificación de los trabajadores. Así, ID_SUPV es una clave foránea en la relación TRABAJADOR que referencia la clave de su propia relación. Para cualquier trabajador, el atributo ID_SUPV indica el supervisor de ese trabajador, que es otro trabajador. Por lo tanto, ID_SUPV debe contener un valor que es una clave en alguna otra tupla de la relación TRABAJADOR. Por ejemplo, en la Figura 5.3, el supervisor del trabajador 1235 es el trabajador 1311. En otras palabras, el supervisor de M. Faraday es C. Coulomb. ID_SUPV es un ejemplo de una clave **externa recursiva**, una clave externa que referencia su propia relación.

clave externa recursiva. Una clave que referencia su propia relación.

Por la importancia vital de la información de la clave externa en la definición de un esquema de base de datos relacional, se revisará el esquema definido anteriormente para mostrar las definiciones de clave externa:

```
TRABAJADOR (ID_TRABAJADOR, NOMBRE, TARIFA_HR, TIPO_DE_OFICIO, ID_SUPV)
Claves Foráneas: TIPO_DE_OFICIO REFERENCIA A OFICIO
                 ID_SUPV REFERENCIA A TRABAJADOR
ASIGNACIÓN (ID_TRABAJADOR, ID_EDIFICIO, FECHA_INICIO, NÚM_DÍAS)
Claves Foráneas: ID_TRABAJADOR REFERENCIA A TRABAJADOR
                 ID_EDIFICIO REFERENCIA A EDIFICIO
EDIFICIO (ID_EDIFICIO, DIR_EDIFICIO, TIPO, NIVEL_CALIDAD, CATEGORÍA)
OFICIO (TIPO_DE_OFICIO, PRIMA, HORAS_POR SEM)
```

Fíjese que las claves foráneas de una relación son definidas inmediatamente después de la definición del nombre de la relación, de los atributos y de las claves. El enunciado

```
TIPO_DE_OFICIO REFERENCIA A OFICIO
```

es una definición de clave externa para la relación TRABAJADOR e indica que el atributo TIPO_DE_OFICIO en TRABAJADOR es una clave externa que referencia el atributo clave (el cual puede también ser llamado TIPO_DE_OFICIO) en la relación OFICIO.

Un listado como ese, que muestre los nombres de las relaciones seguidos por los nombres de sus atributos con los atributos claves subrayados y con las claves foráneas designadas, se llama un **esquema de base de datos relacional**. Esa es la primera respuesta del diseño de implementación en un DDLC que implementa una base de datos relacional. Además, se corresponde con el nivel conceptual del modelo ANSI/SPARC.

esquema de base de datos relacional. Un listado que muestra los nombres de las relaciones, los nombres de los atributos y las claves foráneas.

Restricciones de integridad

restricción. Una regla que restringe los valores en una base de datos.

Una **restricción** es una regla que limita los valores que pueden estar presentes en la base de datos. El modelo de datos relacional de Codd incluye varias restricciones que se usan para verificar la validación de los datos en una base de datos. Se considerarán las siguientes restricciones:

Integridad de la entidad
Integridad referencial
Dependencias funcionales

Ahora se considerarán la integridad de la entidad y la integridad referencial. Las dependencias funcionales se discutirán más tarde en este capítulo. Las restricciones de integridad proporcionan las bases lógicas para mantener la validación de los valores en la base de datos. Tal capacidad tiene un valor evidente, puesto que uno de los principales propósitos del procesamiento de base de datos es proporcionar información correcta para manejar y tomar decisiones.

Integridad referencial. Las filas en una relación representan instancias en la base de datos de objetos específicos del mundo real o *entidades* (como se llamarán aquí para ser consistentes con la terminología relacional). Por ejemplo, una fila en TRABAJADOR representa un empleado específico, una fila en EDIFICIO representa un edificio específico, una fila en ASIGNACIÓN representa una asignación específica de un empleado a un edificio y así sucesivamente. La clave de una relación identifica unívocamente cada fila y, de aquí, cada instancia de la entidad. De este modo, si los usuarios quieren recuperar o manipular los datos almacenados en una fila específica, deben conocer el valor de la clave de esa fila. Eso significa que no se quiere que una entidad sea representada en la base de datos si no se tiene una identificación completa de los atributos que son claves de la entidad. De este modo, no se permite que la clave, o una parte de la misma sea un valor nulo. Esto se resume en la **regla de integridad de la entidad**:

regla de integridad de la entidad. El atributo que es clave de una fila no puede ser nulo.

> El atributo que es clave de una fila en una relación no puede tener un valor nulo.

Integridad referencial. En relaciones constructivas, las claves foráneas se usan para vincular filas en una relación con filas en otra relación. Por ejemplo, TIPO_DE_OFICIO se usa en la relación TRABAJADOR para decirnos el oficio principal de cada empleado para que pueda calcularse el pago de la prima. Por tanto, es extremadamente importante que el valor de TIPO_DE_OFICIO en cualquier fila de un empleado se corresponda con un valor real TIPO_DE_OFICIO en la relación OFICIO. De otro modo, el TIPO_DE_OFICIO del empleado no apuntaría a ninguna parte. Una base de datos en la cual todas las claves foráneas no nulas referencian valores reales de claves en otra relación cumple la integridad referencial. Así, se tiene la **regla de integridad referencial**:

regla de integridad referencial. El valor no nulo de una clave externa debe ser un valor real de la clave de otra relación.

> Toda clave externa puede también ser nula, o su valor puede ser el valor real de una clave en otra relación.

▼ El proceso de normalización

Considere la relación de la Figura 5.4, que combina parcialmente los datos de TRABAJADOR y de ASIGNACIÓN. Se supone para esta sección que el esquema de base de datos relacional no se transformó de un modelo conceptual, sino que se diseñó directamente a partir de información recogida de usuarios potenciales de bases de datos. También se asume que el diseño original de la base de datos no incluyó la relación de la Figura 5.3, pero sí la relación de la Figura 5.4. Veremos cómo pueden surgir problemas a través de un diseño de base de datos irreflexivo y cómo los problemas pueden evitarse siguiendo un conjunto de principios bien definidos llamado **normalización**.

normalización. El proceso de conversión de una relación en una forma estándar.

Con un pequeño análisis, se puede ver que la relación de la Figura 5.4 no está bien diseñada. Por ejemplo, las cuatro tuplas para el trabajador 1412 repiten el mismo nombre y la información del tipo de oficio. Esta **redundancia en los datos** o repetición, no sólo

redundancia en los datos. Repetición de datos en una base de datos.

TRABAJADOR

ID_TRABAJADOR	NOMBRE	TIPO_DE_OFICIO	ID_SUPV	ID_EDIFICIO
1235	M. Faraday	Electricista	1311	312
1235	M. Faraday	Electricista	1311	515
1412	C. Nemo	Fontanero		312
1412	C. Nemo	Fontanero		460
1412	C. Nemo	Fontanero		435
1412	C. Nemo	Fontanero		515
1311	C. Coulomb	Fontanero		435

integridad de los datos. Consistencia de los datos en una base de datos.

anomalías de actualización. Inconsistencia de los datos como resultado de datos redundantes y actualizaciones parciales.

anomalías de borrado. Pérdida no intencionada de datos debido a que se han borrado otros datos.

anomalías de inserción. Imposibilidad de adicionar datos en la base de datos debido a la ausencia de otros datos.

descomposición de relaciones. División de una relación en múltiples relaciones.

formas normales. Reglas para relaciones estructuradas que eliminan anomalías.

primera forma normal (1FN). Todos los valores de los atributos deben ser atómicos.

valor atómico. Un valor que no es un conjunto de valores o un grupo repetitivo.

ocupa espacio, sino que puede conducir a perder la **integridad de los datos** (pérdida de la consistencia) en la base de datos. El problema surge por el hecho de que un individuo puede estar trabajando en más de un edificio al mismo tiempo. Suponga que el tipo de oficio de C. Nemo está erróneo, y sólo la primera tupla está correcta. Entonces se tendría una inconsistencia entre las tuplas que contienen información sobre C. Nemo. Esto se llama una **anomalía de actualización**.

Suponga que Nemo ha estado enfermo durante tres meses y todos los edificios a los que ha sido asignado se terminaron. Si se decidió borrar todas las filas de la relación que contienen información sobre los edificios terminados, entonces la información sobre el ID trabajador, el nombre y el tipo de oficio de C. Nemo se perderá. Esto se denomina una **anomalía de borrado**. A la inversa, puede tenerse contratado un nuevo empleado llamado Spandolf, al que aún no se le haya asignado edificio alguno. Esto se denomina una **anomalía de inserción**.

Las anomalías de actualización, borrado e inserción son obviamente inconvenientes. ¿Cómo se previenen o se minimizan tales problemas? Claramente, dividiendo la relación TRABAJADOR de la Figura 5.4 en dos relaciones, TRABAJADOR y ASIGNACIÓN, que aparecen en la Figura 5.3, se eliminan las anomalías. Esto es una solución intuitiva. Ahora se mostrará un método más formal llamado descomposición para lograr el mismo resultado. La **descomposición** es el proceso de dividir relaciones en múltiples relaciones para eliminar las anomalías y mantener la integridad de los datos. Para hacer esto, se usan las **formas normales** o reglas para relaciones estructuradas.

Primera forma normal

Una relación está en **primera forma normal (1FN)** si los valores en la relación son **atómicos** para cada atributo en la relación. Esto quiere decir simplemente que los valores de los atributos no pueden ser un conjunto de valores o un *grupo repetitivo*. La definición de Codd de una relación incluye la condición de que la relación esté en primera forma normal. Por tanto, todos los esquemas relacionales que se encontrarán estarán en 1FN. Sin embargo, para aclarar el concepto, veremos un ejemplo de una tabla que no es una relación en 1FN.

En la Figura 5.5 considere los valores introducidos para el atributo ID_EDIFICIO. Allí se han combinado los edificios asignados a cada trabajador en un solo conjunto. El valor del atributo ID_EDIFICIO es el conjunto de edificios en los cuales la persona está trabajando. Suponga que se está interesado en sólo uno de esos edificios. Esta información puede ser difícil de extraer, debido a que el identificador para el edificio de interés está oculto dentro de un conjunto, dentro de una tupla.

La relación en la Figura 5.5 no está en 1FN porque ID_EDIFICIO no es atómica. Eso es, que en una determinada tupla, ID_EDIFICIO puede tener múltiples valores. Sin embargo, la relación que se muestra en la Figura 5.4 estaría en 1FN, puesto que el valor que interesa, que es el de un edificio individual, puede ser identificado simplemente referenciando un nombre de atributo, ID_EDIFICIO.

TRABAJADOR

ID_TRABAJADOR	NOMBRE	TIPO_DE_OFICIO	ID_SUPV	ID_EDIFICIO
1235	M. Faraday	Electricista	1311	{312, 515}
1412	C. Nemo	Fontanero		{312, 460,
	C. Nemo			435, 515}
1311	C. Coulomb	Electricista		435

Debido a que la definición original de Codd del modelo relacional requería que todas las relaciones estuvieran en 1FN, la Figura 5.5 no es ni siquiera una relación propiamente dicha. Se seguirá la definición de Codd y se asumirá que todas las relaciones deben estar en 1FN.

Las próximas dos formas normales, segunda forma normal y tercera forma normal, se aplican a las relaciones que estén restringidas por dependencias funcionales. Antes de proseguir con estas formas normales se debe explicar primero las dependencias funcionales.

Dependencias funcionales

Anteriormente, en este capítulo se estudiaron las restricciones de integridad de la entidad y las restricciones de integridad referencial. Las dependencias funcionales (DFs) proveen una manera para definir restricciones adicionales en un esquema relacional. La idea esencial es que el valor de la tupla en un atributo determina unívocamente el valor de la tupla en otro atributo. Por ejemplo, en la Figura 5.4, en cada tupla, ID_TRABAJADOR determina, especialmente, NOMBRE; ID_TRABAJADOR determina especialmente TIPO_DE_OFICIO. Se escriben estas dos dependencias funcionales de la siguiente forma:

```
FD: ID_TRABAJADOR → NOMBRE
FD: ID_TRABAJADOR → TIPO_DE_OFICIO
```

dependencia funcional. El valor de un atributo en una tupla determina el valor de otro atributo en la tupla.

Más formalmente se define una **dependencia funcional** como sigue: si A y B son atributos en la relación R, entonces

```
FD: A → B
```

significa que si cualesquiera dos tuplas en R tienen el mismo valor para su atributo A, deben tener el mismo valor para su atributo B. Esta definición se aplica también si A y B son conjuntos de columnas en lugar de que columnas simples.

La notación "→" se lee "determina funcionalmente". Por tanto, en estos ejemplos, ID_TRABAJADOR determina funcionalmente NOMBRE, ID_TRABAJADOR determina funcionalmente TIPO_DE_OFICIO y A determina funcionalmente B.

determinante. El atributo de la parte izquierda de una DF determina el valor de los otros atributos en una tupla.

El atributo en la parte izquierda de una DF se llama **determinante** porque ese valor determina el valor del atributo de la parte derecha. La clave de una relación es un determinante, debido a que su valor determina unívocamente el valor de todos los atributos en una tupla.

Segunda forma normal

segunda forma normal (2FN). Los atributos no claves no pueden ser funcionalmente dependientes de una parte de la clave.

La segunda y tercera forma normal se ocupan de la relación entre los atributos claves y no claves. Una relación está en **segunda forma normal (2FN)** si el atributo no clave no es funcionalmente dependiente de una parte de la clave. Por tanto, la 2FN puede violarse sólo cuando una clave sea una clave compuesta o, en otras palabras, que conste de más de un atributo.

Examinemos el esquema relacional de la Figura 5.6. La clave consiste en los atributos ID_TRABAJADOR e ID_EDIFICIO juntos. NOMBRE está determinado por ID_TRABAJADOR y también es fnncionalmente dependiente de una parte de la clave. Eso significa que, conociendo ID_TRABAJADOR, para el trabajador es suficiente para identificar su nombre. Por tanto, la relación no está en 2FN. Dejar esta relación en esta forma, que no es 2FN, puede conducir a los problemas siguientes:

1. El nombre del trabajador se repite en cada fila que se refiera a una asignación para ese trabajador.
2. Si el nombre del trabajador cambia, cada fila que se refiera a una asignación de ese trabajador debe actualizarse. Esto, como se recordará, es una anomalía de actualización.

ASIGNACIÓN (*ID_TRABAJADOR*, *ID_EDIFICIO*, FECHA_INICIO, NOMBRE)

TRABAJADOR

ID_TRABAJADOR	ID_EDIFICIO	FECHA_INICIO	NOMBRE
1235	312	10/10	M. Faraday
1412	312	01/10	C. Nemo
1235	515	17/10	M. Faraday
1412	460	08/12	C. Nemo
1412	435	15/10	C. Nemo

3. Debido a esta redundancia, los datos podrían convertirse en inconsistentes, con distintas filas mostrando diferentes nombres para el mismo trabajador.
4. Si al mismo tiempo no hay asignaciones para el trabajador, puede no haber filas en las cuales guardar su nombre.

Para resolver estos problemas, la relación puede descomponerse en los dos esquemas relacionales siguientes, los cuales están en 2FN:

```
ASIGNACIÓN (ID_TRABAJADOR, ID_EDIFICIO, FECHA_INICIO)
Clave Externa: ID_TRABAJADOR REFERENCIA A TRABAJADOR
TRABAJADOR (ID_TRABAJADOR, NOMBRE)
```

¿Ve que estas relaciones están en 2FN y que eliminan los problemas mencionados anteriormente? Por tanto, la 2FN reduce la redundancia y la inconsistencia. Sin embargo, eso puede complicar ciertas aplicaciones de recuperación. Si se quiere los nombres de todos los trabajadores asignados a un edificio dado, se tiene que combinar la información de estas dos relaciones. En capítulos posteriores se mostrará cómo hacer esto cuando se estudien lenguajes de datos relacionales.

proyección de una relación. Una relación compuesta de atributos seleccionados de otra relación.

Estas dos relaciones más pequeñas que la relación original se llaman **proyecciones**. Es fácil ver que una proyección simplemente selecciona ciertos atributos de una relación existente y los representa como una nueva relación. Esto parece suficientemente simple. El contenido de ASIGNACIÓN y de TRABAJADOR se muestran en la Figura 5.7. Observe que en ASIGNACIÓN se continúa teniendo cinco filas, debido a que los valores para ID_TRABAJADOR, ID_EDIFICIO y FECHA_INICIO, tomados juntos, fueron únicos. Sin embargo, en la relación TRABAJADOR ahora se tienen sólo dos filas, ya que hubo sólo dos únicos conjuntos de valores para ID_TRABAJADOR y NOMBRE. Así, la redundancia de los datos y la posibilidad de anomalías han sido eliminadas.

ASIGNACIÓN

ID_TRABAJADOR	*ID_EDIFICIO*	FECHA_INICIO
1235	312	10/10
1412	312	01/10
1235	515	17/10
1412	460	08/12
1412	435	15/10

TRABAJADOR

ID_TRABAJADOR	NOMBRE
1235	M. Faraday
1412	C. Nemo

El proceso de descomponer una relación que no está en 2FN en dos relaciones que estén en 2FN sigue unos pocos pasos simples, los dos primeros de los cuales se ilustraron en el ejemplo: (1) Crear una nueva relación usando los atributos de la dependencia funcional (DF) culpable como los atributos en la nueva relación. (2) Los atributos de la parte derecha de la DF se eliminan entonces de la relación original. (3) Si más de una DF impide la relación en 2FN, repita los pasos 1 y 2 para cada DF. (4) Si el mismo determinante aparece en más de una DF, tome todos los atributos funcionalmente dependientes con ese determinante como atributos no claves en la relación teniendo al determinante como clave.

Tercera forma normal

tercera forma normal (3FN). Cada determinante es una clave.

Una relación está en **tercera forma normal (3FN)** si para toda DF: $X \rightarrow Y$, X es una clave. Observe que si se sigue la definición de tercera forma normal si una relación está en 3FN, está también en 2FN. Sin embargo, como se verá ahora, lo inverso no es cierto.

Considere la relación TRABAJADOR' de la Figura 5.8. Se ve que

```
DF: ID_TRABAJADOR → TIPO_DE_OFICIO
DF: ID_TRABAJADOR → PRIMA
```

son dependencias funcionales para esta relación debido a que ID_TRABAJADOR es una clave. Sin embargo,

```
DF: TIPO_DE_OFICIO → PRIMA
```

es también una dependencia funcional. Está claro que el criterio de 3FN se satisface para las dos primeras dependencias funcionales, pero ¿qué hay sobre la última dependencia funcional? Obviamente, TIPO_DE_OFICIO no es una clave, por lo que el criterio de 3FN falla. Por tanto, TRABAJADOR' no está en 3FN; sin embargo, TRABAJADOR' *está* en 2FN. (Eso se debe a que su clave consiste en sólo un atributo.) Así, es posible para una relación estar en 2FN sin estar en 3FN.

¿Por qué se debe estar preocupado si una relación no está en 3FN? Los problemas son similares a los problemas comentados en la violación del diseño de 2FN.

1. La prima para cada tipo de oficio se repite en las filas de cada empleado que tiene el mismo tipo de oficio. Esta redundancia de datos gasta espacio de memoria.
2. Si la prima para el tipo de oficio cambia, tales filas deben ser actualizadas. Si una fila es borrada se puede perder el dato de la prima para un tipo de oficio. Por tanto, la relación está sujeta a anomalías de actualización y de borrado.
3. Si hay actualmente empleados no permanentes para un tipo de oficio dado, puede no haber filas en las cuales guardar la prima del tipo de oficio. Esta es una anomalía de inserción.

TRABAJADOR'

ID_TRABAJADOR	TIPO_DE_OFICIO	PRIMA
1235	Electricista	3,50
1412	Fontanero	3,00
1311	Electricista	3,50

Afortunadamente, si se prueba que una relación está en 3FN se puede estar seguro que la relación está en 2FN. Pero, ¿cómo se puede convertir un esquema relacional que no está en 3FN en un conjunto de relaciones que satisfacen la 3FN? La descomposición es el método más fácil. Este es el método usado para convertir una relación que no estaba en 2FN (ASIGNACIÓN de la Figura 5.6) en dos relaciones en 2FN (Figura 5.7). Ahora se mostrará cómo aplicar el proceso de descomposición a relaciones que no están en 3FN.

Se comienza con el esquema relacional de TRABAJADOR'. Forme una relación (*R1*) moviendo de TRABAJADOR' los atributos de la parte derecha de cualquier DF que haga fallar el criterio de 3FN. En el ejemplo, eso es PRIMA. Forme una nueva relación compuesta de los atributos de la parte derecha y de la parte izquierda de la DF que hace fallar el criterio de 3FN. En el ejemplo, estos son PRIMA y TIPO_DE_OFICIO. El determinante de la DF, TIPO_DE_OFICIO, es la clave. Si se llama esta nueva relación *R2*, entonces

```
R1 (ID_TRABAJADOR, TIPO_DE_OFICIO)
Clave Externa: TIPO_DE_OFICIO REFERENCIA A R2
```

y

```
R2 (TIPO_DE_OFICIO, PRIMA)
```

son los esquemas de las dos relaciones que tienen lugar de TRABAJADOR'. Si *R1* o *R2* no están en 3FN, se continúa aplicando el proceso de descomposición hasta que todas las relaciones estén en 3FN. En este caso, *R1* y *R2* están ambas en 3FN y se puede parar.

Se ha descompuesto TRABAJADOR' en dos relaciones *R1* y *R2*, las cuales están en 3FN. Debería convencerse de que *R1* y *R2* están ambas en 3FN. Observe además que TIPO_DE_OFICIO es una clave externa en *R1*.

Debido a que toda relación está, por definición, en 1FN y debido a que las relaciones en 3FN están siempre en 2FN, la siguiente cadena de implicaciones es válida:

```
3FN implica 2FN implica 1FN
```

Por esta razón, poner las relaciones en 3FN y en 2FN es lo único que se necesita para usar el criterio de 3FN. Si se verifica que cada determinante en cada relación es una clave de esa relación —el criterio de 3FN—, entonces todas las relaciones están en primera, segunda y tercera forma normal. Esto simplifica grandemente el proceso de normalización, debido a que ahora se necesita comprobar sólo un criterio.

forma normal. Boyce-Codd (FNBC). Todo determinante es una clave.

Esta versión de tercera forma normal se llama a menudo **Forma Normal Boyce-Codd (FNBC)**. El criterio usado para la 3FN por muchos autores es lógicamente un poco más débil que el criterio de FNBC que se está usando. Este criterio más débil establece que una relación está en 3FN si no tiene dependencias transitivas. Una **dependencia transitiva** aparece cuando un atributo no clave es funcionalmente dependiente de uno o más de otros atributos no claves. Este criterio no maneja dos casos:

dependencia transitiva. Aparece cuando un atributo no clave es funcionalmente dependiente de uno o más atributos no claves.

1. Un atributo no clave es dependiente de un atributo clave en una clave compuesta (el criterio para relaciones que no están en 2FN).
2. Un atributo clave en una clave compuesta es dependiente de un atributo no clave.

FNBC maneja ambos casos. Por tanto, si una relación está en FNBC entonces está también en 3FN, en el sentido de la dependencia transitiva, y está también en 2FN. Se ve entonces que esta definición de 3FN simplifica mucho el proceso de normalización.

Cuarta forma normal

La primera forma normal prohíbe relaciones donde se tengan atributos no atómicos o multievaluados. Sin embargo, existen muchas situaciones de modelado de datos, en que las relaciones requieren atributos multievaluados. Por ejemplo, en un sistema colectivo de seguro sanitario, las múltiples dependencias de un empleado pueden ser rastreadas. En un escenario escolar, un miembro de la facnltad se asigna a múltiples comisiones y es responsable de múltiples cursos. Ejemplos como estos podrían ocurrir muchas veces. ¿Cómo se podrían modelar en una base de datos relacional que no permitiese atributos multievaluados?

La Figura 5.9 muestra cuatro posibles aproximaciones para resolver este problema para los cursos y las comisiones de los miembros de la facultad. Cada solución parece insatisfactoria en algún modo. Todas gastan espacio, tanto por usar valores nulos como por introducir datos redundantes. Aquellas que usan valores nulos violan la integridad de la entidad, debido a que todos los atributos juntos constituyen la clave de la relación. Además, suponga que Jones fue asignado al comité de Promoción, por lo que se requirió una actualización. ¿Se debería añadir una nueva fila o actualizar una fila existente? Finalmente, no está claro que los atributos COMITÉ y CURSO sean independientes uno del otro. Por ejemplo, en la Figura 5.9(a), ¿el comité de Admisión se relaciona con el curso IM101?

La relación aparente entre atributos independientes puede eliminarse requiriendo que cada valor de un atributo aparezca en al menos una fila con cada valor del otro atributo. Esto se ilustra en la Figura 5.10, donde ambas Admisión y Beca aparecen en las filas con IM101, IM102, IM103.

FACULTAD

FNOMBRE	*COMITÉ*	*CURSO*
Jones	Admisión	IM101
Jones	Beca	IM102
Jones	Beca	IM103

a) Número mínimo de registros con repeticiones

FACULTAD

FNOMBRE	*COMITÉ*	*CURSO*
Jones	Admisión	IM101
Jones	Beca	IM102
Jones		IM103

b) Número mínimo de registros con valores nulos

FACULTAD

FNOMBRE	*COMITÉ*	*CURSO*
Jones	Admisión	
Jones	Beca	
Jones		IM101
Jones		IM102
Jones		IM103

c) Filas sin repeticiones

FACULTAD

FNOMBRE	*COMITÉ*	*CURSO*
Jones	Admisión	IM101
Jones	Beca	
Jones		IM102
Jones	Beca	IM103

d) No restringida

FACULTAD	*FNOMBRE*	*COMITÉ*	*CURSO*
	Jones	Admisión	IM101
	Jones	Beca	IM101
	Jones	Admisión	IM102
	Jones	Beca	IM102
	Jones	Admisión	IM103
	Jones	Beca	IM103

dependencia multievaluada (DMV). Una restricción que garantiza la independencia mutua de atributos multievaluados.

cuarta forma normal (4FN). Una relación que está en 3FN y no tiene dependencias multievaluadas.

Una condición que hace cumplir la independencia de los atributos requiriendo esta duplicación de valores se llama una **dependencia multievaluada (DMV).** Las DMVs son restricciones exactamente como lo son las DFs. Claramente, debido a que requieren una enorme duplicación de valores de datos, un aspecto importante del proceso de normalización debería ser eliminar las dependencias multievaluadas. Esto se hace con la cuarta forma normal.

Una relación está en **cuarta forma normal (4FN)** si está en 3FN y no tiene atributos multievaluados. Debido a que el problema de las dependencias multievaluadas surge de los atributos multievaluados, se puede encontrar una solución poniendo todos los atributos multievaluados en relaciones formadas por ellos mismos, junto con la clave a la cual se aplican los valores de los atributos. La Figura 5.11 ilustra esto. FNOMBRE es un valor clave en alguna otra relación que identifica el miembro de la facultad para el cual se aplica la información. Se listan las comisiones a las que Jones está asignado, incluyendo una fila por cada comité. El nombre Jones se repite cada vez. Lo mismo sucede con los cursos que Jones dicta. Las relaciones en la Figura 5.11 están en cuarta forma normal (4FN) porque todos los atributos multievaluados (en este caso, COMITÉ y CURSO) han sido puestos en relaciones formadas por ellos mismos. Además, este enfoque supera los problemas de varios de los enfoques que se mostraron en la Figura 5.9. Como nota final debe señalarse que las claves de estas relaciones en 4FN son *ambos* atributos de la relación. Es decir, la clave de FAC_COM es (FNOMBRE, COMITÉ) y la clave de FAC_CURSO es (FNOMBRE, CURSO).

Otras formas normales

Se han propuesto algunas otras formas normales eliminar anomalías adicionales. Se discutirán brevemente dos de estas formas: la quinta forma normal (5FN) y la forma normal dominio/clave (FNDLL).

quinta forma normal (5FN). Una forma normal que elimina las dependencias de reunión (*join*; también unión).

Quinta forma normal. Las restricciones de dependencia funcional y dependencia multievaluada resultan necesarias para la segunda, tercera y cuarta forma normal. La **quinta forma normal (5FN)** elimina las anomalías que resultan de un tipo de restricción lla-

FAC_COMITÉ	*FNOMBRE*	*COMITÉ*
	Jones	Admisión
	Jones	Beca
FAC_CURSO	*FNOMBRE*	*CURSO*
	Jones	IM101
	Jones	IM102
	Jones	IM103

mado *dependencias de reunión (join dependencies*, dependencias de unión). Estas dependencias son principalmente de interés teórico y de muy dudoso valor práctico. Por ello, la quinta forma normal no tiene virtualmente aplicación práctica.

forma normal dominio clave (FNDLL). Requiere que cada restricción sea resultado de definiciones de dominios y claves.

Forma normal dominio/clave. Fagin (1981) propuso una forma normal basada en las definiciones de claves y dominios de atributos. Mostró que una relación está en **forma normal dominio/clave (FNDLL)** si y sólo si cada restricción en la relación es una consecuencia de las definiciones de dominios y claves. Esto es un resultado importante. Sin embargo, no proporcionó un método general para convertir una relación que no está en FNDLL en una relación que esté en FNDLL.

▼ Transformar un modelo conceptual en un modelo relacional

Generalmente se acepta que los modelos conceptuales ofrecen una representación de las complejidades de un problema de aplicación más precisa que la del modelo relacional y otros modelos de datos anteriores. Por eso, en el Capítulo 4 se discutió el modelado conceptual y se demostró cómo una gran variedad de problemas de aplicación podrían resolverse con los modelos conceptuales. Sin embargo, en la actualidad existen pocos sistemas en los cuales los modelos conceptuales pueden implementarse. Debido a que se está estudiando el modelo relacional en este capítulo, se centrará la atención en los métodos de transformación de un modelo conceptual en un modelo relacional.

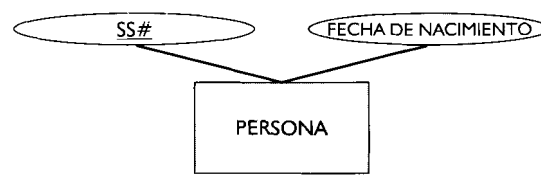
Un modelo de datos conceptual consta de objetos, interrelaciones, atributos, especializaciones, agregados, etc. Ahora se mostrarán los métodos de transformación de cada uno de ellos. Una importante característica del proceso que se describirá es que da como resultado la creación de relaciones normalizadas para la cuarta forma normal. Consecuentemente, la siguiente conversión de un modelo conceptual en un modelo relacional no necesita el proceso de normalización.

Transformar conjuntos de objetos y atributos

Considere el simple modelo conceptual de la Figura 5.12. Se ve un conjunto de objetos con dos atributos. PERSONA es un conjunto de objetos abstractos (no léxicos), pero #SS y FECHA_NAC son atributos léxicos. Debido a que los atributos en el modelo relacional deben ser atributos léxicos, #SS y FECHA_NAC pueden ser ambos atributos en una relación. Por tanto, se transforma este diagrama en una relación con atributos como sigue:

```
PERSONA (#SS, FECHA_NAC)
```

¿Cuál es la clave? Debido a que se puede asumir (y se asumirá) que #SS identifica unívocamente una persona, se concluye que #SS es una clave. Se tiene entonces



```
PERSONA (#SS, FECHA_NAC)
```

como la transformación del modelo conceptual de la Figura 5.12 a un modelo relacional.

Transformar modelos sin claves externas

Considere la Figura 5.13. Se puede intentar transformar este modelo de la misma forma, resultando:

```
VENTA (IMPORTE, #PRODUCTO)
```

Pero en este caso no hay clave, debido a que puede haber muchas ventas que tengan el mismo valor para IMPORTE Y #PRODUCTO. Por tanto, se debe añadir un atributo clave, #VENTA:

```
VENTA (#VENTA, IMPORTE, #PRODUCTO)
```

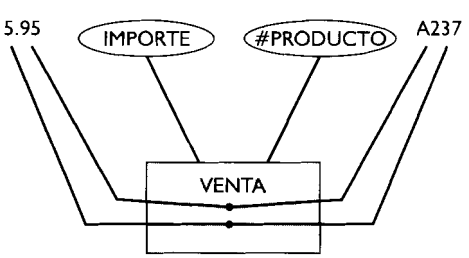
Se puede resumir este proceso como sigue. Un conjunto de objetos con atributos puede transformarse en una relación usando el nombre del conjunto como el nombre de la relación y los atributos del conjunto de objetos como los atributos de la relación. Si puede usarse un conjunto de estos atributos como clave para la relación, entonces esos atributos se convierten en la clave de la relación. En caso contrario, se añade un atributo a la relación con el acuerdo de que ese valor identifique unívocamente instancias del objeto en el conjunto de objetos original y pueda, por tanto, servir como una clave para la relación. En el ejemplo anterior, se añadió #VENTA como un atributo de la relación que significa que #VENTA identifica unívocamente una instancia en el conjunto de objetos VENTA.

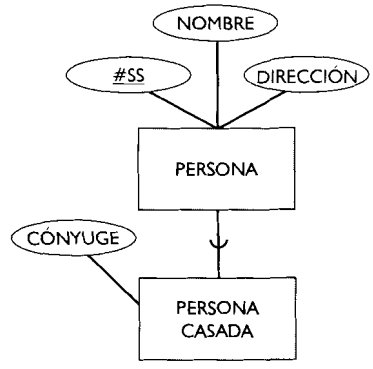
Se ha mostrado un método simple y rápido para generar una clave para una relación en caso de que no tenga una. Actualmente, el diseñador de bases de datos puede consultar con el usuario la selección de una clave para la relación. A menudo habrá un número de facturas o algún otro valor que se registra y que puede servir como la clave de la relación. Esto sería la selección lógica para el atributo clave. Otra posibilidad es que el usuario recomiende añadir una combinación de atributos que juntos constituirían una clave. El analista tiene la responsabilidad de trabajar con los usuarios para determinar qué atributos compondrán la clave.

Transformar la especialización y la generalización de conjuntos de objetos

Ahora considere la Figura 5.14. El conjunto de objetos PERSONA se transforma fácilmente:

```
PERSONA (#SS, NOMBRE, DIRECCIÓN)
```





Pero ¿qué se hace con PERSONA_CASADA? Debido a que es un subconjunto de PERSONA, hereda todos los atributos de PERSONA. Además tiene sus atributos propios. Consecuentemente, se deriva la siguiente relación:

```
PERSONA_CASADA (#SS, NOMBRE, DIRECCIÓN, CÓNYUGE)
Clave Externa: #SS REFERENCIA A PERSONA
```

Así, una especialización de un conjunto de objetos tendrá todos los atributos del conjunto de objetos que él especializa más todos sus atributos propios. Las dos relaciones tendrán la misma clave. Es importante señalar que la clave (#SS) de la relación de especialización (PERSONA_CASADA en el ejemplo) es también una clave externa, que indica la relación de generalización (PERSONA en el ejemplo). Consecuentemente, NOMBRE y DIRECCIÓN en PERSONA_CASADA constituyen información duplicada. Para eliminar esta redundancia de datos, simplemente se suprimen de la relación de especialización todos los atributos no claves que están duplicados. Finalmente, entonces la relación PERSONA_CASADA es:

```
PERSONA_CASADA (#SS, CÓNYUGE)
Clave Externa: #SS REFERENCIA A PERSONA
```

Transformar interrelaciones

Las interrelaciones se transforman en tres formas diferentes dependiendo de la cardinalidad de las interrelaciones. Se manejarán las interrelaciones uno-uno, uno-muchos y muchos-muchos separadamente.

Interrelaciones Uno-Uno. Se tomará un ejemplo del Banco Alchemical. La interrelación TIENE_C_CUENTA, mostrada en la Figura 5.15, es uno-uno. Eso es que un cliente tiene a lo sumo un control de cuenta y un control de cuenta se asigna sólo a un cliente. Si se asumen los atributos claves #CLIENTE para CLIENTE y #C_CUENTA para CUENTA, entonces se tienen dos relaciones de una columna:

```
CLIENTE (#CLIENTE)
CONTROL_CUENTA(#C_CUENTA)
```

Para mostrar la conexión entre las dos relaciones, se puede adicionar #C_CUENTA a CLIENTE y #CLIENTE a CUENTA. Note que cada una de estas columnas es una clave externa que indica a la otra relación:

```
CLIENTE (#CLIENTE, #C_CUENTA)
Clave Externa: #C_CUENTA REFERENCIA A CUENTA
CUENTA (#C_CUENTA, #CLIENTE)
Clave Externa: #CLIENTE REFERENCIA A CLIENTE
```

Obviamente, esta solución duplica datos, debido a que sólo es necesaria una combinación de #CLIENTE y #C_CUENTA. ¿Cuál de las dos podría eliminarse? Si un cliente no tiene un control de cuenta, entonces en la relación CLIENTE #C_CUENTA es nulo para ese cliente. Sin embargo, en la relación CUENTA habrá siempre un #CLIENTE para cada #C_CUENTA. Esto se muestra por las cardinalidades mínimas de la interrelación, tal como se ilustró en la Figura 5.15. Eso es, la cardinalidad mínima en la parte de CUENTA es cero, mientras que la cardinalidad mínima en la parte de CLIENTE es uno. Sin embargo, en este caso la mejor selección sería eliminar #C_CUENTA de la relación CLIENTE. Esto da:

```
CLIENTE (#CLIENTE)
CUENTA (#C_CUENTA, #CLIENTE)
Clave Externa: #CLIENTE REFERENCIA A CLIENTE
```

Por supuesto, en un esquema completo de base de datos para una aplicación real, ambas relaciones tendrán muchos más atributos. Aquí se mostraron sólo los atributos necesarios para transformar el simple modelo conceptual de la Figura 5.15. Si los conjuntos de objetos en la Figura 5.15 tuvieran atributos adicionales, éstos estarían puestos en las relaciones correspondientes a ellos. Por ejemplo, CLIENTE pudiera tener como atributos NOMBRE, DIRECCIÓN y #TELÉFONO, y CUENTA pudiera tener como atributos SALDO y FECHA_APERTURA. Estos atributos adicionales causarían en el esquema de base de datos esta apariencia:

```
CLIENTE (#CLIENTE, NOMBRE, DIRECCIÓN, #TELÉFONO)
CUENTA (#C_CUENTA, #CLIENTE, SALDO, FECHA_APERTURA)
Clave Externa: #CLIENTE REFERENCIA A CLIENTE
```

En resumen, las interrelaciones uno-uno se transforman mostrando uno de los conjuntos de objetos como un atributo de la otra relación. La relación elegida se determina por la necesidad de la propia aplicación. En muchos casos, cualquier relación puede elegirse.

Relaciones Uno-Muchos. Suponga que la interrelación TIENE_C_CONTROL tuvo "mucha" cardinalidad en la parte del conjunto de objetos CUENTA. Eso es, un cliente puede tener muchas cuentas, pero una cuenta se asigna sólo a un cliente. En este caso, la propia cardinalidad uno-muchos determina que la estructura de la relación deba ser como sigue:

```
CUENTA (#C_CUENTA, #CLIENTE)
Clave Externa: #CLIENTE REFERENCIA A CLIENTE
CLIENTE (#CLIENTE)
```

Así, en una interrelación uno-muchos, la relación que describe el objeto en la parte "mucha" de la interrelación recibe la columna clave externa que indica al otro objeto. En

el ejemplo, CUENTA es la relación correspondiente al conjunto de objetos de la parte "mucha" de la interrelación y por esta razón CUENTA recibe la columna clave externa #CLIENTE.

Relaciones muchos-muchos. La Figura 5.16 muestra TIENE_C_CONTROL como una interrelación muchos-muchos. En este contexto se asume que un cliente puede tener múltiples controles de cuentas y que un control de cuenta puede asignarse a múltiples clientes. Para transformar las interrelaciones muchos-muchos, se crea una **relación de intersección**. En este caso, se necesitan tres relaciones: una relación para cada conjunto de objetos y otra para la interrelación TIENE_C_CONTROL:

relación de intersección. Una relación que representa instancias donde otras dos relaciones se encuentran en una interrelación muchos-muchos.

```
CLIENTE (#CLIENTE)
CUENTA (#C_CUENTA)
TIENE_C_CUENTA (#CLIENTE, #C_CUENTA)
Claves Foráneas: #CLIENTE REFERENCIA A CLIENTE
#C_CUENTA REFERENCIA A CUENTA
```

Debido a que #CLIENTE no determina #C_CUENTA y #C_CUENTA no determina #CLIENTE, la clave de la interrelación TIENE_C_CUENTA en ambas columnas. TIENE_C_CUENTA contiene los datos que identifican qué clientes están asociados con qué controles de cuentas. La Figura 5.17 muestra algunos ejemplos de datos para estas tres relaciones. Note que los clientes 2222 y 1111 están asociados cada uno con dos controles de cuentas diferentes y las cuentas CA777 y CA888 están asociados cada una con dos clientes diferentes. Estas interrelaciones están indicadas en la relación TIENE_C_CONTROL.

CLIENTE
#CLIENTE

1111
2222
3333

CUENTAS
#C CUENTA

CA888
CA777
CA999

TIENE C CONTROL

# CLIENTE	*# C CUENTA*
2222	CA999
2222	CA888
3333	CA777
1111	CA777
1111	CA888

Observe además que sólo las columnas claves de CLIENTE y CUENTA se usan en TIENE_C_CONTROL. Es decir, incluso si CLIENTE y CUENTA tuvieran otras columnas, TIENE_C_CONTROL usaría sólo las columnas claves de estas dos relaciones. El siguiente esquema ilustra esto:

```
CLIENTE (#CLIENTE, NOMBRE, DIRECCIÓN, #TELÉFONO)
CUENTA (#C_CUENTA, #CLIENTE, SALDO, FECHA_APERTURA)
TIENE_C_CONTROL (#CLIENTE, #C_CUENTA)
Claves Foráneas: #CLIENTE REFERENCIA A CLIENTE
          #C_CUENTA REFERENCIA A CUENTA
```

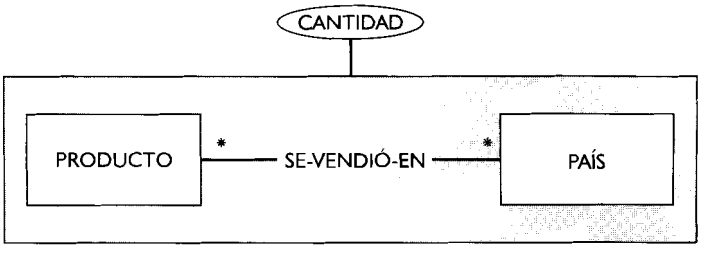
Las descripciones de claves foráneas indican que ambos de los atributos claves de TIENE_C_CONTROL son también claves foráneas. TIENE_C_CUENTA se llama una *relación de intersección* porque ella representa las instancias donde CLIENTE y CUENTA se encuentran. Como se verá en la próxima sección, es posible para una relación de intersección, tal como TIENE_C_CUENTA, tener atributos adicionales no claves que se aplican sólo a ella.

Transformar conjuntos de objetos agregados

Considere la Figura 5.18, la cual muestra un modelo conceptual para el rastreo de ventas de International Product Distribution. SE_VENDIÓ_EN, una interrelación agregada que se considera un conjunto de objetos, tiene un atributo CANTIDAD. Se transforma este modelo de acuerdo con las reglas dadas anteriormente. Debido a que esta interrelación es de muchos-muchos, se crean tres relaciones:

```
PRODUCTO (#PRODUCTO)
PAÍS (NOMBRE_PAÍS)
SE_VENDIÓ_EN (#PRODUCTO, NOMBRE_PAÍS, CANTIDAD)
Claves Foráneas: #PRODUCTO REFERENCIA A PRODUCTO
                 NOMBRE_PAÍS REFERENCIA A PAÍS
```

Se han creado los atributos claves #PRODUCTO y NOMBRE_PAÍS para distinguirlos de los nombres de sus relaciones respectivas. También se tiene CANTIDAD en la relación SE_VENDIÓ_EN porque es un atributo que se aplica a esa relación. Si SE_VENDIÓ_EN tiene otros atributos que se le aplican, ellos serían añadidos de una manera similar. Las relaciones PRODUCTO y PAÍS se muestran con un solo atributo, pero por supuesto ellos podrían tener otros atributos. Si no se necesitan otros atributos para estas relaciones en la base de datos, entonces ellos pueden ser eliminados del esquema y el esquema consistirá solamente de la relación SE_VENDIÓ_EN.



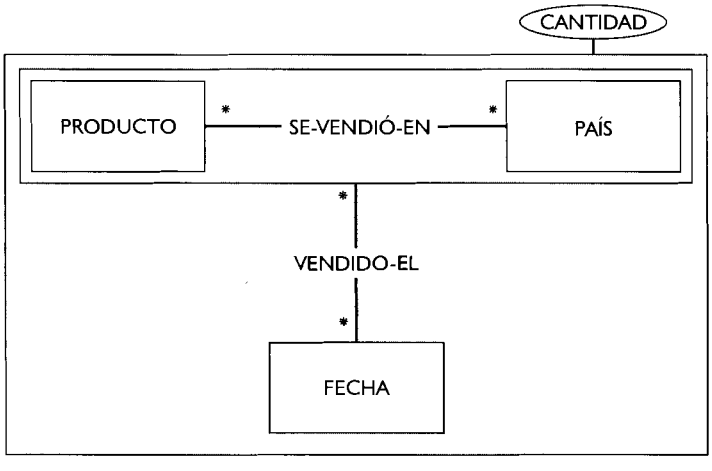
La Figura 5.19 muestra SE_VENDIÓ_EN relacionada con FECHA. En este caso, el atributo CANTIDAD se aplica a aquellos productos vendidos en un país determinado en una fecha determinada. Se transforma este modelo en un modelo relacional como:

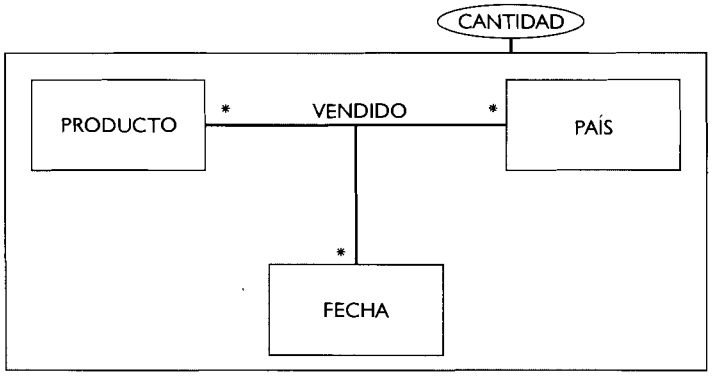
```
SE_VENDIÓ_EN (#PRODUCTO, NOMBRE_PAÍS)
VENDIDO_EL (#PRODUCTO, NOMBRE_PAÍS, FECHA, CANTIDAD)
```

(Se han omitido las relaciones de una columna que definen los conjuntos de objetos.) Note, sin embargo, que toda esta información contenida en SE_VENDIÓ_EN está también contenida en VENDIDO_EL. Por tanto, se puede eliminar SE_VENDIÓ_EN del esquema. Si SE_VENDIÓ_EN tuviera atributos no claves, entonces no podría eliminarse. Eliminando SE_VENDIÓ_EN del esquema, estamos diciendo que el modelo de la Figura 5.19 es esencialmente una interrelación de tres-vías. Por tanto, es equivalente a la Figura 5.20. El modelo relacional para la Figura 5.20 sería:

```
VENDIDO (#PRODUCTO, NOMBRE_PAÍS, FECHA, CANTIDAD)
```

que es el mismo que el anterior VENDIDO_EL, excepto por el nombre de la relación.





Transformar interrelaciones recursivas

La Figura 5.21 muestra una parte del modelo de la Figura 5.1. Es importante comprender que el conjunto de objetos TRABAJADOR, que aparece dos veces en el diagrama, es el mismo conjunto de objetos ambas veces. Ambas copias de TRABAJADOR tienen los mismos atributos, aun cuando ellos son sólo mostrados unidos a la copia de la derecha. El modelo usa dos copias de TRABAJADOR por conveniencia en la muestra de la interrelación SUPERVISA, que existe entre TRABAJADOR y TRABAJADOR. Esta interrelación se llama recursiva porque existe entre objetos del mismo conjunto. En este caso, la interrelación con su cardinalidad uno-a-muchos significa que un trabajador supervisa muchos trabajadores.

¿Cómo se transforma la relación TRABAJADOR junto con sus atributos y la interrelación SUPERVISA en una relación? Usando las propuestas ya estudiadas se llega a lo siguiente:

```
TRABAJADOR (ID_TRABAJADOR, NOMBRE, TARIFA_HR, ID_TRABAJADOR)
```

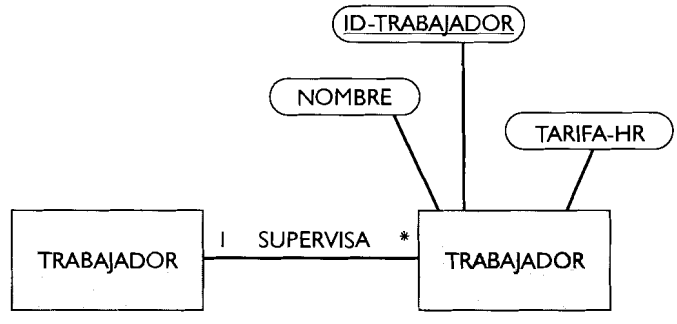
Esta solución es incorrecta porque la relación TRABAJADOR tiene dos atributos llamados ID_TRABAJADOR y no se permite que dos atributos en una relación tengan el mismo nombre. La solución es cambiar el nombre del segundo atributo ID_TRABAJADOR por un nombre que refleje la interrelación SUPERVISA que representa. Por tanto, se cambia el nombre del atributo por ID_SUPV:

```
TRABAJADOR (ID_TRABAJADOR, NOMBRE, TARIFA_HR, ID_SUPV)
Clave Externa: ID_SUPV REFERENCIA A TRABAJADOR
```

Observe que ID_SUPV es una clave externa recursiva, debido a que referencia a ID_TRABAJADOR, la cual es la clave de la relación en la que ID_SUPV está. Las claves foráneas recursivas son el resultado de la transformación de las interrelaciones recursivas. En la Figura 5.22 se muestran ejemplos de datos para la relación TRABAJADOR.

En resumen, se han mostrado las vías de traducir las construcciones del modelo conceptual —objetos, atributos, interrelaciones, especializaciones y agregados— a relaciones. Después de completarse la traducción, el esquema resultante debería revisarse por las redundancias. Las relaciones redundantes, es decir, relaciones cuya información está totalmente contenida en otras relaciones en el esquema, deberían eliminarse del esquema.

Además, vea que todas las relaciones están normalizadas a la cuarta forma normal. La razón para esto es la siguiente: Las dependencias funcionales, definidas para el modelo relacional, son atributos, interrelaciones uno-uno o interrelaciones uno-muchos. El pro-



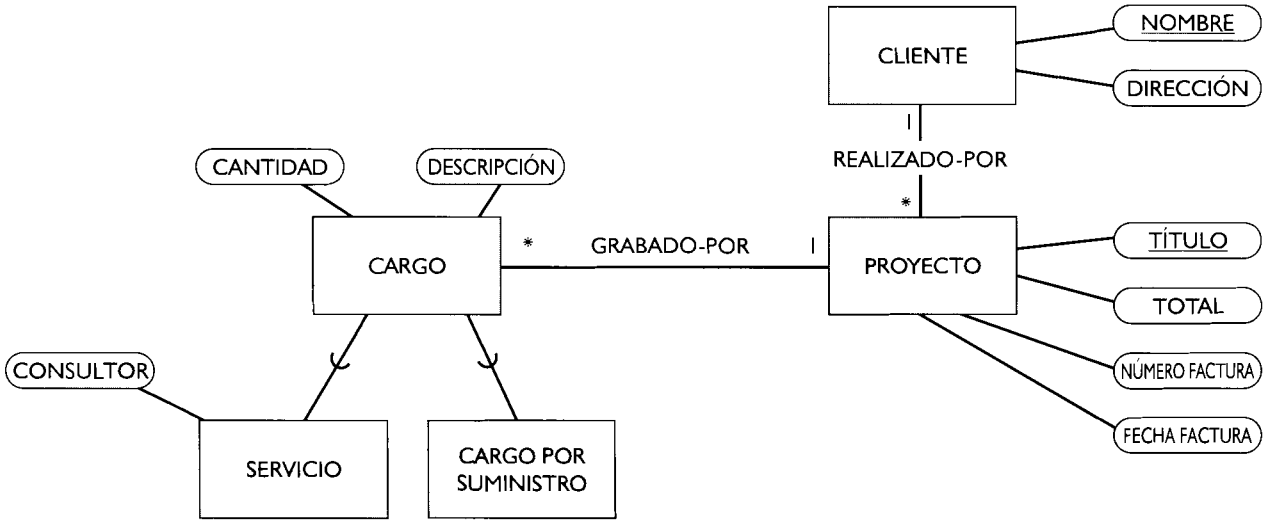
TRABAJADOR *ID_TRABAJADOR*	NOMBRE	TARIFA	ID_SUPV
1235	M. Faraday	12,50	1311
1412	C. Nemo	13,75	1520
2920	R. Garret	10,00	
63231	P. M ason	17,40	
1520	H. Rickover	11,75	
1311	C. Coulomb	15,50	
3001	J. Barrister	8,20	3231

ceso que se describió para convertir cada una de éstas a atributos en una relación garantizó que serían sólo dependientes de los atributos claves. De esta manera, toda relación estará en 3FN. Los atributos multievaluados del modelo relacional existen sólo en interrelaciones muchos-muchos. Debido a que éstos son convertidos a relaciones cuyas claves compuestas consisten en las claves de los conjuntos de objetos individuales, están seguro en 4FN.

Ejemplos de transformaciones: Servicios de Consultoría Manwaring

En el capítulo 4 se creó un modelo conceptual para la facturación de proyectos de los Servicios de Consultoría Manwaring. Este modelo se muestra en la Figura 5.23. Aplicando las reglas de este capítulo, se convertirá este modelo en un esquema relacional.

Las relaciones para los conjuntos de objetos PROYECTO y CLIENTE son como siguen:



```
CLIENTE (NOMBRE_CLIENTE, DIR_CLIENTE)
PROYECTO (#PROYECTO, NOMBRE_CLIENTE, TÍTULO_PROYECTO,
          TOTAL_CARGO, #FACTURA, FECHA_FACTURA)
Clave Externa: NOMBRE_CLIENTE REFERENCIA A CLIENTE
```

#PROYECTO se añadió como un atributo porque se necesitaba una clave externa para PROYECTO. TÍTULO_PROYECTO no identifica unívocamente un proyecto. #FACTURA es único, pero no es una clave apropiada para la relación que describe proyectos. Para mayor claridad, se han cambiado algunos de los nombres de los atributos, por ejemplo, NOMBRE_CLIENTE y TÍTULO_PROYECTO. El cambio a NOMBRE_CLIENTE fue esencial, debido a que se usa como clave externa en PROYECTO.

Además de estas dos relaciones, se necesitan relaciones para CARGO y SERVICIO:

```
CARGO (#CARGO, #PROYECTO, CANTIDAD, DESCRIPCIÓN)
Clave Externa: #PROYECTO REFERENCIA PROYECTO
SERVICIO (#CARGO, #PROYECTO, CONSULTOR)
Clave Externa: #CARGO, #PROYECTO REFERENCIA CARGO
```

En la conversión del conjunto de objetos CARGO a una relación, se tuvo que tomar una decisión sobre su clave. En este caso se decidió que el proyecto al que se le aplique un cargo específico identifica parcialmente. Sin embargo, se usa #PROYECTO como una parte de la clave. Se añadió entonces #CARGO como la parte que queda de la clave. Esto significa que diferentes cargos pueden tener el mismo #CARGO si no están relacionados con el mismo proyecto.

El conjunto de objetos SERVICIO es una especialización del conjunto de objetos CARGO. Por esta razón, la relación SERVICIO debe tener la misma clave que la relación CARGO. Por supuesto, esto significa que esta clave de dos columnas es una clave externa que referencia a CARGO. Además, la relación SERVICIO tiene el atributo CONSULTOR, que indica qué consultor ejerce el servicio. Aunque CARGO POR SUMINISTRO es también una especialización de CARGO, no merece su propia relación porque no tiene atributos propios. Toda la información necesaria sobre cargos por suministros puede encontrarse en la relación CARGO.

La Figura 5.24 muestra el modelo de datos desarrollado para la facturación de proyectos Manwaring en el capítulo 4. Ahora se convertirá esto a un modelo relacional. Los conjuntos de objetos CLIENTE y PROYECTO se convierten como antes:

```
CLIENTE (NOMBRE_CLIENTE, DIR_CLIENTE)
PROYECTO (#PROYECTO, NOMBRE_CLIENTE, TÍTULO_PROYECTO,
TOTAL_CARGO, #FACTURA, FECHA_FACTURA)
Clave Externa: NOMBRE_CLIENTE REFERENCIA A CLIENTE
```

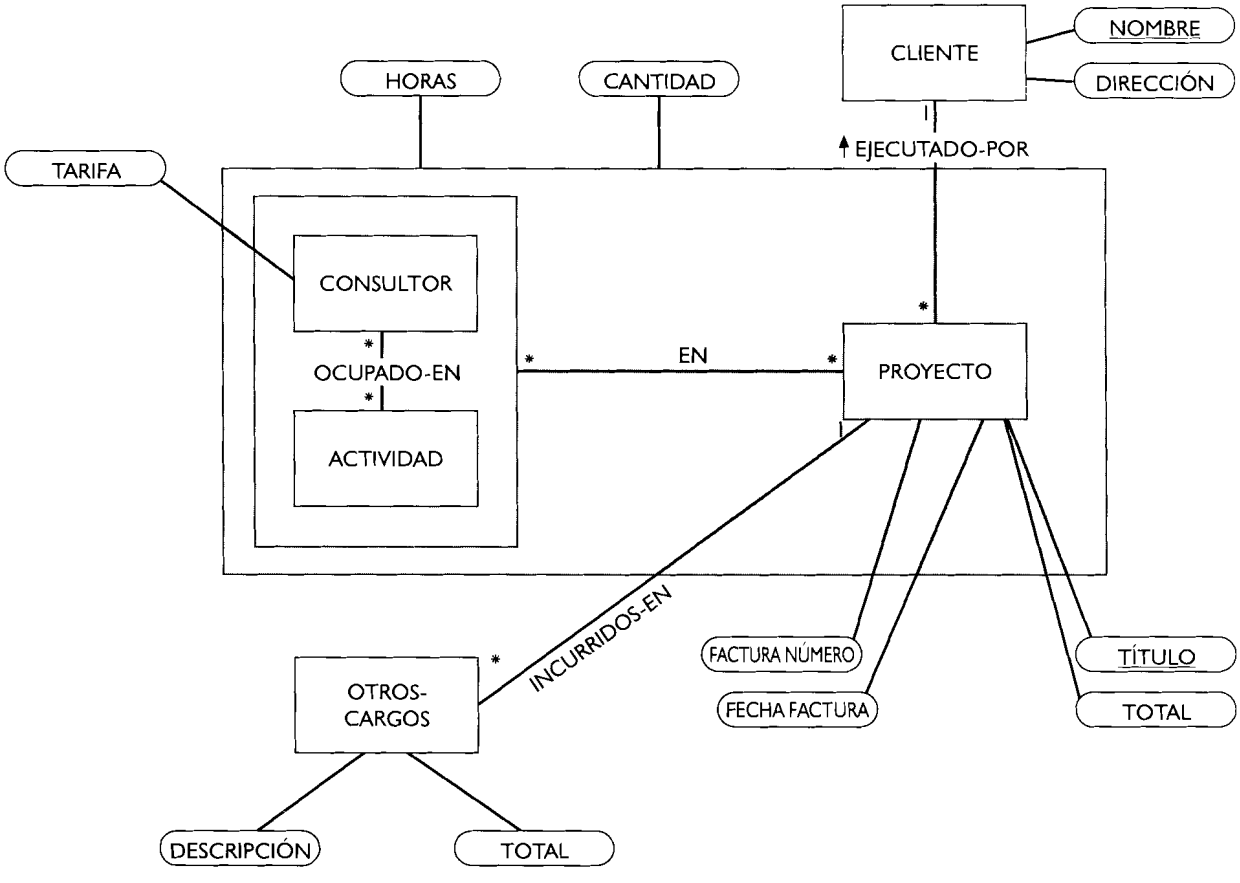
El conjunto de objetos OTRO_CARGO en este modelo corresponde a CARGO en el modelo anterior:

```
OTRO_CARGO (#CARGO, #PROYECTO, CANTIDAD, DESCRIPCIÓN)
Clave Externa: #PROYECTO REFERENCIA A PROYECTO
```

También se necesita una relación para CONSULTOR:

```
CONSULTOR (NOMBRE_CONSULTOR, PAGO)
```

Finalmente, se necesita representar la interrelación en una gran caja. Debido a que la interrelación OCUPADO_EN no tiene atributos propios, se puede empotrar en la relación representada por la interrelación EN. Esta relación es equivalente a una relación de tres-vías, también su clave consiste de tres atributos. Además ella tiene los atributos no claves HORAS y CANTIDAD.




```
OCUPADO_EN_ACTIVIDAD_EN (NOMBRE_CONSULTOR, ACTIVIDAD, #PROYECTO,
                         HORAS, CANTIDAD)
Claves Ajenas: NOMBRE_CONSULTOR REFERENCIA A CONSULTOR
               #PROYECTO REFERENCIA A PROYECTO
```

Debido a que "EN" no da una descripción muy buena del significado de la relación, se le ha dado este nuevo nombre, que es más descriptivo.

Aunque el proceso de conversión de un modelo conceptual a un modelo relacional es mecánico y lineal, requiere alguna inteligencia humana. Se ha ilustrado esto en los dos ejemplos anteriores, donde se seleccionaron claves de varias maneras y se cambiaron los nombres de atributos y relaciones donde se necesitó. No obstante, el proceso es comparativamente simple y todas las relaciones resultantes están en cuarta forma normal.

▼ Comparación del modelado de datos conceptual y relacional

Los estudiantes, particularmente aquellos que han tenido experiencias con los SGBD relacionales, algunas veces se preguntan por qué nos preocupamos con el modelado de datos conceptual cuando finalmente se implementará el sistema de base de datos en el

modelo relacional de todas formas. ¿Por qué crear un modelo conceptual para todo? ¿Por qué no hacer simplemente en primer lugar el modelado de los datos con el modelo relacional, usando la teoría de las formas normales para eliminar las anomalías potenciales? ¿Por qué no ahorrar un poco de tiempo y energía? Estas son preguntas importantes que merecen una respuesta. Quizás se las ha planteado.

La respuesta a estas cuestiones gira alrededor de una simple palabra: complejidad. Cuando el modelo para una base de datos se transforma en un modelo más complejo, eso hace más difícil comprenderlo y por lo tanto más difícil desarrollarlo correctamente. Esta complejidad aumenta con la adición de conjuntos de objetos, especializaciones, agregados e interrelaciones. Debido a que el valor definitivo de la base de datos está en su uso para proporcionar información a directores y ejecutivos, es vital que la estructura de la base de datos esté lógicamente sin defectos. Aunque la falibilidad humana hace imposible garantizar que la metodología del modelado asegurará completamente los modelos de datos correctos, la aproximación gráfica del modelado conceptual hace que la probabilidad de que los modelos correctos sea mucho mayor que la aproximación textual del modelado relacional.

Se ilustra este punto con un ejemplo. Considere el modelo relacional que fue desarrollado anteriormente para la facturación de proyectos Manwaring, usando el modelo conceptual de la Figura 5.24:

```
CLIENTE (NOMBRE_CLIENTE, DIR_CLIENTE)
PROYECTO (#PROYECTO, NOMBRE_CLIENTE, TÍTULO_PROYECTO,
        TOTAL_CARGO, #FACTURA, FECHA_FACTURA)
Clave Externa: NOMBRE_CLIENTE REFERENCIA A CLIENTE
OTRO_CARGO (#CARGO, #PROYECTO, CANTIDAD, DESCRIPCIÓN)
Clave Externa: #PROYECTO REFERENCIA A PROYECTO
CONSULTOR (NOMBRE_CONSULTOR, PAGO)
OCUPADO_EN_ACTIVIDAD_EN (NOMBRE_CONSULTOR, ACTIVIDAD, #PROYECTO,
                    HORAS, CANTIDAD)
Claves Foráneas: NOMBRE_CONSULTOR REFERENCIA A CONSULTOR
            #PROYECTO REFERENCIA A PROYECTO
```

Ahora suponga que no tuvo acceso al modelo mostrado en la Figura 5.24 y que en realidad no conocía nada sobre el modelado de datos conceptual. Si fuera un usuario que buscaba por primera vez en el esquema del modelo relacional precedente, ¿ piensa que se entendería fácilmente el significado de la relación OCUPADO_EN_ACTIVIDAD_EN? Usando la referencia de clave externa tendría que encontrar primero el esquema relacional para la relación CONSULTOR. Entonces tendría que buscar el esquema relacional para la relación PROYECTO. Finalmente, usando el nombre de la relación OCUPADO_EN_ACTIVIDAD_EN y el hecho de que ACTIVIDAD es una columna clave en esta relación, deduciría prudentemente qué consultores se ocupan de múltiples actividades en diferentes proyectos y esta relación captura las horas y las cantidades cargadas.

Ahora mire la Figura 5.24. ¿Cuánto tiempo le llevaría deducir esa misma información? Claramente, sólo unos pocos segundos de análisis se necesitan debido a que la información se describe mucho más natural en el modelo gráfico. Este efecto natural se captura en el lenguaje natural a través del uso de sustantivos (correspondiendo a conjuntos de objetos) y verbos (correspondiendo a interrelaciones). En el modelo conceptual se ven sustantivos como cajas y verbos como líneas interrelacionadas conectando las cajas. Esta representación visual hace más fácil comprender la estructura.

En el modelo relacional, por otra parte, las únicas herramientas que se tienen disponibles son tablas relacionales y claves foráneas, además los conjuntos de objetos y las interrelaciones están todas modeladas en la misma forma. Mirando solamente los esquemas relacionales no está claro inmediatamente cuáles de ellos representan sustantivos (o conjuntos de objetos) y cuáles representan verbos (o interrelaciones).

Sin una ayuda gráfica se tiene que hacer la distinción sustantivo-verbo por nosotros mismos. ¡Verdaderamente, una imagen vale más que mil palabras! Realmente, analizando el modelo relacional, está dibujando mentalmente una imagen para sí mismo, que conecta

estas relaciones de la misma manera que los conjuntos de objetos se conectan en la Figura 5.24. Así, para realmente entender un modelo relacional, se hace mentalmente el trabajo que ya está hecho gráficamente en un modelo conceptual.

Ahora, si imagina que este modelo relacional se amplía, de manera que en lugar de incluir sólo cinco relaciones incluya 30, 40 ó 50, puede ver lo difícil que sería entenderlo sin una ayuda gráfica. En la creación de un esquema de base de datos es esencial que todas las sutilezas inherentes en una situación del mundo real sean capturados exactamente. Sin el uso de un modelo conceptual, el diseñador de base de datos corre el riesgo de cometer errores críticos de diseño.

Ahora se confronta la pregunta opuesta: ¿Por qué nos molestamos en convertirlo a un modelo relacional? ¿Por qué no solamente se implementa el modelo conceptual? La respuesta a estas preguntas es que la gran mayoría de las SGBDs instaladas son relacionales. Los SGBDs orientados a objetos, los cuales serían capaces de implementar directamente esquemas conceptuales, se considera que generalmente no han alcanzado la "industrialización" requerida para grandes aplicaciones y son por lo tanto opciones no reales. Además, para bases de datos simples, el modelo relacional es probablemente tan fácil de diseñar directamente como el modelo conceptual. Por tanto, ambas proposiciones son viables y pueden ser usadas apropiadamente.

Este capítulo introduce el modelo de datos relacional, las formas normales asociadas con él y un proceso para convertir un modelo conceptual a un modelo relacional. Se mostraron formas de diseñar una base de datos relacional directamente de la información del usuario o transformando un modelo conceptual. Además se discutió la ventaja intrínseca de usar un modelo conceptual antes de la creación del modelo relacional.

El modelo de datos relacional fue introducido en 1970 por E. F. Codd, quien mostró que un enfoque lógico de la definición y manipulación de datos es superior al enfoque físico que prevalecía en ese tiempo. El modelo relacional se basa en la noción matemática de una relación, que consiste en filas y columnas de datos. Las columnas se llaman atributos y las filas se llaman tuplas. Cada relación tiene un conjunto de atributos, conocidos como una clave, que identifican unívocamente cada fila. Las referencias de interrelaciones son manipuladas a través de claves foráneas, que son punteros lógicos de una fila de una relación a una fila de otra relación. Para estar seguros de la validación de los datos en una base de datos relacional, Codd formuló las reglas de integridad de la entidad y de integridad referencial. Estas reglas establecen que el valor de una clave no puede ser nulo y que el valor de una clave externa debe corresponder con el valor actual de una clave en otra relación.

Las relaciones que son diseñadas como parte de un diseño de base de datos conceptual pueden tener anomalías de actualización. Éstas normalmente ocurren debido a las dependencias funcionales o interrelaciones funcionales entre atributos, ninguno de los cuales es un atributo clave de la relación. Haciendo las dependencias funcionales una consecuencia natural de las definiciones de claves de una relación, se eliminan muchas anomalías comunes. Las relaciones que corresponden con esta regla están en tercera forma normal. Además, la existencia de atributos multievaluados y reglas diseñadas para garantizar la independencia entre atributos resultan necesarias para una forma normal superior. La cuarta forma normal se usa para garantizar que las relaciones en tercera forma normal manipularán probablemente atributos multievaluados. La quinta forma normal y la forma normal dominio/clave son formas normales adicionales que han sido discutidas en la literatura y son descritas brevemente en este capítulo.

El proceso de transformación de un modelo conceptual a un modelo relacional impli-

ca la creación de una relación para cada conjunto de objetos en este modelo. Los atributos del conjunto de objetos son atributos de la relación. Si existe un atributo clave externo, se puede usar como la clave de la relación. Por otra parte, una clave puede ser creada por el analista. Sin embargo, es mejor si un atributo surge espontáneamente en la aplicación que está siendo modelada. Interrelaciones de uno-uno y uno-muchos se convierten al modelo relacional haciéndolas atributos de la relación apropiada. Las interrelaciones muchos-muchos corresponden a atributos multievaluados y se convierten a la cuarta forma normal, creando una relación cuya clave se toma de las claves de los dos conjuntos de objetos que participan en la interrelación. Los conjuntos especializados se convierten creando relaciones separadas, que toman sus claves de la relación correspondiente al conjunto de generalización. Las relaciones recursivas pueden ser modeladas creando un nuevo nombre de atributo, que es descriptivo de la interrelación.

Los modelos de datos conceptuales son inherentemente más fáciles de comprender que los modelos relacionales, porque se ajustan más a la manera en que naturalmente se observan las cosas, y se reflejan en el lenguaje a través del uso de sustantivos y verbos. Esto muestra que el proceso de desarrollo de un modelo conceptual y la conversión a un modelo relacional normalizado merece el esfuerzo desarrollado porque en las situaciones de modelado complejas se pueden cometer errores críticos. Debido a que la implementación debe ser normalmente realizada en el modelo relacional, el paso de convertir el modelo conceptual al relacional continúa siendo esencial.

1. Defina cada uno de los siguientes términos con sus propias palabras:
   a. modelo de datos relacional
   b. interrelación recursiva
   c. grado de una relación
   d. dominio de atributo
   e. superclave
   f. determina funcionalmente
   g. clave candidata
   h. clave externa
   i. esquema de base de datos relacional
   j. regla de integridad de la entidad
   k. normalización
   l. redundancia de datos
   m. anomalías de actualización
   n. anomalías de inserción
   o. formas normales
   p. valor atómico
   q. determinante
   r. proyección de una relación
   s. forma normal Boyce-Codd
   t. dependencia multievaluada
   u. quinta forma normal
   v. relación de intersección

2. Compare y contraste:
   a. claves y superclaves
   b. claves foráneas y claves
   c. claves foráneas y claves foráneas recursivas.
   d. atributos y dominios

e. atributos y columnas
f. tuplas y filas
g. integridad de la entidad e integridad referencial
h. claves candidatas y claves primarias

3. Reaccione a la siguiente afirmación, parafraseada de Kent (1983): Una relación está en tercera forma normal si todo atributo no clave es dependiente de la clave, de la clave entera y de nada más que la clave. ¿Qué parte de la afirmación corresponde a la segunda forma normal y cuál a la tercera forma normal?

4. Explique por qué no es conveniente tener relaciones que estén en segunda o tercera forma normal.

5. Describa el proceso de transformación de un modelo conceptual a un modelo relacional para:
   a. conjunto de objetos y atributos con y sin una clave externa
   b. interrelación uno-uno
   c. interrelación uno-muchos
   d. interrelación muchos-muchos
   e. interrelaciones de especialización
   f. agregados
   g. interrelaciones recursivas

6. Compare y contraste estos dos enfoques para el modelado de datos: (1) Primero desarrollar un modelo conceptual y entonces convertirlo mecánicamente a un esquema relacional normalizado. (2) Saltar el paso del modelo conceptual y desarrollar un modelo relacional, usando las teorías de normalización para eliminar las anomalías. ¿Qué ventajas y desventajas tiene cada enfoque?

1. Marque cada término con su definición:

—*forma normal dominio/clave*	a. Un atributo no clave es funcionalmente dependiente de uno o más atributos no claves.
—*segunda forma normal*	b. Cada determinante es una clave.
—*atributos de la relación*	c. Valor de un atributo si el atributo es inaplicable o su valor es desconocido.
—*integridad referencial*	d. Consistencia de los datos en una base de datos.
—*tupla*	e. Tabla de dos dimensiones que contiene filas y columnas de datos.
—*valor nulo*	f. Conjunto de atributos mínimos que identifican unívocamente cada fila.
—*tercera forma normal*	g. En tercera forma normal sin dependencias multievaluadas.
—*clave primaria*	h. Clave externa que referencia su propia relación.
—*relación*	i. Una columna en una relación.
—*clave externa recursiva*	j. El valor de una clave externa no nula debe ser un valor actual de una clave en alguna relación.

—*restricción*	**k.** Clave que consiste de más de un atributo.
—*integridad de los datos*	**l.** División de una relación en múltiples relaciones.
—*clave compuesta*	**m.** Fila en una relación.
—*descomposición de relaciones*	**n.** El atributo no clave no puede ser funcionalmente dependiente de sólo una parte de la clave.
—*primera forma normal*	**o.** Todos los valores de los atributos deben ser atómicos.
—*dependencia funcional*	**p.** Pérdida no intencional de datos debido al borrado de otros datos.
—*anomalías de borrado*	**q.** Regla que restringe los valores en una base de datos.
—*cuarta forma normal*	**r.** Requiere que toda restricción sea resultado de la definición de dominios y claves.
—*clave*	**s.** Clave candidata designada por el uso principal en la identificación de filas.
—*dependencia transitiva*	**t.** El valor de un atributo en una tupla determina el valor de otro atributo en la tupla.

2. Considere la siguiente relación (las letras mayúsculas son nombres de atributos y las letras minúsculas y los números son valores):

X

A	*B*	*C*	*D*	*E*
a1	*b2*	*c1*	*d3*	*e2*
a3	*b2*	*c3*	*d2*	*e4*
a1	*b3*	*c1*	*d1*	*e2*
a2	*b4*	*c1*	*d4*	*e2*

Encierre en un círculo las dependencias funcionales que se aplican a *X*:

a. $A \rightarrow C$ b. $D \rightarrow E$ c. $C \rightarrow A$ d. $E \rightarrow B$
e. $E \rightarrow A$ f. $C \rightarrow B$ g. $B \rightarrow D$ h. $B \rightarrow A$

Identifique una posible clave para *X*.

3. Considere la siguiente relación (las letras mayúsculas son nombres de atributos y las letras minúsculas y los números son valores):

Y

A	*B*	*C*	*D*	*E*
a1	*b2*	*c1*	*d3*	*e2*
a2	*b2*	*c3*	*d3*	*e4*
a1	*b3*	*c2*	*d1*	*e4*
a2	*b4*	*c5*	*d1*	*e5*

Encierre en un círculo las dependencias funcionales que no se aplican a *Y*.

a. $A \rightarrow C$ b. $D \rightarrow E$ c. $C \rightarrow A$ d. $E \rightarrow B$
e. $E \rightarrow A$ f. $C \rightarrow B$ g. $B \rightarrow D$ h. $B \rightarrow A$

Identifique una posible clave para *Y*.

4. Considere la siguiente relación (las letras mayúsculas son nombres de atributos y las letras minúsculas y los números son valores):

Z

A	*B*	*C*	*D*	*E*
a1	*b2*	*c2*	*d3*	*e2*
a1	*b2*	*c2*	*d1*	*e4*
a2	*b3*	*c2*	*d1*	*e5*
a2	*b4*	*c5*	*d1*	*e5*

Circule las dependencias funcionales que se aplican a Z:

a. $E \rightarrow D$ b. $D \rightarrow E$ c. $C \rightarrow A$ d. $E \rightarrow B$
e. $E \rightarrow A$ f. $B \rightarrow C$ g. $B \rightarrow D$ h. $B \rightarrow A$

Identifique una posible clave para Z:

5. Para cada una de las siguientes relaciones, indique a qué formas normales se ajustan las relaciones (si hay alguna) y muestre cómo las relaciones pueden descomponerse en múltiples relaciones, cada una de las cuales se ajustan a la forma normal superior.

A. EMPLEADO (#SS, NOMBRE, DIRECCIÓN, TELÉFONO, PADRE, OFICIOS)
DF: DIRECCIÓN → TELÉFONO

B. TRABAJADOR (*ID_TRAB*, NOMBRE_T, #SS_ CÓNYUGE, NOMBRE_CÓNYUGE)
DF: #SS_CÓNYUGE → NOMBRE_CÓNYUGE)

C. VENTA (*FECHA*, *CLIENTE*, *PRODUCTO*, VENDEDOR, CIUDAD_VENDEDOR, VENTASREP)
DF: CLIENTE → VENTASREP

D. EMPLEADO (#SS, NOMBRE, DIRECCIÓN, TELÉFONO, PADRE, DIRECCIÓN_PADRE)
DF: PADRE → DIRECCIÓN_PADRE

E. TRABAJADOR (ID_TRAB, NOMBRE_TRAB, NOMBRE_CÓNYUGE, HIJOS)

F. VENTA (*FECHA*, *CLIENTE*, *PRODUCTO*, VENDEDOR, CIUDAD_VENDEDOR, VENTASREP)
DF: VENDEDOR → CIUDAD_VENDEDOR, DF: PRODUCTO ' VENDEDOR

G. ESTUDIANTE (*#EST*, NOMBRE, EDIFICIO, PISO, RESIDENCIA_EST)
DF: EDIFICIO, PISO ' RESIDENCIA_EST

H. MATRÍCULA (*#CURSO*, *#EST*, GRADO, INSTRUCTOR, #HABITACIÓN)
DF: #CURSO → INSTRUCTOR, DF: CURSO → #HABITACIÓN

I. ACTIVIDAD (*#EST*, *DESCRIPCIÓN*, FECHA, EDIFICIO, HABITACIÓN, COSTO)
DF: DESCRIPCIÓN → EDIFICIO, DF: DESCRIPCIÓN → HABITACIÓN,
DF: EDIFICIO → COSTO

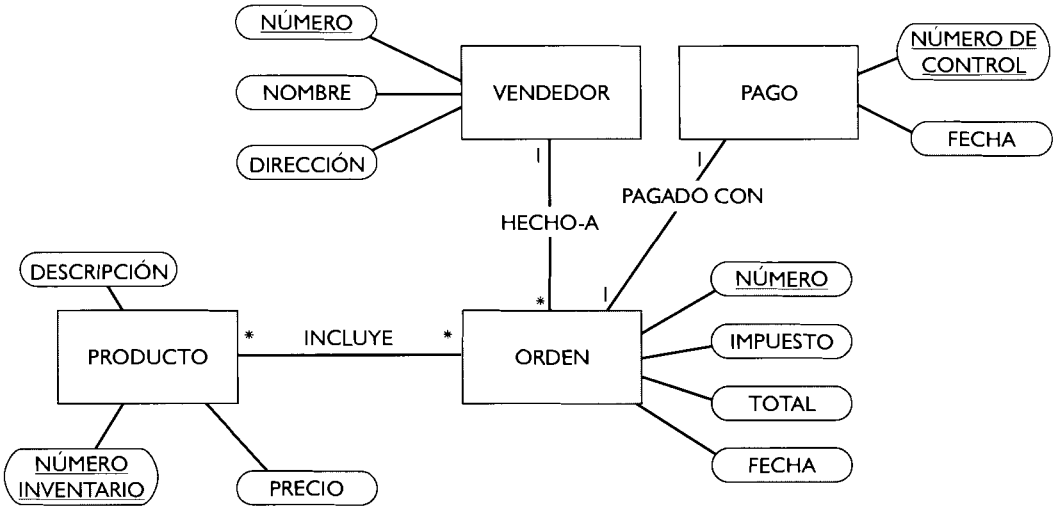
6. Cree un esquema relacional, con todas las relaciones en 4FN, para la siguiente información sobre una compañía de seguros de vida.

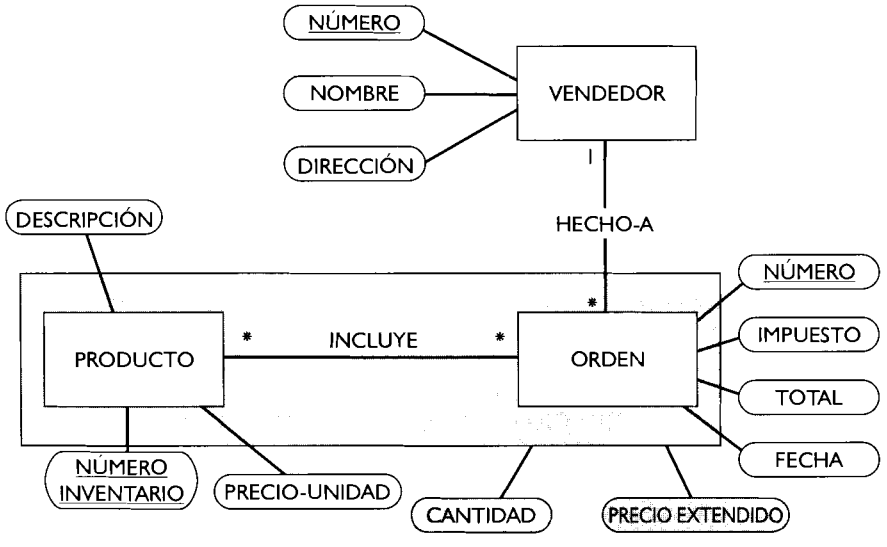
La compañía tiene una gran cantidad de pólizas, se quiere conocer el número de seguridad social del asegurado, el nombre, la dirección y la fecha de nacimiento.

También se necesita conocer el número de póliza, la prima anual y la cuenta del beneficiario en caso de muerte. Además, se quiere conocer el número del agente, el nombre y el lugar de residencia del agente que escribió la póliza. Un asegurado puede tener muchas pólizas y un agente puede escribir muchas pólizas.

7. Convierta los siguientes modelos conceptuales en esquemas relacionales que muestren nombres de relaciones, atributos, claves y claves foráneas.

   a. Figura 5.1E
   b. Figura 5.2E





1. Sin usar un modelo conceptual para el diseño conceptual, cree un esquema de base de datos relacional para una organización con la que tenga contacto o experiencia. Construya al menos ocho relaciones que estén en 4FN.

2. Realice el proyecto 1 nuevamente, esta vez primero diseñando un modelo conceptual y convirtiéndolo a un modelo relacional, usando los métodos de este capítulo. Compare sus experiencias en ambos proyectos.

PARTE

TRES

IMPLEMENTACIÓN DE BASE DE DATOS RELACIONALES

En la parte III se exploran los lenguajes y sistemas usados para la implementación de sistemas de base de datos en el modelo relacional. Esto incluye el álgebra y el cálculo relacional, así como también los lenguajes comerciales SQL y Query-by-Example. Además se examinan los principios relacionados con los problemas de implementación física.

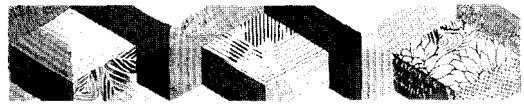
En la discusión del álgebra y el cálculo relacional, en el Capítulo 6 se examinan cuidadosamente estos lenguajes, que son las bases para todos los otros lenguajes relacionales. Se usan numerosos ejemplos para esclarecer ambos lenguajes.

El capítulo 7 examina el SQL, lenguaje relacional ANSI estándar. Además se presentan sus aspectos esenciales dados en el SQL-89 estándar y también se analizan un número importante de aspectos dados por el SQL-1992 estándar.

El capítulo 8 presenta el lenguaje Query-by-Example, un lenguaje relacional gráfico, el cual tiene varias implementaciones comerciales. En la discusión se examina una de estas implementaciones comerciales, PARADOX.

El capítulo 9 se acerca al nuevo área de sistemas cliente/servidor. Se ven dos sistemas de base de datos servidores, SQL server y ORACLE. También se analiza un entorno de desarrollo para aplicaciones cliente/servidor, PowerBuilder.

El capítulo 10 examina las técnicas de implementación física. Se examinan los dispositivos de almacenamiento de acceso-directo, los formatos de datos y la organización tradicional de archivos. Además se definen técnicas de correspondencia usando punteros, cadenas, anillos, listas invertidas y árboles B+ y se muestra cómo éstas pueden usarse para representar estructuras de datos lógicas. También se estudia la estructura de la clave secundaria y las técnicas de optimización de consultas.



CAPÍTULO

6

EL ÁLGEBRA Y EL CÁLCULO RELACIONAL

Una ventaja revolucionaria en la manipulación de datos
Álgebra relacional
- Unión
- Intersección
- Diferencia
- Producto
- Selección
- Protección
- Reunión (join) [1]
  - Reunión natural
  - Reunión theta
  - Reunión externa
- División
  - Asignación
  - Un ejemplo adicional

Cálculo relacional
- La lista resultado y la sentencia de cualificación.
- El cuantificador existencial
- El cuantificador universal

La dificultad relativa del álgebra relacional y el cálculo relacional.
Resumen
Preguntas de repaso
Problemas y ejercicios
Proyectos y cuestiones profesionales

Cordelia Molini y Reggie Townsend, usuarios de sistemas de información de la International Product Distribution (IPD), están discutiendo las diferencias en los lenguajes de bases de datos relacionales. Reggie ha sido introducido recientemente a las bases de datos relacionales y está esforzándose por comprender sus bases conceptuales. Sin embargo, no entiende las diferencias estructurales significativas entre algunos de los lenguajes comerciales. Cordelia le explica: "Cuando Codd originalmente propuso el modelo de datos relacional, recomendó un lenguaje procedimental, el álgebra relacional y un lenguaje no procedimental, el cálculo relacional."

"Estoy confundido, no entiendo esos términos, Cordelia. ¿Qué entiendes por procedimental y no procedimental?"

"En un lenguaje no procedimental, le decimos al computador qué tiene que hacer y no cómo hacerlo. En el lenguaje procedimental le decimos al computador cómo se ejecuta cada paso. Los lenguajes tradicionales de los computadores son procedimentales. El modelo relacional ha hecho más práctico el desarrollo de los lenguajes no procedimentales."

"Parecería entonces que los lenguajes no procedimentales son superiores a los lenguajes procedimentales. ¿Entonces por qué todos los lenguajes relacionales no son modelados a partir del cálculo relacional?"

"Los lenguajes más populares, tales como el SQL, el QBL y el QUEL, tienden a serlo. Para comprender completamente el modelo relacional y los lenguajes utilizados para manipularlo, es importante entender ambos enfoques."

Este capítulo brinda una amplia introducción al álgebra relacional y al cálculo relacional, los cuales forman las bases para los lenguajes comerciales usados con las bases de datos relacionales. Después de comenzar a familiarizarse con estos lenguajes, estará preparado para entender y usar algunos de los muchos lenguajes de base de datos que se basan en ellos. Después de leer este capítulo debe ser capaz de:

- Listar las operaciones del álgebra relacional y mostrar cómo pueden usarse para crear nuevas relaciones a partir de relaciones existentes.
- Demostrar la estructura de las soluciones de consulta en el cálculo relacional, más especialmente las instrucciones condicionales que deben formarse para definir una solución de consulta.
- Formular soluciones para tipos específicos de consultas de ambos lenguajes.

▼ Una ventaja revolucionaria en la manipulación de datos

En 1970-1971, E. F. Codd publicó dos artículos introduciendo el modelo de datos relacional y los lenguajes de manipulación de datos relacionales, el álgebra relacional y el cálculo relacional. Aunque el modelo de datos relacional era importante en sí mismo, los lenguajes relacionales eran más significativos en cuanto a la revolución de las bases de datos relacionales. Después de todo, el modelo relacional, en el cual los datos se representan en tablas, es muy similar a los modelos orientados a archivos que ya existían. Reconocidamente, los cambios en la terminología: de *archivo* a *relación*, de *campo* a *atributo* y otros se debieron a que ellos enfatizaban el significado lógico de los datos mucho más que sus estructuras físicas. Pero, retrospectivamente, parece que el aspecto más importante del nuevo modelo eran sus lenguajes de datos concretos, que permiten la manipulación de

datos únicamente basada en sus características lógicas. En este capítulo se estudiarán los dos lenguajes que Codd propuso, el álgebra relacional y el cálculo relacional.

álgebra relacional. Un lenguaje procedimental para la manipulación de relaciones.

En su artículo original, Codd introdujo el modelo de datos relacional y el álgebra relacional (Codd, 1970). El **álgebra relacional** es un lenguaje **procedimental** para la manipulación de relaciones. Esto significa que el álgebra relacional usa una aproximación paso-por-paso para crear una relación que contenga los datos que responden a la consulta. En el siguiente artículo, Codd introdujo el cálculo relacional (Codd, 1971a, 1971b). El **cálculo relacional** es **no procedimental**. En el cálculo relacional, una consulta se resuelve definiendo una relación en un simple paso.

procedimental. Lenguaje que proporciona un método paso-a-paso para la solución de problemas.

cálculo relacional. Un lenguaje no procedimental para la definición de soluciones a consultas.

Codd mostró que el álgebra relacional y el cálculo relacional son lógicamente equivalentes —un hecho de considerable importancia—. Eso significó que cualquier consulta que podría ser formulada en el cálculo relacional podría ser formulada en el álgebra relacional, y viceversa. Esto proporcionó una medida del poder lógico de un lenguaje de consulta. Si un lenguaje era al menos tan poderoso como el álgebra relacional, era llamado **relacionalmente completo**. Eso significa que una consulta que puede ser formulada en el álgebra relacional debe ser formulable en el lenguaje "relacionalmente completo". Por tanto, con el desarrollo de los lenguajes relacionales comerciales, su poder lógico se puede probar comparándolos con el álgebra relacional o el cálculo relacional. Si un lenguaje es menos poderoso que ellos, entonces habrá ciertas consultas que no podrán formularse en el lenguaje comercial.

no procedimental. Lenguaje que proporciona un medio de establecer *qué* se desea en lugar de *cómo* hacerlo.

El álgebra relacional es también importante porque aporta mucho del vocabulario y muchos de los conceptos básicos de manipulación de datos relacionales que se encuentran comúnmente en los lenguajes de bases de datos comerciales. Términos tales como *select*, *project*, *join* y *union compatible* se originan en el álgebra relacional. Además, algunos lenguajes de bases de datos comerciales se basan en el álgebra relacional.

El cálculo relacional es importante por dos razones: (1) Está basado en la lógica formal del cálculo de predicados, que es un método poderoso de determinación de la verdad de una instrucción a partir de la verdad de sus componentes. Consecuentemente, el cálculo relacional tiene una fundamentación lógica tan firme como cualquier lenguaje de programación existente. (2) Varios lenguajes relacionales comerciales están conceptualmente cercanos a él. Se estudiarán algunos de estos lenguajes en capítulos posteriores.

relacionalmente completo. Que tiene el mismo poder lógico que el álgebra o el cálculo.

Ambos, el álgebra relacional y el cálculo relacional, en la manera en que están formulados por Codd y en la que se analizan en este capítulo, son lenguajes teóricos. Esto significa que estamos interesados sólo en los aspectos conceptuales del álgebra y el cálculo relacional, no en las implementaciones específicas de ellos. Por tanto, será bastante libre e informal la definición y el uso de la sintaxis. Si éstos fueran realmente lenguajes comerciales, como los que se estudiarán posteriormente en este libro, se necesitaría ser más preciso. Para ilustrar los ejemplos a lo largo de este capítulo se usará la base de datos de la Figura 6.1. Esta base de datos se toma de la International Product Distribution, caso que es el introducido en el Capítulo 1.

▼ Álgebra relacional

Las operaciones del álgebra relacional manipulan relaciones. Esto significa que estas operaciones usan una o dos relaciones existentes para crear una nueva relación. Esta nueva relación puede entonces usarse como entrada para una nueva operación. Este poderoso concepto —la creación de una nueva relación a partir de relaciones existentes— hace posible una variedad infinita de manipulaciones sobre los datos. Eso también hace considerablemente más fácil la solución de las consultas, debido a que se puede experimentar con soluciones parciales hasta encontrar una proposición con la que se trabajará.

CLIENTE

ID_CLIENTE	*NOMB_CLIENTE*	*DIRECCIÓN*	*PAÍS*	*SALDO INICIAL*	*SALDO ACTUAL*
100	Watabe Bros	Box 241, Tokyo	Japón	45.551	52.113
101	Matlzl	Salzburg	Austria	75.314	77.200
105	Jefferson	B 918, Chicago	USA	49.333	57.811
110	Gómez	Santiago	Chile	27.400	35.414

VENDEDOR

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

PRODUCTO

ID_PRODUCTO	*DESC_PRODUCTO*	*ID_FABRICANTE*	*COSTO*	*PRECIO*
1035	abrigo	210	1,25	2,00
2241	lámpara de mesa	317	2,25	3,25
2249	lámpara de mesa	317	3,55	4,80
2518	escultura	253	0,60	1,20

VENTA

FECHA	*ID_CLIENTE*	*ID_VENDEDOR*	*ID_PROD*	*CANTIDAD*
28/02	100	10	2241	200
12/02	101	23	2518	300
15/02	101	23	1035	150
19/02	100	39	2518	200
02/02	101	23	1035	200
05/02	105	10	2241	100
22/02	110	37	2518	150
14/02	105	10	2249	50
01/02	101	23	2249	75
04/02	101	23	2241	250

FABRICANTE

ID_FABRICANTE	*NOMB_FABRICANTE*	*DIRECCIÓN*	*PAÍS*
210	Kiwi Klothes	Aukland	Nueva Zelanda
253	Brass Works	Lagos	Nigeria
317	Llama Llamps	Lima	Perú

El álgebra relacional consta de las siguientes nueve operaciones: unión, intersección, diferencia, producto, selección, proyectar, reunión, división y asignación. Las cuatro primeras de estas operaciones se toman de la teoría de conjuntos de la Matemática y que son considerablemente parecidas a las operaciones encontradas allí. Esto es razonable, debido a que las relaciones son en sí misma conjuntos, de esta manera se le pueden aplicar las operaciones de conjunto. Las cuatro siguientes son operaciones nuevas que se aplican específicamente al modelo de datos relacional. La última operación (asignación) es la operación estándar de los lenguajes de computación, de dar un valor a un nombre. En

este caso, la asignación se usa para dar un nombre a una nueva relación que se crea de relaciones existentes. Se usará el signo ":=" (es el nombre asignado a) para indicar la asignación de nombres a relaciones. Se discutirán estas operaciones en el orden dado anteriormente.

Unión

unión. Operación del álgebra relacional que crea un conjunto unión de dos relaciones unión-compatible.

La operación de **unión** ($\cup$) permite combinar los datos de dos relaciones. Por ejemplo, suponga que la relación VENDEDOR de la figura 6.1 no existe en la base de datos. En cambio, se tienen las dos relaciones de la Figura 6.2. Estas dos relaciones representan todos los vendedores que están subordinados a otros vendedores (VENDEDOR_SUBORDINADO) y todos los vendedores que son jefes de otros vendedores (VENDEDOR_JEFE). Obviamente, existe redundancia de datos. Esto habría sucedido a través de la ejecución de una serie de órdenes del álgebra relacional antes de ilustrar acerca de esta orden. Si se desea obtener una relación que contenga *todos* los vendedores, se debe realizar la *unión* de VENDEDOR_SUBORDINADO y VENDEDOR_JEFE.

```
VENDEDOR := VENDEDOR_SUBORDINADO VENDEDOR_JEFE
```

VENDEDOR (Figura 6.3) es el nombre asignado a la relación resultante y consiste en las filas que están en VENDEDOR_SUBORDINADO o VENDEDOR_JEFE, o ambas.

Es importante observar que una fila que existe en ambas relaciones aparece sólo una vez en la relación unión. Esto sigue la definición de una relación como un conjunto, debido a que un elemento dado se encuentra sólo una vez en un conjunto. Esto se ilustra en las Figuras 6.2 y 6.3. Tres filas, identificadas por ID_VENDEDOR 44, 35 y 12, se encuentran en ambas relaciones de la Figura 6.2, pero cada una de estas filas aparece sólo una vez en la relación de la Figura 6.3.

Se debe señalar un requisito especial de la operación de unión que la hace ligeramente diferente de la unión normal de conjuntos de matemáticas. En matemáticas, dos conjuntos cualesquiera pueden ser combinados a través de la operación unión. Pero en el álgebra relacional, antes de aplicar la unión a las dos relaciones, las relaciones deben tener exactamente las mismas columnas, tanto en número de columnas como en el dominio de

VENDEDOR_SUBORDINADO

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

VENDEDOR_JEFE

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
1035	Terry Cardon		Chicago	15
2241	Albert Ige	27	Tokyo	12
2249	Brigit Bovary	27	Brussels	11
2518	Buster Sánchez	27	B.A.	10

VENDEDOR

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

unión compatible. Dos o más relaciones que tienen columnas equivalentes en número y dominios.

las columnas. Si este es el caso, se dice que las dos relaciones son **unión compatible**. Obviamente, VENDEDOR_SUBORDINADO y VENDEDOR_JEFE son unión compatible.

Se requiere la compatibilidad de la unión para que el resultado de la unión sea una relación. Si se toma la unión de CLIENTE y PRODUCTO, se obtendría un conjunto, pero no una relación. Las filas en el conjunto resultante no tendrían columnas comunes, de este modo ellas no se agruparían en una tabla relacional. La compatibilidad de la unión es claramente esencial para la operación de unión. Por la misma razón, es esencial para las operaciones de intersección y diferencia.

Intersección

intersección. Operación del álgebra relacional que crea un conjunto intersección de dos relaciones unión-compatible.

La operación de **intersección** ($\cap$) permite identificar las filas que son comunes a dos relaciones. Por ejemplo, si se quiere identificar los vendedores que son subordinados a algún jefe y que son jefes, se toma la *intersección* de VENDEDOR_SUBORDINADO y VENDEDOR_JEFE, llamando al resultado VENDEDOR_SUBORDINADO_JEFE:

```
VENDEDOR_SUBORDINADO_JEFE := VENDEDOR_SUBORDINADO VENDEDOR_JEFE
```

Esto produce la siguiente relación:

VENDEDOR_SUBORDINADO_JEFE

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B. A.	10

El resultado de una operación de intersección es la relación consistente en todas las filas que están en ambas relaciones. Esto significa que, si C es la intersección de A y B,

$$C := A \cap B$$

entonces C consta de aquellas filas que están en A y en B. Al igual que antes, A y B deben ser unión compatible.

Diferencia

diferencia. Operación del álgebra relacional que crea un conjunto diferencia de dos relaciones unión-compatible.

La operación de **diferencia** (indicada por un signo menos) permite identificar filas que están en una relación y no en otra. Suponga que es de interés identificar los jefes que no son subordinados de otros jefes. Entonces, se toma la diferencia entre VENDEDOR_JEFE y VENDEDOR_SUBORDINADO, en ese orden.

```
VENDEDOR_JEFE_JEFE := VENDEDOR_JEFE - VENDEDOR_SUBORDINADO
```

Esto produce la siguiente relación:

VENDEDOR_JEFE_JEFE

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
27	Terry Cardon		Chicago	15

La diferencia entre dos relaciones se define como la relación consistente de todas las filas que están en la primera relación y no están en la segunda relación. Así, si

```
C := A - B
```

entonces una fila está en *C* si y sólo si está en *A* y no está en *B*.

Observe que

```
A - B
```

no es lo mismo que

```
B - A
```

Si se invierte el orden de las relaciones usadas en el ejemplo anterior,

```
VENDEDOR_SUBORDINADO - VENDEDOR_JEFE
```

la relación resultante consistiría de todos aquellos vendedores que no son jefes de nadie, lo cual es justamente lo opuesto de los vendedores que no son dirigidos por nadie. Así es que el orden de las relaciones en una operación de diferencia es muy importante. Una vez más, ambas relaciones deben ser unión compatible.

sustracción. La operación diferencia del álgebra relacional.

La operación de diferencia puede también ser llamada operación de **sustracción**. Esta operación particular es muy valiosa en la solución de algunos problemas difíciles, que de otro modo no se solucionarían. Se dará un ejemplo al final de la discusión del álgebra relacional.

Producto

producto. Operación del álgebra relacional que crea el producto cartesiano de dos relaciones.

La operación **producto**, que se indica por el símbolo *, es valiosa como un bloque para la construcción de una reunión (*join*), que es probablemente la operación más importante en el álgebra relacional. Es idéntica a la operación en matemáticas que crea el producto cartesiano de dos conjuntos. Ahora se explicará qué significa esto, ilustrándolo con la operación de producto en un ejemplo abstracto muy simple.

Considere las relaciones de la Figura 6.4(a). *A* y *B* son relaciones de dos columnas que tienen los atributos *X*, *Y* y *W*, *Z*, respectivamente. El producto de *A* y *B* es *C*, que se muestra en la Figura 6.4(b).

(a) La relación *A* y *B*

A

X	*Y*
10	22
11	25

B

W	*Z*
33	54
37	98
42	100

(b) El producto de *A* y *B*

C (:= A * B)

X	*Y*	*W*	*Z*
10	22	33	54
10	22	37	98
10	22	42	100
11	25	33	54
11	25	37	98
11	25	42	100

```
C := A * B
```

Observe que el producto se crea:

1. Concatenando los atributos de las dos relaciones.
2. Uniendo cada fila en *A*, con cada una de las filas en *B*.

Esto significa que los atributos de *C* son todos los atributos de *A* y *B* juntos. Debido a que *A* y *B* tienen dos atributos cada uno, *C* tiene cuatro atributos. Las filas de *C* se crean ensartando filas de *A* y *B* juntas. Cada fila de *A* se corresponde con cada fila de *B*. Así, debido a que hay tres filas en *B*, cada fila de *A* se corresponde tres veces con *B* y entonces aparece en tres diferentes filas de *C*. Es fácil ver que el número de filas de *C* siempre será el número de filas en *A por* el número de filas en *B*.

Otro ejemplo usando la base de datos de la Figura 6.1 es si tomamos el producto de PRODUCTO y VENTA.

```
P_V := PRODUCTO * VENTA
```

P_V tendría 10 columnas y 40 filas. Hay un pequeño problema en este caso, debido a que una columna en PRODUCTO y una columna en VENTA (ID_PRODUCTO) tienen el mismo nombre. Esto se resuelve modificando el nombre de la columna en cada caso, añadiendo el nombre de la relación original. Así, en P_V se tienen columnas llamadas PRODUCTO.ID_PRODUCTO y VENTA.ID_PRODUCTO.

Esto parece ser una aplicación no natural para la operación producto. Eso significa que no está claro qué tipo de consulta se responde tomando el producto de dos relaciones.

No obstante, como se verá, el producto se usa como la operación de construcción de un bloque para la reunión (*join*). Esto es conceptualmente importante; por tanto, tendrá una amplia aplicación más adelante en el capítulo. También se usa en el lenguaje de consulta SQL, que es el lenguaje comercial relacional más importante.

Selección

selección. Operación del álgebra relacional que usa una condición para seleccionar filas de una relación.

La operación de **selección** se usa para crear una relación a partir de otra relación, seleccionando sólo aquellas filas que satisfacen una condición específica. Por ejemplo, considere la siguiente consulta:

Consulta: *Dar toda la información de los vendedores en la oficina de Tokyo.*

Esto puede solucionarse seleccionando las filas de la relación VENDEDOR de la Figura 6.1, sujetas a la condición de que una fila se selecciona sólo si el atributo OFICINA es igual a "Tokyo". Esto se hace en el álgebra relacional como sigue:

```
VEND_TOKYO := SELECT (VENDEDOR: OFICINA = 'Tokyo'
```

El nombre "VEND_TOKYO" se ha definido para identificar la relación. La palabra clave "SELECT" se usa para indicar que se está realizando una operación de selección. A continuación de "SELECT" se pone entre paréntesis el nombre de la relación de la cual son seleccionadas las filas, seguido por dos puntos (:), seguido por una condición de selección. Esas filas que satisfacen la condición de selección serán seleccionadas y puestas en la relación resultante. El resultado de esta operación de selección es la siguiente relación:

VEND_TOKYO

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
14	Masaji Matsu	44	Tokyo	11
39	Goro Azuma	44	Tokyo	10
44	Albert Ige	27	Tokyo	12

Las condiciones de selección son esencialmente las mismas condiciones usadas en las instrucciones IF en los lenguajes tradicionales de programación. Sin embargo, los nombres de columnas usados en una condición de selección dada deben encontrarse en la relación nombrada en la operación de selección. Algunos ejemplos de condiciones de selección que podrían usarse con la relación VENDEDOR son:

```
ID_VENDEDOR = 23
NOMB_VENDEDOR = 'Brigit Bovary'
ID_JEFE >= 20
not OFICINA = 'B. A.'
%COMISIÓN < 11
```

Observe que pueden usarse operadores de comparación tales como "<" y ">". Hay cinco operadores de comparación: =, <, >,<=, >=. Para cada uno de éstos hay un operador correspondiente que usa el operador booleano "not". Así, se tiene "=" y "not =", "<" y "not <" y así sucesivamente. También se pueden usar los conectores booleanos "and"y "or". El operador "not" se puede usar para negar una condición entera. Estos conceptos se ilustran en las consultas siguientes.

Consulta: *¿Qué vendedores tienen ID 23?*

Solución: SELECT (VENDEDOR: ID_VENDEDOR = 23)

Resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
23	François Moire	35	Brussels	9

Consulta: *Dar toda la información sobre el vendedor Brigit Bovary*

Solución: SELECT (VENDEDOR: NOMB_VENDEDOR = 'Brigit Bovary')

Resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
35	Brigit Bovary	27	Brussels	11

Consulta: *¿Quiénes son los vendedores con un jefe con ID mayor o igual que 20?*

Solución: SELECT (VENDEDOR: ID_JEFE >= 20)

Resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
39	Goro Azuma	44	Tokyo	10
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B. A.	10

Consulta: *Dar toda la información sobre los vendedores, excepto aquellos de la oficina de Buenos Aires.*

Solución: SELECT (VENDEDOR: OFICINA = 'B. A.')

Resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
39	Goro Azuma	44	Tokyo	10
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11

Consulta: *¿Qué vendedores tienen una comisión menor que 11%?*

Solución: SELECT (VENDEDOR: % COMISIÓN < 11)

Resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
10	Rodney Jones	27	Chicago	10
23	François Moire	35	Brussels	9
39	Goro Azuma	44	Tokyo	10
12	Buster Sánchez	27	B. A.	10

Consulta: *¿Quiénes son los vendedores en la oficina de Tokyo que tienen una comisión mayor que 11%?*

Solución: SELECT (VENDEDOR: OFICINA = 'Tokyo' and %_COMISIÓN > 11)

Resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
14	Masaji Matsu	44	Tokyo	11
44	Albert Ige	27	Tokyo	12

Consulta: *¿Quiénes son los vendedores cuyo jefe tiene identificador 27 o una comisión mayor que 10%?*

Solución: SELECT (VENDEDOR: ID_JEFE = 27 and %_COMISIÓN > 10)

Resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*% COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
37	Elena Hermana	12	B. A.	13
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B. A.	10

Protección

protección. Operación del álgebra relacional que crea una relación borrando columnas de una relación existente.

proyección. Relación que resulta de una operación de proyectar.

Algunas de las consultas usadas para ilustrar la operación de selección preguntaban "Quiénes...", lo cual sugiere que los usuarios querían sólo los nombres de los vendedores que satisfacían la condición de la consulta. Sin embargo, la respuesta a cada consulta incluía filas enteras de datos tomadas de la relación VENDEDOR, debido a que la operación de selección siempre selecciona filas enteras. Está claro que se necesita alguna manera de eliminar las columnas no deseadas. Si la operación de selección puede pensarse como la eliminación de las filas no deseadas, la operación de **proyectar** puede pensarse como la eliminación de las columnas no deseadas. La relación resultante de una operación de proyectar se llama una **proyección** de la relación original.

A diferencia de otras operaciones del álgebra relacional, la operación de proyectar no requiere una palabra clave especial o símbolo. Más bien, para crear una proyección —una relación consistente sólo de ciertas columnas identificadas de otra relación— se lista simplemente la relación original seguida de las columnas que se quieren conservar encerradas entre corchetes. Por ejemplo, si se desea identificar los vendedores de la oficina de Tokyo, se proyectaría la columna nombre de la relación VENDEDOR_TOKYO mostrada en la página 182.

```
VENDEDOR_TOKYO [NOMB_VENDEDOR]
```

Esto da la siguiente relación:

NOMB_VENDEDOR
Masaji Matsu
Goro Azuma
Albert Ige

Se podría haber escogido más de una columna. Así, si se quiere el ID, el Nombre y el Jefe de esos vendedores se introduce:

```
VENDEDOR_TOKYO (ID_VENDEDOR, NOMB_VENDEDOR, ID_JEFE)
```

Esto da el siguiente resultado:

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*
14	Masaji Matsu	44
39	Goro Azuma	44
44	Albert Ige	27

Suponga que es de interés conocer todos los diferentes porcentajes de comisión pagados a los vendedores. Se puede obtener esto proyectando simplemente la columna %_COMISIÓN de la relación VENDEDOR:

```
VENDEDOR [% COMISIÓN]
```

Esto da el siguiente resultado:

% COMISIÓN
10
11
9
13
15
12

Observe que cada tasa de comisión aparece sólo una vez, aun cuando distintos vendedores tienen la misma tasa de comisión. Debido a que una relación es un conjunto, una tasa dada aparece sólo una vez. Esto es una característica importante de la operación de proyectar. Eso automáticamente elimina filas duplicadas de la operación resultante. Esto además ocurre cuando la relación resultante consiste de más de una columna. Si cualesquiera dos filas enteras en una relación son idénticas columna por columna, la fila aparece sólo una vez en la relación.

La operación de proyectar presenta una oportunidad muy conveniente para mostrar el anidamiento de operaciones en el álgebra relacional. Por *anidamiento* se entiende la ejecución de más de una operación sin asignar explícitamente un nombre a las relaciones intermedias resultantes. Por ejemplo, añadamos una operación de proyectar a una de las consultas usadas para ilustrar la operación de selección:

Consulta: *¿Quiénes son los vendedores que tienen una comisión menor que 11%?*

Solución: SELECT (VENDEDOR: % COMISIÓN < 11) [NOMB_VENDEDOR]

Resultado:

NOMB_VENDEDOR
Rodney Jones
François Moire
Goro Azuma
Buster Sánchez

En este ejemplo, la operación de selección se realiza primero seguida por la proyección sobre la columna NOMB_VENDEDOR resultante. Esto es permisible para anidar operaciones del álgebra relacional como se quiera, usando paréntesis donde se necesite mostrar el orden de las operaciones.

Reunión

reunión. Operación del álgebra relacional que conecta relaciones.

La operación de **reunión** (*join*) se usa para conectar datos a través de relaciones —quizás la función más importante en cualquier lenguaje de base de datos—. Existen varias versiones: la reunión natural (*natural join*), la reunión theta (*theta join*) y la reunión externa (*outer join*). De éstas, la reunión natural es la más importante.

Reunión natural. Considere la relación VENTA de la Figura 6.1. Esta relación almacena el cliente, el vendedor y el producto involucrado en una transacción de venta particular, incluyendo los IDs de estos tres elementos de datos. Esta información permite hacer conexiones lógicas entre las relaciones CLIENTE, VENDEDOR y PRODUCTO. Por ejemplo, suponga que se quieren conocer los nombres de los clientes que han hecho compras al vendedor 10. Primero, se seleccionan aquellas ventas aplicadas sólo al vendedor 10 y se ponen entonces en una relación que se llama VENTA_10. Esto da la siguiente relación:

VENTA_10

FECHA	*ID_CLIENTE*	*ID_VENDEDOR*	*ID_PRODUCTO*	*CANTIDAD*
28/02	100	10	2241	200
05/02	105	10	2241	100
14/02	105	10	2249	50

Entonces se puede obtener la información deseada *reuniendo* las relaciones VENTA_10 y CLIENTE. Esta operación procede como sigue:

1. Se crea el producto de VENTA_10 y CLIENTE. Se obtiene una relación con 11 columnas (5 de VENTA_10 y 6 de CLIENTE) y 12 filas (3 en VENTA_10 * 4 en CLIENTE).
2. Todas las filas de esta relación producto se eliminan, excepto aquellas en las cuales el ID_CLIENTE de VENTA_10 es igual al ID_CLIENTE de CLIENTE. El resultado es la relación mostrada en la Figura 6.5. Observe que hay dos columnas ID_CLIENTE en la relación y que en cada fila los valores en estas dos columnas son idénticos.
3. Debido a que las dos columnas ID_CLIENTE contienen idéntica información, una de ellas puede eliminarse. Esto resulta en **la reunión natural** (*natural join*) de VENTA_10 y CLIENTE mostrada en la Figura 6.6.

reunión natural. Operación de reunión que conecta relaciones cuando las columnas comunes tienen iguales valores.

Por supuesto, se ha obtenido mucha más información que sólo los nombres de los clientes. Si se quieren sólo los nombres se tendría que proyectar la columna NOMB_CLIENTE de la relación de la Figura 6.6.

FECHA	*ID_CLIENTE*	*ID_VENDEDOR*	*ID_PRODUCTO*	*CANTIDAD*
28/02	100	10	2241	200
05/02	105	10	2241	100
14/02	105	10	2249	50

ID_CLIENTE	*NOMB_CLIENTE*	*DIRECCIÓN*	*PAÍS*	*SALDO INICIAL*	*SALDO ACTUAL*
100	Watabe Bros	Box 241, Tokyo	Japón	45.551	52.113
105	Jefferson	B 918, Chicago	USA	49.333	57.811
105	Jefferson	B 918, Chicago	USA	49.333	57.811

FECHA	*ID_CLIENTE*	*ID_PRODUCTO*	*CANTIDAD*
28/02	10	2241	200
05/02	10	2241	100
14/02	10	2249	50

ID_CLIENTE	*NOMB_CLIENTE*	*DIRECCIÓN*	*PAÍS*	*SALDO INICIAL*	*SALDO ACTUAL*
100	Watabe Bros	Box 241, Tokyo	Japón	45.551	52.113
105	Jefferson	B 918, Chicago	USA	49.333	57.811
105	Jefferson	B 918, Chicago	USA	49.333	57.811

Quizás una vía más fácil para ver qué sucede en una reunión es simplemente mirar el proceso como una tabla de consulta (*table lookup*). Para cada fila en la relación VENTA_10, se buscan las filas en la relación CLIENTE que tengan el mismo valor para ID_CLIENTE. Debido a que ID_CLIENTE es la clave de CLIENTE y como la base de datos cumple la integridad referencial, habrá siempre exactamente una fila de ese tipo. Así, debido a que hay originalmente tres filas en la relación VENTA_10, reuniendo la relación CLIENTE a VENTA_10 se crea una relación que también tiene tres filas. Se ha expandido simplemente cada fila de VENTA_10, añadiendo toda la información disponible sobre el cliente involucrado en la venta.

La operación de reunión natural en este ejemplo se escribe como sigue:

```
JOIN (VENTA_10, CLIENTE)
```

La definición general de la *reunión natural* es como sigue: Se asume que se quiere tomar la reunión natural de dos relaciones, *A* y *B*, las cuales tienen las columnas *C1*,...,*Cn* en común. Entonces JOIN (*A*, *B*) se obtiene a través de estos tres pasos:

1. Tome el producto de *A* y *B*. La relación resultaute tendrá dos columnas para cada *C1*,...,*Cn*.
2. Elimine todas las filas del producto, excepto aquellas en las cuales los valores de las columnas C1,...,Cn en *A* son iguales, respectivamente, a los valores de esas columnas en *B*.
3. Proyecte una copia de las columnas *C1*,...,*Cn*.

Para la mayoría de los ejemplos, las dos relaciones que están siendo reunidas tendrán sólo una columna en comúu. Sin embargo, como muestra la definición general, si dos relaciones tienen más de una columna en común, entonces la reunión depende de la igualdad de los valores en *todas* las columnas comunes. Se indica que si *A* tiene *k* columnas y *B* tiene *m* columnas, entonces la reunión natural de *A* y *B* tendrá (*k* + *m* - *n*) columnas, donde *n* es el número de columnas que *A* y *B* tienen en común.

Ahora se ilustrará el uso y el poder de la reunión (*join*) con varias cousultas simples.

Consulta: *Adjunte la información de las ventas a la información sobre los vendedores.*

Solución: JOIN (VENDEDOR, VENTA)

Resultado: Ver Figura 6.7

ID_VENDEDOR	NOMB_VENDEDOR	ID_JEFE	OFICINA	%_COMISIÓN
10	Rodney Jones	27	Chicago	10
10	Rodney Jones	27	Chicago	10
10	Rodney Jones	27	Chicago	10
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10

FECHA	ID_CLIENTE	ID_PRODUCTO	CANTIDAD
28/02	100	2241	200
05/02	105	2241	100
14/02	105	2249	50
12/02	101	2518	300
15/02	101	1035	150
02/02	101	1035	200
01/02	101	2249	75
04/02	101	2241	250
22/02	110	2518	150
19/02	100	2518	200

Es esencialmente la misma consulta que en el ejemplo anterior, excepto que en este caso se ha usado la relación VENDEDOR en lugar de la relación CLIENTE. Se da este ejemplo para ilustrar otra forma de ver la reunión (*join*). En este caso, la consulta indicada comienza buscando en la relación VENDEDOR y pregunta qué información sobre las ventas está ligada a las filas de esta relación. Como varios vendedores pueden tener más de una venta y otros ninguna, nos encontramos que las filas para algunos vendedores están repetidas varias veces, mientras que las filas para otros no aparecen siempre. Muchas personas piensan que algo se pierde en un caso como éste, donde los vendedores que no tienen ventas en la base de datos no aparecen en el resultado de la reunión natural. La *reunión externa* (*outer join*), que se analiza posteriormente, se ha definido para tratar este problema.

Consulta: *¿Cómo se llama el cliente involucrado en cada venta?*

Solución:
```
A := CLIENTE [ID_CLIENTE, NOMBRE_CLIENTE]
B := JOIN (VENTA, A)
```

Resultado:
B

FECHA	ID_VENDEDOR	ID_PROD	CANTIDAD	ID_CLIENTE	NOMB_CLIENTE
28/02	10	2241	200	100	Watabe Bros
12/02	23	2241	100	105	Maltzl
15/02	23	2249	50	105	Maltzl
19/02	39	2518	300	101	Watabe Bros
02/02	23	1035	150	101	Maltzl
05/02	10	1035	200	101	Jefferson
22/02	37	2249	75	101	Gómez
14/02	10	2241	250	101	Jefferson
01/02	23	2518	150	110	Maltzl
04/02	23	2518	200	100	Maltzl

En este ejemplo no se proyectó la información no deseada de CLIENTE antes de hacer la rennión (*join*). La respuesta a la consulta se encuentra en la relación que se ha nombrado *B*. Observe que la columna ID_CLIENTE de la relación VENDEDOR se considera una columna redundante y se ha eliminado.

Consulta: *Dé los nombres de los clientes que han comprado el producto 2518*

```
Solución: A := SELECT (VENTA: ID_PROD = 2518)
          B := JOIN (A, CLIENTE) [NOMB_CLIENTE]
```

Resultado:

B

NOMB_CLIENTE

Maltzl
Watabe Bros
Gómez

La solución en este ejemplo primero crea una relación VENTA reducida, poniendo en la relación *A* sólo aquellas ventas que involucran el producto 2518. Entonces se hace la reunión de esta relación con la relación CLIENTE y se proyecta NOMB_CLIENTE, dejando el resultado deseado en la relación *B*.

La operación de selección podría haber sido especificada de segunda en lugar de primera. En ese caso la solución habría sido:

```
A := JOIN (VENTA, CLIENTE)
B := SELECT (A: ID_PROD = 2518) [NOMB_CLIENTE]
```

Esta solución es lógicamente equivalente a la primera solución. Eso significa que ambas soluciones dan el mismo resultado. Sin embargo, normalmente se prefiere la primera solución porque se ejecuta mucho más rápido. Esto es debido a que la relación *A* en la primera solución es mucho más pequeña que la relación VENTA. Por lo tanto se requerirían unas pocas comparaciones más para completar la rennión. Como regla, es más eficiente realizar operaciones tales como la reunión, la cual requiere numerosas comparaciones, *después* de operaciones que reducen el número de filas a comparar, tales como la selección. Por supuesto, esto es posible sólo si hay más de una solución válida lógicamente.

Consulta: *¿Quiénes han comprado lámparas de mesa?*

```
Solución: A := SELECT (PRODUCTO: DESCRIPCIÓN = 'lámpara de mesa')
          B := JOIN (A, VENTA)
          C := JOIN (B, CLIENTE) [NOMB_CLIENTE]
```

Resultado:

C

NOMB_CLIENTE

Watabe Bros
Jefferson
Maltzl

En este ejemplo se debe seguir un camino lógico desde "lámpara de mesa" en la relación PRODUCTO hasta NOMB_CLIENTE en la relación CLIENTE. Para hacer esto, se identifican todos los productos lámparas de mesa y se sigue entonces el camino desde PRO-

DUCTO a VENTA hasta CLIENTE. Esto requiere la reunión de tres relaciones. Se ha hecho esto ejecutando la operación de reunión dos veces. Como sólo es de interés obtener el nombre del cliente, en el paso final no se proyectan todas las otras columnas.

Consulta: *¿Qué vendedores han vendido productos manufacturados en Perú?*

```
Solución: A := SELECT (FABRICANTE: PAÍS = 'Perú': )
          B := JOIN (A, PRODUCTO)
          C := JOIN (B, VENTA)
          D := JOIN (C, VENDEDOR ) [NOMB_VENDEDOR]
```

Resultado:

D

NOMB_VENDEDOR
Rodney Jones
François Moire

Esta consulta es esencialmente la misma que la consulta anterior, sólo que el camino desde el país de fabricación al vendedor es una relación más larga. De este modo, se deben ejecutar tres reuniones para conectar cuatro relaciones. Una de las relaciones intermedias tendrá 16 columnas, pero debido a que los atributos de interés son país y nombre del vendedor, se pueden ignorar los otros atributos. La relación final *D* tendrá, sin embargo, sólo una columna simple, como se muestra en la solución anterior.

Reunión theta: Considere la siguiente consulta.

Consulta: *Identifique los vendedores cuyos jefes obtienen una tasa de comisión por encima del 11%*

Todos los datos necesarios para resolver esta consulta están contenidos en la relación VENDEDOR, debido a que contiene los datos sobre *todos* los vendedores —aquellos que son jefes y aquellos que no lo son—. Sin embargo, esta consulta no puede resolverse con una simple operación de selección, porque la tasa de comisión en los registros de los vendedores se aplican a los vendedores y no a los jefes de los vendedores. Para obtener la tasa de comisión de un vendedor jefe, se deben reunir los registros de los jefes en la relaciou VENDEDOR con los registros de vendedores. Esto significa que se debe reunir (*join*) la relación VENDEDOR con ella misma. Entonces, se tendrá la tasa de comisión del jefe en la misma fila qne el nombre del vendedor y puede completarse fácilmente la solución de la consulta.

Por ejemplo, la primera fila de VENDEDOR contiene información sobre Rodney Jones. Como se puede ver, el ID de su jefe es 27. Este es el ID_VENDEDOR para Terry Cardon, por tanto es el jefe de Rodney. Debido a que su tasa de comisión es el 15 por 100, el jefe de Rodney Jones recibe más del 11 por 100 de comisión y Rodney Jones es parte de la solución de la consulta. Haciendo la reunión de VENDEDOR con ella misma, se puede unir el registro del jefe a cada registro de vendedores y se puede usar una instrucción de selección en esta relación para resolver la consulta.

Pero ¿cómo hacer la reunión de una relación con ella misma? No se puede usar la reunión natural debido a que ésta se basa en la igualdad de *todas* las columnas comunes en las dos relaciones que están reuniéndose. Si ambas relaciones en la operación de reunión son la misma relación, entonces todas las columnas serán comunes a las dos relaciones y la reunión (*join*) carecería de sentido.

Para resolver este problema se define una nueva versión de la reunión. Esta versión permitirá especificar una condición para la reunión de filas. La siguiente solución ilustra esta uueva versión de reunión, después de la creación de dos copias de VENDEDOR:

```
VEND1 := VENDEDOR
VEND2 := VENDEDOR
    A := JOIN (VEND1, VEND2 : VEND1.ID_JEFE = VEND2.ID_VENDEDOR)
```

La última instrucción muestra la nueva versión de reunión. Los dos puntos seguidos de la condición declaran que las filas serán reunidas si el ID_JEFE de la primera es igual al ID_VENDEDOR de la segunda. En otras palabras, se están uniendo a las filas de los vendedores la fila que contiene información sobre su jefe. Esto se muestra en la Figura 6.8. Note que no existe una fila que muestre a Terry Cardon con su jefe. Esto se debe a que Terry Cardon no tiene jefe, tiene un valor nulo en la columna ID_JEFE. Sin embargo, su fila aparece como un *jefe* con muchos otros vendedores.

Aún no se ha terminado con esta consulta. Se debe todavía identificar aquellos vendedores cuyos jefes reciben una tasa de comisión que excede el 11%. Se puede terminar la consulta usando simplemente la operación de selección en la relación A:

```
B := SELECT (A: VEND2.%_COMISIÓN > 11) [VEND1.NOMB_VENDEDOR]
```

Esto produce la siguiente relación:

B

VEND1.NOMB_VENDEDOR
Rodney Jones
Masaji Matsu
Goro Azuma
Albert Ige
Brigit Bovary
Buster Sánchez

A

VEND1.ID_VENDEDOR	*VEND1.NOMB_VENDEDOR*	*VEND1.ID_JEFE*	*VEND1.OFICINA*	*VEND1.% COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B. A.	13
39	Goro Azuma	44	Tokyo	10
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

VEND2.ID_VENDEDOR	*VEND2.NOMB_VENDEDOR*	*VEND2.ID_JEFE*	*VEND2.OFICINA*	*VEND2.% COMISIÓN*
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	B uster Sánchez	27	B. A.	10
44	Albert Ige	27	Tokyo	12
27	Terry Cardon		Chicago	15
27	Terry Cardon		Chicago	15
27	Terry Cardon		Chicago	15

Observe que se *tuvo* que especificar VEND2.%_COMISIÓN y no VEND1.%_COMISIÓN. El primero contiene la tasa de comisión de los jefes, mientras que el último contiene la tasa de comisión de los vendedores.

reunión theta. Operación de reunión que conecta relaciones cuando los valores de determinadas columnas tienen una interrelación específica.

La **reunión theta** (*theta join*) es una reunión con una condición específica que involucra a una columna de cada relación. Esta condición especifica que las dos columnas deberían compararse de alguna forma. El operador de comparación puede ser cualquiera de los seis:

```
=, not =, <, >, <=, >=
```

La manera general de expresar esto es que la reunión theta tome la forma

```
JOIN (A, B: X δ Y)
```

donde A y B son las relaciones que serán reunidas, X y Y son las columnas de las dos relaciones y la letra griega δ es uno de los seis operadores de comparación relacionados anteriormente.

equirreunión. Reunión theta basada en la igualdad de columnas específicas.

El ejemplo ilustra la reunión theta, donde el operador de comparación es "=". Esta reunión también se llama **equirreunión** (equijoin). Las reuniones usando los otros operadores también son necesarias para algunos problemas. Posteriormente en este capítulo se ilustra un caso.

Como una nota final, se indica que la reunión theta no se parece a la reunión natural, porque no incluye la eliminación de una o más columnas como paso final. En otras palabras, si A tiene k columnas y B tiene m columnas, entonces la reunión theta de A y B tendrá $k + m$ columnas.

Reunión externa. Se verá nuevamente una consulta discutida anteriormente:

Consulta: *Adjunte la información de las ventas a la información sobre los vendedores.*

```
JOIN (VENDEDOR, VENTA)
```

El resultado de esto se mostró en la Figura 6.7.

Sin embargo, este resultado no es satisfactorio, porque la intención de esta consulta es mostrar *todos* los vendedores junto con sus ventas. Por tanto, no se puede usar la reunión natural si se desea listar los vendedores que no han tenido ventas junto con aquellos que han tenido. La reunión externa remedia este problema.

reunión externa. Expansión de la reunión natural que incluye *todas* las filas de ambas relaciones.

La **reunión externa** extiende la reunión natural, asegurándose que cada registro de ambas relaciones sea listado en la relación reunida (*join relation*) al menos una vez. La reunión externa consiste de dos pasos. Primero se ejecuta una reunión natural. Entonces si un registro en una relación no se corresponde con un registro de la otra relación en la reunión natural, ese registro se añade a la relación que ha sido reunida y las columnas adicionales se llenan con valores nulos. Esto se ilustra para el problema actual en la Figura 6.9. Esta relación es el resultado de la siguiente operación:

```
OUTERJOIN (VENDEDOR, VENTA)
```

Algunas versiones de SQL, tales como Oracle, permiten hacer una reunión externa. Aunque la necesidad de esto es menor, se ha visto que hay ocasiones en que esto es deseable.

ID_VENDEDOR	NOMB_VENDEDOR	ID_JEFE	OFICINA	%_COMISIÓN
10	Rodney Jones	27	Chicago	10
10	Rodney Jones	27	Chicago	10
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

FECHA	ID_CLIENTE	ID_PROD	CANTIDAD
28/02	100	2241	200
05/02	105	2241	100
14/02	105	2249	50
12/02	101	2518	300
15/02	101	1035	150
02/02	101	1035	200
01/02	101	2249	75
04/02	101	2241	250
22/02	110	2518	150
19/02	100	2518	200

División*

Suponga que se tiene una consulta como la siguiente:

Consulta: *Liste los vendedores que han vendido todos los productos.*

Nuestra simple base de datos contiene cuatro productos diferentes con ID: 1035, 2241, 2249 y 2518. Un vendedor satisface la consulta si él o ella han vendido cada uno de estos productos al menos una vez. En otras palabras, para cada uno de estos productos debe haber al menos uua fila en VENTA que contenga el ID_VENDEDOR de este vendedor. Una consulta como ésta puede solucionarse usando la operación **división** (*divide*) del álgebra relacional.

división. Operación del álgebra relacional que crea una nueva relación, seleccionando las filas en una relación que se corresponden con *todas* las filas en otra relación.

La palabra clave en esta consulta es *todos*, debido a que se requiere que para cada vendedor se examinen las filas de VENTA hasta que se haya encontrado si ese vendedor ha vendido *todos* los productos. El requisito aquí es diferente a aquellos descritos anteriormente, debido a que en aquellos casos se podía trabajar con una o dos filas al mismo tiempo en la realización de la operación. La operación de división requiere mirar una relación entera de una vez.

¿Cómo resolver esta consulta? Se seguirá un procedimiento cercano al que se usaría intuitivamente y se mostrará cómo la operación de división se corresponde con este procedimiento.

* Esta sección puede omitirse sin perder la continuidad.

Obviamente, si se quiere conocer si un vendedor ha vendido todos los productos, se debe obtener una relación que liste todos los productos. La relación PRODUCTO es tal relación. Sin embargo, en otras relaciones, un producto se identifica sólo por la columna *clave*, no por una fila completa de la relación PRODUCTO. Así, la clave se presenta como el producto. El primer paso entonces es obtener una relación consistente del atributo clave para todos los productos en la base de datos. Se hace esto proyectando PRODUCTO en ID_PROD:

```
PI := PRODUCTO [ID_PROD]
```

Así, PI es una relación que contiene todos los valores de ID_PROD.

Seguidamente se debe obtener una relación que contenga todas las instancias donde un vendedor y un producto están juntos en una única venta. Esto se hace claramente proyectando la relación VENTA en ID_PROD e ID_VENDEDOR:

```
PI_VI := VENTA [ID_PROD, ID_VENDEDOR]
```

Una instancia en PI_VI consiste en un ID_PROD y un ID_VENDEDOR y significa que el producto representado por ID_PROD fue vendido por el vendedor representado por ID_VENDEDOR. Por tanto, PI_VI consiste de todas las combinaciones producto/vendedor, donde el vendedor ha vendido el producto.

El resultado de estas dos proyecciones es como sigue:

PI

ID_PROD
1035
2241
2249
2518

PI_VI

ID_VENDEDOR	*ID_PROD*
10	2241
23	2518
23	1035
39	2518
37	2518
10	2249
23	2249
23	2241

Ahora simplemente se necesita determinar si cada vendedor representado en PI_VI se asocia con cada ID del producto en PI. Esto se hace automáticamente con la operación de división:

```
A := PI_VI / PI
```

con el resultado:

A

ID_VENDEDOR
23

Esto probablemente parece mágico. Sin embargo, no es así, debido a que se corresponde simplemente con la definición de división en el álgebra relacional. Seguidamente se

da una descripción general de la operación. Se asume que *A*, *B* y *C* son relaciones y se desea dividir *B* por *C*, dando *A* como resultado.

1. Las columnas de *C* deben ser un subconjunto de las columnas de *B*. Las columnas de *A* son todas y *sólo* aquellas columnas de *B* que *no* son columnas de *C*. Note que esto se corresponde con el ejemplo anterior. Las columnas de PI_VI son ID_PROD e ID_VENDEDOR, mientras que la columna de PI es ID_PROD y la columna de *A* es ID_VENDEDOR.
2. Una fila se encuentra en *A* si y sólo si está asociada en *B* con cada fila de *C*. Esto también se corresponde con el ejemplo, debido a que una fila de A (ID_VENDEDOR = 23) está asociada en PI_VI con todas las filas de PI y es además el *único* ID_VENDEDOR asociado.

La operación división es la opuesta de la operación producto. Es fácil verificar que si una relación es el producto de dos relaciones *B* y *C*, entonces se puede obtener *B*, dividiendo el producto por *C*. Esto es:

```
(B * C) / C = B
```

Esto explica, por la analogía con la aritmética corriente, por qué se ha llamado operación de división. Codd la incluyó en el álgebra relacional para proporcionar la capacidad necesaria para el cuantificador universal del cálculo relacional. Esto se discutirá posteriormente. En términos prácticos, la operación de división se proporcionó para que se pudieran solucionar consultas que involucraran "cada" o "todo" como parte de la condición. En el ejemplo que se ha discutido, la consulta fue:

```
Liste los vendedores que han vendido todos los productos
```

Como se ha visto, la operación de división fue esencial para la solución de esta consulta.

Asignación

asignación. Operación del álgebra relacional que da un nombre a una relación.

A lo largo de este capítulo se ha estado usando la operación de **asignación** para dar nombres a relaciones. Por ejemplo, en la instrucción

```
A := SELECT (VENDEDOR: %_COMISIÓN > 11)
```

el nombre *A* se asigna al resultado de la operación de selección. Este símbolo ":=" significa "es el nombre asignado a".

Un ejemplo adicional

El álgebra relacional es extraordinariamente poderosa en su flexibilidad y capacidad para solucionar un gran número de problemas. En esta sección se da un ejemplo de un problema que puede resolverse usando el álgebra relacional, aunque es posible no ver fácilmente la solución a primera vista. La solución implica el uso de operaciones del álgebra relacional en formas creativas.

Consulta: *¿Cuál es la máxima tasa de comisión?*

La dificultad inmediata con este problema es que parece que no se tiene una forma de comparar todos los valores en la columna tasa de comisión para determinar cuál es mayor. Es necesaria la comparación entre filas; por tanto, la operación de selección no

funcionará, debido a que ella se aplica al mismo tiempo a sólo una fila. Sin embargo, la reunión theta (*join theta*) permite al menos la comparación de dos filas. Se usará en la reunión de una relación con ella misma.

```
A := VENDEDOR [%_COMISIÓN]
B := A
C := JOIN (A, B: A.%_COMISIÓN > B.%_COMISIÓN)
```

El resultado de esta reunión es como sigue:

```
C

A.%_COMISIÓN    B.%_COMISIÓN
10               9
11              10
11               9
13              10
13              11
13               9
13              12
15              10
15              11
15               9
15              12
15              13
12              10
12              11
12               9
```

¿Cómo se puede usar esta relación para resolver el problema? Si se examinan las columnas separadamente. se descubre un hecho importante. La columna izquierda, *A*. %_COMISIÓN contiene todas las tasas de comisión, excepto la menor. y la columna derecha, *B*. %_COMISIÓN, contiene todas las tasas de comisión, excepto la mayor. Esto nos lleva a la solución del problema. *Restando* el conjunto de tasa de comisión en la columna derecha del conjunto de *todas* las tasas de comisión, se tiene la mayor tasa de comisión, como requiere la consulta.

```
D := C [B.%_COMISIÓN]
E := A - D
```

D es la columna derecha de *C* y además contiene todas las tasas de comisión excepto la mayor y *A* contiene todas las tasas de comisión. Su diferencia. *E*, contiene sólo la mayor tasa de comisión y es por tanto la solución a la consulta.

Debería ser claro obtener la tasa de comisión *mínima*, sólo es necesario sustituir la columna *A*. %_COMISIÓN por la columna *B*. %_ COMISIÓN en la definición de *D*. El resto de la solución es la misma.

La solución a esta consulta entraña dos "trucos": (1) reunir una relación con ella misma, usando la reunión theta y (2) sustraer del conjunto de todos los posibles valores una relación que contenga todos los valores, *excepto* el que pregunta la consulta. Este segundo punto es importante y a menudo necesario para la solución de las consultas más difíciles en el álgebra relacional.

▼ Cálculo relacional

El cálculo relacional usa un enfoque completamente diferente al álgebra relacional. No obstante, los dos lenguajes son lógicamente equivalentes. Esto significa que cualquier con-

sulta que pueda resolverse en un lenguaje puede resolverse en el otro. Se será más breve en el cálculo relacional, debido a que el lenguaje en sí mismo tiene menos construcciones.

Consulta: *¿Quiénes son los vendedores en la oficina de Tokyo?*

Esta consulta se resuelve en el cálculo relacional como sigue:

```
{r.NOMB_VENDEDOR : r IN VENDEDOR and r.OFICINA = 'Tokyo' }
```

Los paréntesis encerrando la instrucción indican que la solución de la consulta es un conjunto de valores. Precisamente lo que hay en este conjunto es lo que está descrito por la instrucción. La solución dada aquí ilustra la mayoría de las características del cálculo relacional. Se relacionan los componentes de la solución y se explican sus significados:

1. *r*
2. *r*.NOMB_VENDEDOR
3. los dos puntos (:)
4. *r* IN VENDEDOR
5. *r*.OFICINA = 'Tokyo'

1. *r* es una variable que indica una fila arbitraria. La relación de donde viene *r* se define por "*r* IN VENDEDOR", que significa que *r* es una fila de VENDEDOR. Se usarán letras minúsculas cercanas a *r* en el alfabeto, tales como *s*, *t*, *p* y *q*, como variables de filas.
2. *r*.NOMB_VENDEDOR es el valor del atributo NOMB_VENDEDOR en la fila *r*.
3. Los dos puntos (:) separan la lista resultado (*target list*) de la sentencia de cualificación (*qualifying statement*). La lista resnltado en este caso es:

   ```
   r.NOMB_VENDEDOR
   ```

   y la sentencia de cualificación es

   ```
   r IN VENDEDOR and r.OFICINA = 'Tokyo'
   ```

   Se explicará el significado de éstos brevemente. Los dos puntos ':'pueden leerse como "tal que".
4. "*r* IN VENDEDOR" se explicó en el punto 1.
5. "*r*. OFICINA = 'Tokyo'" significa que el valor del atributo OFICINA en la fila *r* es 'Tokyo'.

lista resultado. Una lista en una instrucción del cálculo relacional que define los atributos de la relación solución.

sentencia de calificación. Una condición en una instrucción del cálculo relacional que restringe los componentes en una relación solnción.

La lista resultado y sentencia de cualificación

La solución para toda consulta en el cálculo relacional es una relación que se define por una lista resultado y una sentencia de cualificación. La **lista resultado** define los atributos de la relación solución. La **sentencia de cualificación** es una condición usada para determinar qué valores de la base de datos actual van a la relación solución. Ahora explicaremos cómo trabaja esto.

En el ejemplo anterior, la lista resultado era *r*.NOMB_VENDEDOR. En otras palabras, la relación solución tiene sólo un atributo, el nombre del vendedor. Los valores actuales que van a la relación solución son aquellos valores de las filas que satisfacen la sentencia de cualificación. En este ejemplo, el nombre de un vendedor se toma de la fila *r* y se pone en la relación solución si la fila *r* satisface la condición

```
r IN VENDEDOR and r.OFICINA = 'Tokyo'
```

El sistema examina las filas de VENDEDOR, tal como se muestran en la Figura 6.1, una por una. Se le da a la primera fila el nombre de *r* temporalmente y la sentencia de cualificación se evalúa como verdadera o falsa. En este caso, debido a que *r*.OFICINA = 'Chicago', la sentencia de cualificación es falsa, por lo que *r*.NOMB_VENDEDOR (Rodney Jones) *no* es puesto en la relación solución. El sistema se mueve entonces a la segunda fila, dándole el nombre *r* y verificando la sentencia de cualificación nuevamente. Esta vez la sentencia es verdadera, por lo que se pone Masaji en la relación solución. Este proceso se repite para cada fila en VENDEDOR. El resultado es como sigue:

```
NOMB_VENDEDOR
Masaji Matsu
Goro Azuma
Albert Ige
```

Para las demás consultas, la lista resultado consistirá de un atributo simple. Sin embargo, la lista resultado puede consistir de múltiples atributos. Por ejemplo, considere la consulta:

Consulta: *Dé todos los atributos de los vendedores en la oficina de Tokyo.*

```
Solución: {r : r IN VENDEDOR and r.OFICINA = 'Tokyo' }
```

Aquí se ha indicado que todos los atributos serán incluidos simplemente listando *r*. Esto significa qne la fila entera debería ser incluida. También se podría lograr esto listando los atributos separados por comas:

```
{ (r .ID_VENDEDOR, r.NOMB_VENDEDOR, r.ID_JEFE, r.OFICINA,
r.%_COMISIÓN) : r IN VENDEDOR and r.OFICINA = 'Tokyo' }
```

Además, se puede mostrar la lista de cualquier subconjunto de estos atributos que se desee.

De la explicación anterior, sería fácil ver cómo las operaciones de selección y de proyectar del álgebra relacional están soportadas en el cálculo relacional. Las operaciones de unión, intersección, diferencia y producto pueden también ser fácilmente derivadas de las construcciones del cálculo relacional que se han discutido en este punto. Debido a que el cálculo no usa el procedimiento del álgebra paso a paso, no se necesita la sentencia de asignación. Por tanto, las únicas operaciones del álgebra relacional para las cuales aún no se han mostrado equivalencias en el cálculo relacional son la reunión y la división. Éstas requieren los *cuantificadores*: existencial para la reunión y universal para la división.

El cuantificador existencial

cuantificador existencial. Expresión del cálculo relacional que afirma la existencia de al menos una fila a la cual se le aplica la condición.

Un cuantificador *cuantifica* o indica la cantidad de algo. El **cuantificador existencial** indica que existe al menos una instancia de algo de un tipo particular. En el cálculo relacional, el cuantificador existencial se usa para indicar que existe una fila de un tipo particular en una relación.

Consideremos un ejemplo para aclarar este concepto:

Consulta: *Liste los nombres de los clientes que han comprado el producto 2518.*

Obviamente, la solución a esta consulta es una relación que contiene los nombres de ciertos clientes. Esto es una relación de una única columna. por tanto, la lista resultado es claramente

```
r.NOMB_CLIENTE
```

donde *r* es nna fila en CLIENTE.

Se tiene entonces la lista resnltado, pero ¿qué es una sentencia de cualificación? Para estar en la solución, el cliente debe satisfacer la condición de haber comprado el producto 2518. En otras palabras, si un ID del cliente se encuentra en una fila de VENTA con ID_PROD = 2518, entonces ese cliente está en la solución. Así, la condición debe ser que exista al menos nna fila en VENTA que contenga el ID del cliente y un ID_PROD de 2518. Se establece esto como sigue:

```
there exists s IN VENTA
(s.ID_CLIENTE = r.ID_CLIENTE and s.ID_PROD = 2518)
```

Esto se lee como: "Existe una fila *s* en VENTA, tal que *s*.ID_CLIENTE = = *r*.ID_CLIENTE y *s*.ID_PROD = 2518". (Las palabras *there exists* constituyen el cuantificador existencial.)

Observe que esta es nna instrucción sobre la fila *r*. Si es verdadera para un valor dado *r* tal que exista una fila *s*, entonces *r*.NOMB_CLIENTE es puesta en la relación solución. Si la instrucción es falsa —esto es, si no existe tal *s* para esta *r*—, entonces *r*.NOMB_CLIENTE no es puesta en la relación solución.

La solución completa del cálculo relacional para esta consulta es:

```
{ r.NOMB_CLIENTE : r IN CLIENTE and there exists s IN VENTA
                   (s.ID_CLIENTE = r.ID_CLIENTE
                   and s.ID_PROD = 2518) }
```

La solución describe una relación que consiste en una única columna y contiene nombres de clientes tomados de las filas de la relación CLIENTE. Un nombre dado es pnesto en la relación solución si su fila (*r*) satisface la condición que sigue los dos puntos. Se verán unas cuantas filas en CLIENTE para ver cómo se aplicaría la condición.

Considere la Figura 6.10. La primera fila de CLIENTE (que se ha llamado *r*) tiene ID_CLIENTE = 100. El NOMB_CLIENTE (Watabe Bros) será puesto en la relación solución si existe una fila en VENTA que tenga ID_CLIENTE = 100 e ID_PROD = 2518. Supongamos que existe tal fila y se ha marcado con *s*. Por tanto, *r* satisface la sentencia de cualificación y *r*.NOMB_CLIENTE se pone en la solución. Se repite este proceso para cada fila de CLIENTE. Cuando se designa la segunda fila *r*, entonces se debe encontrar una correspondiente *s* en VENTA. En este caso, la correspondiente *s* es la segunda fila de VENTA. Por tanto, Maltzl se pone en la relación solución. Como se procede más adelante, vemos que el cliente 105 (Jefferson) *no* es puesto en la solución, pero el cliente 110 (Gómez) sí. La solución del conjunto sería:

```
NOMB_CLIENTE
Watabe Bros
Maltzl
Gómez
```

En el álgebra relacional, la solución de esta consulta involucraría la reunión (*join*). Así, se ha mostrado cómo el cuantificador existencial se usa en el cálculo relacional para lograr la función de la reunión. Como un ejemplo final, veamos una consulta más compleja que requiere dos reuniones.

CLIENTE

ID_CLIENTE	*NOMB_CLIENTE*	*DIRECCIÓN*	*PAÍS*	*SALDO INICIAL*	*SALDO ACTUAL*
100	Watabe Bros	Box 241, Tokyo	Japón	45.551	52.113
101	Matlzl	Salzburg	Austria	75.314	77.200
105	Jefferson	B 918, Chicago	USA	49.333	57.811
110	Gómez	Santiago	Chile	27.400	35.414

VENTA

FECHA	*ID_CLIENTE*	*ID_VENDEDOR*	*ID_PROD*	*CANTIDAD*
28/02	100	10	2241	200
12/02	101	23	2518	300
15/02	101	23	1035	150
19/02	100	39	2518	200
02/02	101	23	1035	200
05/02	105	10	2241	100
22/02	110	37	2518	150
14/02	105	10	2249	50
01/02	101	23	2249	75
04/02	101	23	2241	250

Consulta: *¿Quiénes han comprado lámparas de mesa?*

Esta consulta se usó para ilustrar la reunión en la discusión del álgebra relacional. La relación solución se da en la página 189. La solución en el cálculo relacional es:

```
{ r.NOMB_CLIENTE : r IN CLIENTE and there exists s IN VENTA and
                   there exists t IN PRODUCT
                   (r.ID_CLIENTE = 's.ID_CLINTE and
                   s.ID_PROD = t.ID_PROD and
                   t.DESCRIPCIÓN = 'lámpara de mesa') }
```

Observe precisamente cómo la solución del álgebra relacional requirió dos reuniones, la solución del cálculo relacional requiere cuantificar existencialmente dos variables filas, *s* y *t*. Un ejemplo de un conjunto de valores para *r*, *s* y *t* se muestra en la Figura 6.11. Este conjunto ejemplo muestra que Jefferson se incluyó en la relación solución porque existen filas en VENTA y en PRODUCTO que prueban que Jefferson ha comprado lámparas de mesa.

El cuantificador universal *

cuantificador universal. Expresión del cálculo relacional que establece que una cierta condición se le aplica a *toda* fila de algún tipo.

El **cuantificador universal** indica que una condición se aplica a *todas* o a *cada* fila de algún tipo. Se usa para brindar la misma capacidad que la operación de división del álgebra relacional. Se ilustra su uso usando la misma consulta que se utilizó para la división.

Consulta: *Liste los vendedores que han vendido todos los productos.*

Observe que la condición para seleccionar un vendedor incluye la palabra *todos*. Sólo los vendedores que han vendido todos los tipos de productos son incluidos en la relación solución. Si observa el resultado siguiente, puede ver fácilmente que sólo un vendedor satisface la condición de la consulta.

* Esta sección puede omitirse sin perder la continuidad.

CLIENTE

ID_CLIENTE	*NOMB_CLIENTE*	*DIRECCIÓN*	*PAÍS*	*SALDO INICIAL*	*SALDO ACTUAL*
100	Watabe Bros	Box 241, Tokyo	Japón	45.551	52.113
101	Matlzl	Salzburg	Austria	75.314	77.200
105	Jefferson	B 918, Chicago	USA	49.333	57.811
110	Gómez	Santiago	Chile	27.400	35.414

r

VENTA

FECHA	*ID_CLIENTE*	*ID_VENDEDOR*	*ID_PROD*	*CANTIDAD*
28/02	100	10	2241	200
12/02	101	23	2518	300
15/02	101	23	1035	150
19/02	100	39	2518	200
02/02	101	23	1035	200
05/02	105	10	2241	100
22/02	110	37	2518	150
14/02	105	10	2249	50
01/02	101	23	2249	75
04/02	101	23	2241	250

s

PRODUCTO

ID_PROD	*DESCRIPCIÓN*	*ID_FABRICANTE*	*COSTO*	*PRECIO*
1035	abrigo	210	1,25	2,00
2241	lámpara de mesa	317	2,25	3,25
2249	lámpara de mesa	317	3,55	4,80
2518	escultura	253	0,60	1,20

t

La solución del cálculo relacional para esta consulta es como sigue:

```
{ r.NOMB_VENDEDOR : r IN VENDEDOR and for every p IN PRODUCTO
                    there exists s IN VENTA
                    (r.ID_VENDEDOR = s.ID_VENDEDOR and
                    s.ID_PROD = p.ID_PROD) }
```

El resultado es:

NOMB_VENDEDOR

François Moire

Un nombre del vendedor de una fila *r* en VENDEDOR se pone en la relación solución si la sentencia de cualificación es verdadera para la fila *r*. La Figura 6.12 muestra cómo la sentencia de cualificación es verdadera cuando *r* es la fila que contiene los datos sobre François Moire. Para cada fila *p* en PRODUCTO, debe haber una fila *s* en VENTA que satisface la condición. La figura muestra la correspondencia entre las filas de PRODUCTO y las filas de VENTA en la satisfacción de la condición. Esto es,

s(1) se corresponde con *p*(1)
s(2) se corresponde con *p*(2)
s(3) se corresponde con *p*(3)
s(4) se corresponde con *p*(4)

VENDEDOR

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

PRODUCTO

ID_PROD	*DESCRIPCIÓN*	*ID_FABRICANTE*	*COSTO*	*PRECIO*	
1035	abrigo	210	1,25	2,00	p(1)
2241	lámpara de mesa	317	2,25	3,25	p(2)
2249	lámpara de mesa	317	3,55	4,80	p(3)
2518	escultura	253	0,60	1,20	p(4)

VENTA

FECHA	*ID_CLIENTE*	*ID_VENDEDOR*	*ID_PROD*	*CANTIDAD*	
28/02	100	10	2241	200	
12/02	101	23	2518	300	s(4)
15/02	101	23	1035	150	s(1)
19/02	100	39	2518	200	
02/02	101	23	1035	200	
05/02	105	10	2241	100	
22/02	110	37	2518	150	
14/02	105	10	2249	50	
01/02	101	23	2249	75	s(3)
04/02	101	23	2241	250	s(2)

Para cada una de estas filas de VENTA, el ID_VENDEDOR es François Moire (23) y cada una de las cuatro filas de ventas se corresponde con uno de los cuatro posibles productos. Por tanto, Moire ha vendido *todos* los productos.

▼ La dificultad relativa del álgebra relacional y el cálculo relacional

La sabiduría convencional en los lenguajes de base de datos sostiene que los lenguajes no procedimentales deben ser más fáciles de usar que los lenguajes procedimentales. Sin embargo, esto no ha sido corroborado por la práctica. El resultado de un número de experimentos (Welty y Stemple, 1981; Hansen, 1987, 1988) indican que las personas generalmente realizan la solución de los problemas mucho mejor con un lenguaje procedimental que con un lenguaje no procedimental. Esto es particularmente cierto cuando los problemas son lógicamente más complejos. En el caso específico del álgebra relacional frente al cálculo relacional, los usuarios se inclinan a encontrar el cuantificador universal del cálculo relacional difícil de comprender. Consecuentemente, ellos a menudo son incapaces de resolver las consultas que requiere este cuantificador. Aunque la operación correspondiente en el álgebra relacional —división— es difícil de dominar, el porcentaje de personas

que la usan con éxito es mucho mayor que el porcentaje de personas que aprenden a usar el cuantificador universal exitosamente.

Nuestro punto de vista sobre el asunto es que aún no se ha desarrollado una buena alternativa para la solución de las consultas más difíciles que involucran "every" *(todo)* en la condición. Como se discutirá en el Capítulo 7, el enfoque usado por el lenguaje SQL, el predicado NOT EXITS, no parece ser mejor que las alternativas que brindan el álgebra y el cálculo. Con optimismo, la investigación futura fortalecerá los lenguajes de bases de datos en este área.

En este capítulo se ha presentado el álgebra relacional y el cálculo relacional de Codd. Estos dos lenguajes teóricos manipulan relaciones en una base de datos relacional basándose en sus características lógicas y sin considerar la estructura física usada para la implementación. El álgebra relacional es un lenguaje procedimental que usa soluciones paso-a-paso para los problemas de las consultas. Sin embargo, el cálculo relacional es no procedimental, lo que significa que la solución a la consulta se formula más bien como una definición del resultado deseado que como un proceso que producirá el resultado deseado. Codd mostró que el álgebra relacional y el cálculo relacional son lógicamente equivalentes o, en otras palabras, que cualquier consulta que pueda resolverse en un lenguaje puede resolverse en el otro.

El álgebra relacional consiste de nueve operaciones: unión, intersección, diferencia, producto, selección, proyectar, reunión, división y asignación. La unión, la intersección, la sustracción y el producto son muy similares a las operaciones de conjunto de los mismos nombres. La selección se usa para aplicar una condición a una relación y obtener una nueva relación que consiste de aquellas filas que satisfacen la condición. La operación de proyectar crea una nueva relación, moviendo columnas a partir de una relación existente. La operación de reunión conecta relaciones a través de columnas que contienen información similar. La división identifica las filas en una relación que se corresponde con cada fila en la otra relación. La asignación da un nombre a una relación.

El cálculo relacional define la solución de una consulta como un conjunto relacional. La definición de la relación consiste en una lista resultado, definiendo los atributos en la relación solución, y una sentencia de cualificación, que es una condición que los elementos de la lista resultado deben satisfacer. El cálculo relacional toma su nombre del cálculo de predicados en la lógica simbólica y usa los conectores booleanos (and, or, not) para enlazar condiciones que pueden ser verdaderas o falsas. Además usa los cuantificadores existencial y universal para indicar, respectivamente, que *existe* una instancia de algún tipo, o que una condición es verdadera para *cada* instancia de un tipo específico.

Aunque la sabiduría popular sostiene que es más fácil para las personas usar un lenguaje no procedimental que un lenguaje procedimental, las investigaciones indican que en el caso de los lenguajes relacionales esto puede que no siempre sea cierto. Para algunas consultas difíciles, el enfoque paso-por-paso de un lenguaje procedimental, como el álgebra relacional, ofrece la flexibilidad necesaria para la formulación de las soluciones de las consultas. Además las investigaciones en este área quizás resulten en el desarrollo de lenguajes que los usuarios puedan fácilmente aplicar a la solución de consultas que tengan un extenso rango de complejidad.

Las preguntas marcadas con un asterisco (*) están relacionadas con las materias opcionales del capítulo.

1. Defina con sus propias palabras cada uno de los siguientes términos:
   a. álgebra relacional
   b. no procedimental

c. completo relacionalmente
d. unión compatible
e. intersección
f. sustracción
g. proyectar
h. reunión natural
i. reunión theta
j. sentencia de cualificación
k. división
l. cuantificador universal

2. Describa las circunstancias en las que usaría cada una de las operaciones siguientes del álgebra relacional:
   a. selección
   b. proyectar
   c. reunir
   d. asignación
   e. sustracción
   f. intersección
   g. división

3. Explique la función de cada uno de los siguientes términos en la solución de consultas en el cálculo relacional:
   a. lista resultado
   b. sentencia de cualificación
   c. conectores booleanos
   d. cuantificador existencial
   e. cuantificador universal

4. ¿Por qué el cálculo relacional no necesita algo similar a la instrucción de asignación del álgebra relacional?

5. Considere la siguiente sentencia: "Los lenguajes no procedimentales son más fáciles de usar por los usuarios novatos". Usando su experiencia, analice esta frase.

6. Estudie el significado de decir que el álgebra relacional y el cálculo relacional son completos relacionalmente y qué significa eso para la evaluación de los SGBDs comerciales.

1. Enlace cada término con su definición.

—*diferencia*	a. Dar un nombre a una relación.
—*unión*	b. La reunión theta basada en la igualdad de columnas específicas.
—*procedimental*	c. Lista que define los atributos de una relación solución.
—*cuantificador existencial*	d. Lenguaje que brinda un método paso-por-paso para la solución de problemas.
—*selección*	e. Operación que conecta relaciones.
—*reunión (join)*	f. Relación que resulta de una operación de proyectar.
—*producto*	g. Expansión de la reunión natural que incluye *todas* las filas de ambas relaciones.

—*equirreunión* (equijoin)	h. Crea el producto cartesiano de dos relaciones.
—*reunión externa* (outer join)	i. Crea el conjunto diferencia de dos relaciones unión-compatible.
—*cálculo relacional*	j. Afirma la existencia de al menos una fila a la cual se le aplica una condición.
—*asignación*	k. Crea el conjunto unión de dos relaciones unión-compatible.
—*proyección*	l. Operación del álgebra relacional que usa una condición para mostrar las filas de una relación.
—*lista resultado*	m. Un lenguaje no procedimental para la definición de soluciones a consultas.

2. Usando el siguiente esquema relacional, indique cuáles operaciones del álgebra relacional podrían usarse para responder las consultas dadas:

```
CLIENTE (ID_CLIENTE, NOMB_CLIENTE, RENTA_ANUAL)
EMBARQUE (EMBARQUE_#, ID_CLIENTE, PESO, CAMIÓN_#, DESTINO)
```

a. ¿Qué clientes tienen una renta anual que excede los cinco millones?
b. ¿Cuál es el nombre del cliente 433?
c. ¿Cuál es la ciudad destino del transporte #3244?
d. ¿Qué camiones han transportado paquetes con un peso por encima de las 100 libras?
e. ¿Cuáles son los nombres de los clientes que han enviado paquetes a la ciudad de Sioux, Iowa?
f. ¿A qué destinos han enviado paquetes los clientes con renta anual inferior a un millón?

Use este esquema relacional para el siguiente conjunto de consultas:

```
CLIENTE (ID_CLIENTE, NOMB_CLIENTE, RENTA_ANUAL)
EMBARQUE (EMBARQUE_#, ID_CLIENTE, PESO, CAMIÓN_#, DESTINO)
Clave Foránea: DESTINO REFERENCIA A NOMB_CIUDAD EN CIUDAD
CAMIÓN (CAMIÓN_#, NOMB_CHÓFER)
CIUDAD (NOMB_CIUDAD, POBLACIÓN)
```

3. Dé soluciones del álgebra relacional a las siguientes consultas:
a. Una lista con los números de los embarques que pesan más de 20 libras
b. Los nombres de los clientes con más de 10 millones de renta anual
c. El chofer del camión #45
d. Los nombres de las ciudades que han recibido envíos que pesan más de 100 libras
e. Los nombres y la renta anual de los clientes que han enviado paquetes que pesan más de 100 libras
f. El número de los camiones que han transportado embarques que pesan más de 100 libras
g. Los nombres de los chóferes que han distribuido envíos que pesan más de 100 libras
h. Las ciudades que han recibido embarques de clientes que tienen una renta anual por encima de los 15 millones
i. Los clientes que tienen una renta anual por encima de los 15 millones que han enviado paquetes con peso menor de una libra
j. Los clientes que tienen una renta anual por encima de los 15 millones que han enviado embarques con peso menor de una libra o han enviado embarques a la ciudad de San Francisco
k. Los clientes cuyos envíos han sido distribuidos por el chófer Jensen

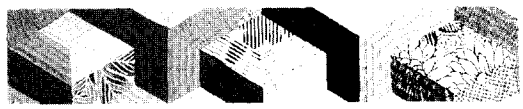
l. Los conductores que han distribuido envíos de clientes con renta anual por encima de los 20 millones a ciudades con población superior a un millón
m. Clientes que han tenido envíos distribuidos por todos los chóferes
n. Ciudades que han recibido embarques de todos los clientes
o. Chóferes que han distribuido envíos a todas las ciudades
p. Clientes que han enviado embarques a toda ciudad con población mayor de 500.000. (Sugerencia: Primero cree un conjunto de ciudades con población mayor de 500.000)
q. Dar una lista de los clientes y la renta anual para esos clientes cuya renta anual es el máximo para todos los clientes de la base de datos
r. Dé una lista de los clientes que envían todos sus embarques a una única ciudad (Nota: La ciudad puede o no puede ser la misma para cada uno de estos clientes.) (Sugerencia: Primero encuentre los clientes que envían embarques a más de una ciudad)

4. Dé soluciones del cálculo relacional para las consultas del problema 3.

1. Compare sus soluciones a las consultas en los problemas 3 y 4 de la sección anterior. ¿Cuáles consultas le parecieron más fáciles de resolver en el álgebra relacional y cuáles en el cálculo relacional? En cada caso, ¿por qué supone que un lenguaje fue más fácil de usar que otro? ¿Cuál de los dos lenguajes prefiere? ¿Por qué?

2. Lea los artículos de Codd del álgebra y el cálculo relacional (Codd 1970, 1971a, 1971b). Escriba un ensayo comparando los dos lenguajes y analice cómo Codd mostró su equivalencia lógica.

3. Escriba esbozos de programas en lenguajes tales como COBOL o Pascal para resolver las consultas 3 a, e, m y r. Compare la complejidad de estos programas con las soluciones a las consultas en el álgebra relacional y el cálculo relacional.



CAPÍTULO

7

IMPLEMENTACIÓN RELACIONAL CON SQL

Implementación relacional: introducción
Definición de tablas y de esquemas
  Definición de esquemas
  Tipos de datos y dominios
    Tipos de datos
    Definición de dominios
  Definición de tablas
    Definición de columnas
    Otras instrucciones de manipulación de esquemas
Manipulación de datos
  Consultas simples
    SELECT
    FROM
    WHERE
  Consultas multi-tablas
  Subconsultas
    Subconsultas correlacionadas
  EXISTS y NOT EXISTS
  Funciones integradas
  GROUP BY y HAVING
  Funciones integradas con subconsultas
  Operaciones del álgebra relacional
    UNION, INTERSECT y EXCEPT
    Operadores
    El operador JOIN
  Operaciones de modificación de la base de datos
    INSERT
    UPDATE
    DELETE
  Usar SQL con lenguajes de procesamiento de datos
Definición de vistas
  Restricciones sobre consultas y actualizaciones sobre vistas
El esquema de información
Resumen
Preguntas de repaso
Problemas y ejercicios
Proyectos y cuestiones profesionales

Tony Melton y Annette Chang, usuarios de los sistemas de información de la Compañía Constructora Premier, están discutiendo sobre su sistema de gestión de base de datos relacional instalado recientemente.

"Tony, ¿qué ventajas se tienen con SQL como lenguaje de nuestro sistema en lugar de alguno de los otros lenguajes de bases de datos relacionales que hay disponibles?"

"Probablemente, la ventaja principal, Anette, es que SQL es el lenguaje relacional estándar de ANSI y a su vez es el estándar de facto *en los negocios. Esto significa que está ampliamente soportado y que nos podemos sentir confiados de minimizar nuestros riesgos si cambiamos de hardware o de fabricante de software."*

"Pero ¿qué hay con el lenguaje en sí mismo? ¿Cómo se erige SQL como lenguaje relacional por derecho propio?

"SQL es un lenguaje poderoso. Tiene toda la potencia lógica del cálculo relacional de Codd, a la vez que tiene capacidades adicionales para manejar grupos de filas y aplicarle funciones estadísticas a éstos. Aún más, podemos definir nuestros esquemas de bases de datos, identificar las claves y las columnas no nulas con valores por defecto, y escribir instrucciones SQL en programas escritos en otros lenguajes. Además de todo esto, nuestro esquema de sistema de información que contiene la información de definición de todos los esquemas en una base de datos es en sí mismo una base de datos relacional a la que podemos consultar con SQL. Quizás lo más importante de todo es que el estándar SQL-92 añade una serie de mejoras al lenguaje que los fabricantes están ansiosos de implementar. A través de SQL podremos verdaderamente sacar ventaja del poder de la revolución relacional."

En este capítulo se estudiarán las partes de los sistemas de gestión de bases de datos relacionales que tienen que ver con el lenguaje SQL, incluyendo tópicos del estándar ANSI SQL-92. Después de leer este capítulo se estará capacitado para:

- Explicar los hechos básicos del desarrollo histórico de los sistema de gestión de bases de datos relacionales.
- Definir un esquema relacional de base de datos en SQL.
- Formular consultas SQL de variada complejidad.
- Insertar, actualizar y borrar datos de una base de datos relacional a través de órdenes SQL.
- Estudiar algunos aspectos de cómo escribir instrucciones SQL en un lenguaje de programación tradicional o en un lenguaje "anfitrión".
- Definir y consultar vistas de datos en SQL.
- Explicar algunos elementos básicos de la estructura de un esquema de información SQL.

▼ Implementación relacional: introducción

El efecto de la publicación de los artículos de Codd en los que se introdujo el modelo relacional y los lenguajes relacionales (álgebra y cálculo) fue el gran impacto que significó en las comunidades comerciales y de investigación para desarrollar versiones implementadas de los lenguajes relacionales. Los tres lenguajes más importantes que resultaron de este esfuerzo son probablemente SQL[1] (que se pronuncia "ess-cue-ell" o "sequel"), QBE

[1] Del inglés *Structured Query Language* (lenguaje de consulta estructurado) (N. del T.).

(Query-by-Example)[2], QUEL (Query Language)[3]. SQL y QBE fueron creados por IBM durante los setenta y realizan funciones muy similares, aunque SQL es un lenguaje textual mientras que QBE es gráfico. QUEL es el lenguaje original de INGRES, un sistema de gestión de base de datos relacional desarrollado en los setenta por la Universidad de Berkeley, en California.

SQL fue el resultado del proyecto de investigación System R de IBM. Este proyecto incluyó el desarrollo de un sistema de base de datos relacional y el lenguaje SEQUEL (Structured English Query Language). Al final de los setenta, SQL (cambio del SEQUEL) pasó a formar parte del dominio público y estuvo primero disponible como lenguaje para un sistema comercial de Oracle Corporation. En 1981 IBM lanzó su SQL/DS, un sistema de gestión de base de datos comercial que soporta SQL. En 1983 IBM lanzó SQL como parte del SGBD DB2.

En 1986 fue aprobado el primer estándar ANSI para SQL. Este estándar fue revisado más tarde en 1989 (modestamente) y en 1992 (significativamente). SQL ha quedado como el único lenguaje relacional de base de datos que es ANSI estándar. Es más, SQL es de hecho el estándar en los negocios, porque es el lenguaje relacional escogido por los sistemas comerciales. Desde 1980 numerosos fabricantes han lanzado implementaciones de SQL.

Adicionalmente a las versiones para grandes computadores de SQL mencionadas anteriormente, hay actualmente muchas versiones de SQL para computadores personales y cliente/servidor. Éstas incluyen Sybase SQL Server, Microsoft SQL Server, IBM OS/2 Extended Edition Database Manager, DEC Rdb/VMS y Oracle Server para OS/2 (Khoshafian y otros) para sistemas cliente/servidor, y XDB y SQLBase, así como versiones de R:Base y dBASE, para computadores personales.

El sistema INGRES se desarrolló originalmente como un sistema de gestión de base de datos con su lenguaje propio, QUEL, el que, al igual que SQL, modelaba el cálculo relacional. En los últimos años, INGRES ha sido expandido para soportar tanto SQL como QUEL. Esto es importante, ya que SQL ha sido adoptado como el estándar ANSI.

Los sistemas de gestión de bases de datos relacionales soportan una variedad de aspectos, además de sus capacidades lingüísticas. Estos aspectos incluyen seguridad, integridad, alto rendimiento en acceso y actualización de los datos y control del diccionario de datos o esquema de información. Algunos de estos aspectos se tratarán en este capítulo, otros serán analizados más adelante (ver, por ejemplo, los capítulos 11 y 12). En este capítulo se estudiará el lenguaje SQL y los aspectos del sistema de información de un SGBD relacional. En el Capítulo 9 se estudiarán los aspectos del lenguaje de dos SGBD relacionales cliente/servidor.

Aunque el nombre SQL sugiere que éste es un lenguaje de "consulta", además de las facilidades de consulta, incluye la definición de tablas, la actualización de la base de datos, definición de vistas, otorgamiento de privilegios. En este capítulo serán estudiados las capacidades de definición de tablas, de consultas, de actualización y de definición de vistas que hay en SQL, en este orden. Se incluirán aspectos del estándar ANSI de 1992, conocido como SQL-92. SQL-92 es una mejora significativa de las primeras versiones del estándar SQL y es demasiado grande para poder cubrirlo en este capítulo o incluso en este libro. (Para una exposición más completa, ver Melton y Simon, 1993 o Date y Darwen, 1994.) No obstante, se mostrarán varias de sus características, concentrándonos en las que se consideran más importantes. También se examinarán sus características de esquema de información. Para ilustrar los ejemplos se utilizará una base de datos relacional tomada del caso de la *Compañía Constructora Premier*. El ejemplo de base de datos se muestra en la Figura 7.1.

---

[2] Consultando por ejemplos.

[3] Lenguaje de consulta.

TRABAJADOR

ID_TRABAJADOR	*NOMB_TRABAJADOR*	*TARIFA_HR*	*OFICIO*	*ID_SUPV*
1235	M Faraday	12,50	Electricista	1311
1412	C Nemo	13,75	Fontanero	1520
2920	R Garret	10,00	Albañil	2920
3231	P Mason	17,40	Carpintero	3231
1520	H Rickover	11,75	Fontanero	1520
1311	C Coulomb	15,50	Electricista	1311
3001	J Barrister	8,20	Carpintero	3231

ASIGNACIÓN

ID_TRABAJADOR	*ID_EDIFICIO*	*FECHA_INICIO*	*NÚM_DÍAS*
1235	312	10/10	5
1412	312	01/10	10
1235	515	17/10	22
2920	460	05/10	18
1412	460	08/12	18
2920	435	28/10	10
2920	210	10/11	15
3231	111	10/10	8
1412	435	15/10	15
1412	515	05/11	8
3231	312	24/10	20
1520	515	09/10	14
1311	435	08/10	12
1412	210	15/11	12
1412	111	01/12	4
3001	111	08/10	14
1311	460	23/10	24
1520	312	30/10	17
3001	210	27/10	14

EDIFICIO

ID_EDIFICIO	*DIR_EDIFICIO*	*TIPO*	*NIVEL_CALIDAD*	*CATEGORÍA*
312	123 Elm	Oficina	2	2
435	456 Maple	Comercio	1	1
515	789 Oak	Residencia	3	1
210	1011 Birch	Oficina	3	1
111	1213 Aspen	Oficina	4	1
460	1415 Beech	Almacén	3	3

▼ Definición de tablas y esquemas

catálogo. En SQL-92, una colección de esquemas con nombre.

SQL-92, a diferencia de versiones anteriores, permite a los usuarios definir esquemas múltiples. Los esquemas múltiples se pueden agrupar en **catálogos** (*catalogs*), que en SQL-92 son colecciones de esquemas a las que se les ha dado un nombre. Cada catálogo contiene un esquema especial, llamado INFORMATION_SCHEMA, que contiene metadatos y que se estudiará posteriormente en este capítulo. Todos los restantes esquemas en un catálogo los define el usuario. Para mayor claridad debe notarse que el término *catálogo*, tal y como se usa en SQL-92, es derivado de la forma estándar y tradicional en que se usa el término en bases de datos. Un catálogo se concibe normalmente como conteniendo los metadatos

que definen la base de datos. En SQL-92, el esquema de información lleva a cabo esta función.

Definición de esquemas

SQL-92 denomina catálogo a una colección de esquemas que tiene nombre, pero no indica cómo debería definirse un catálogo. Esto se deja a la implementación del SGBD. Sin embargo, SQL-92 sí especifica que es una definición de esquema. Definir un esquema en SQL es elemental. Sólo se necesita identificar el comienzo de la definición con una instrucción CREATE SCHEMA (*crear esquema*) y una cláusula adicional AUTHORIZATION (*autorización*) y a continuación definir cada dominio, tabla, vista y demás en el esquema, como se muestra a continuación:

```
CREATE SCHEMA CONSTRUCTORA_PREMIER
   AUTHORIZATION TONY_MELTON
     definición de dominios
     definición de tablas
     definición de vistas
     etc.
```

Cada instrucción CREATE SCHEMA indica al SGBD que lo que sigue es un esquema de la base de datos. El nombre del esquema —CONSTRUCTORA_PREMIER en el ejemplo anterior— también se indica en la instrucción. La cláusula AUTHORIZATION indica el nombre del **propietario** del esquema. Esta persona se conoce entonces por el sistema y puede otorgar a otros usuarios privilegios de acceso y actualización de la base de datos definida en el esquema. Obviamente esta estructura implica que pueden existir en la misma instalación muchos esquemas de bases de datos. Estos pueden ser posesión de individuos diferentes, pero bajo el control del SGBD cada esquema también pudiera ser accedido por usuarios que no sean sus dueños.

propietario de un esquema. Persona que tiene autoridad y responsabilidad para otorgar acceso a tablas, columnas y vistas en un esquema de base de datos.

Tipos de datos y dominios

Antes de mostrar cómo se definen las tablas, veamos primero cómo se definen o predefinen los dominios de los cuales toman valores las columnas.

En el sentido del modelo relacional, como ha sido discutido en el Capítulo 5, un dominio es un conjunto del cual toma sus valores una columna de una relación. En este sentido, los tipos de datos predefinidos son dominios. Sin embargo, SQL define los dominios de forma ligeramente diferente, será analizada brevemente. En cualquier caso, SQL proporciona tipos de datos predefinidos y permite dominios definidos por el usuario. Según la definición de SQL, tanto los tipos de datos predefinidos como los dominios definidos por el usuario son dominios en el sentido del modelo relacional.

Tipos de datos. SQL-92 define los siguientes tipos de datos:

Numéricos exactos:

- Integer (*enteros*)
- Small integer (enteros pequeños)
- Numeric (*p*,*e*)(numéricos)
- Decimal (*p*,*e*)

Para los dos últimos tipos de datos (numérico y decimal) se indica una precisión (*p*) y una escala (*e*). La precisión indica el total de números o dígitos en el número y la escala indica cuántos de éstos están a la derecha del punto decimal.

Numéricos aproximados:

- Real
- Double precision (doble precisión)
- Float (flotante)

Estos tipos de datos se usan normalmente para cálculos científicos y de ingeniería.

Cadenas de caracteres:

- Character (*n*) (carácter)
- Character varying (*n*) (carácter variable)

Los campos de *character* siempre almacenan *n* caracteres, aun cuando tengan que rellenar con blancos a la derecha para completar la longitud *n*. Los campos *character varying* sólo almacenan el número real de caracteres que se introdujeron (hasta un máximo de *n*).

Cadenas de bits:

- Bit (*n*)
- Bit varying (*n*)

Estos campos se usan para banderas u otras máscaras de bits para el control.

Fechas y horas:

- Date (*fecha*)
- Time (*hora*)
- Timestamp (*sello de tiempo*)
- Time con tiempo zonal
- Timestamp con tiempo zonal

El tipo *Date* (*fecha*) se da en el orden año, mes, día con cuatro dígitos para el año. El *Time* se da en horas (0 a 23), minutos, segundos y décimas de segundo. El *Timestamp* es la fecha más la hora (*date plus time*).

restricción. (*constraint*) Una regla que restringe los valores en una base de datos.

Intervalos:

- Year-month (*año-mes*)
- Day-time (*día-hora*)

Un intervalo es la diferencia entre dos fechas (año-mes) o entre dos horas (día-hora). Por ejemplo, entre diciembre 1994 y enero 1996, el intervalo es un año y un mes.

valor por defecto. Un valor que se inserta automáticamente si el usuario no lo especifica en una entrada.

definición de dominio. Un tipo de datos especializado definido dentro de un esquema y usado en las definiciones de columnas.

Definición de dominio. Los tipos de datos con **restricciones** (*constraints*) y **valores por defecto** (*default values*) se pueden combinar en la definición de dominios. Una **definición de dominio** es un tipo de datos especializado que puede estar definido dentro de un esquema y utilizado en la definición de columnas. Por ejemplo, supongamos que se desea definir un dominio de identificadores para usar en la definición de columnas como ID_TRABAJADOR e ID_EDIFICIO. Anticipamos que esta definición será compleja, pues involucra un tipo de dato, un valor por defecto y una restricción no nula. Puesto que se usará ésta una y otra vez en el esquema de la base de datos, se quiere simplificar el trabajo. Por tanto, se crea un dominio de la forma siguiente:

```
CREATE DOMAIN IDENTIFICADOR NUMERIC (4) DEFAULT 0
   CHECK (VALUE IS NOT NULL)
```

Esta definición dice que un dominio llamado IDENTIFICADOR tiene las siguientes propiedades: Su tipo de datos es numérico de cuatro dígitos, su valor por defecto es cero y nunca puede ser nulo. Una columna que defina este dominio como su tipo de dato tendrá estas propiedades. Obsérvese que no se puede decir simplemente "NOT NULL" en la definición. SQL requiere lo que se denomina una restricción CHECK para lograr esto. Habiendo hecho esto se pueden definir columnas en el esquema que tengan IDENTIFICADOR como su tipo de dato.

Definiendo tablas

Las tablas se definen en tres pasos:

1. Dar el nombre de la tabla.
2. Definir cada columna, posiblemente incluyendo restricciones de columna.
3. Definir las restricciones de la tabla.

A continuación se da una definición de esquema para la base de datos de la Figura 7.1.

```
CREATE SCHEMA CONSTRUCTORA_PREMIER
   AUTHORIZATION TONY_MELTON
   domain definitions

CREATE TABLE TRABAJADOR (
   ID_TRABAJADOR      IDENTIFICADOR     PRIMARY KEY,
   NOMB_TRABAJDOR     CHARACTER (12),
   TARIFA_HR          NUMERIC (5, 2),
   OFICIO             CHARACTER (8),
   ID_SUPV            NUMERIC (4),
   FOREIGN KEY ID_SUPV REFERENCES TRABAJADOR
                           ON DELETE SET NULL )

CREATE TABLE ASIGNACIÓN (
   ID_TRABAJADOR      IDENTIFICADOR,
   ID_EDIFICIO        IDENTIFICADOR,
   FECHA_INICIO       DATE,
   NÚM_DÍAS           INTERVAL DAY (3),
   PRIMARY KEY (ID_TRABAJADOR, ID_EDIFICIO),
   FOREIGN KEY ID_TRABAJDOR REFERENCES TRABAJADOR
                           ON DELETE CASCADE,
   FOREIGN KEY ID_EDIFICIO REFERENCES EDIFICIO
                           ON DELETE CASCADE)

CREATE TABLE EDIFICIO (
   ID_EDIFICIO        IDENTIFICADOR     PRIMARY KEY,
   DIR_EDIFICIO       CHAR (12),
   TIPO   CHAR (9) DEFAULT 'Oficina'
     CHECK (TIPO IN ('Oficina', 'Almacén', 'Comercio', 'Residencia')),
   NIVEL_CALIDAD      NUMERIC (1),
   CATEGORÍA          NUMERIC (1) DEFAULT 1
     CHECK (CATEGORÍA > 0 AND CATEGORÍA < 4) )
```

Después del CREATE SCHEMA (crear esquema) y de posiblemente otras instrucciones como CREATE DOMAIN (crear dominio) están las instrucciones de CREATE TABLE (crear tabla). La instrucción CREATE TABLE identifica el nombre de la tabla, que debe ser única dentro del esquema. Después de CREATE TABLE van encerradas

entre paréntesis y separadas por coma las instrucciones de definir columnas y de definir restricciones sobre la tabla.

Definición de columna. Veamos las definiciones de las tres primeras columnas de TRABAJADOR:

```
ID_TRABAJADOR          IDENTIFICADOR     PRIMARY KEY,
NOMB_TRABAJADOR        CHARACTER (12)
TARIFA_HR              NUMERIC (5, 2)
```

dos de ASIGNACIÓN:

```
FECHA_INICIO           DATE
NÚM_DÍAS               INTERVAL DAY (3)
```

y dos de EDIFICIO:

```
TIPO       CHAR (9)      DEFAULT 'Oficina'
   CHECK (TIPO IN
     ('Oficina', 'Almacén', 'Comercio', 'Residencia')),
   CATEGORÍA          NUMERIC (1) DEFAULT 1
     CHECK (CATEGORÍA > 0 AND CATEGORÍA < 4)
```

Cada columna se define dando su nombre, su tipo de dato (que puede ser un tipo predefinido o definido por el usuario), cuál es su valor por defecto y cuándo se le aplican restricciones específicas (por ejemplo, NOT NULL (no nula), PRIMARY KEY (clave primaria) y restricciones CHECK). Las tres primeras columnas de este ejemplo se llaman respectivamente ID_TRABAJADOR, NOMB_TRABAJADOR y TARIFA_HR. Sus tipos de datos son IDENTIFICADOR, CHARACTER y NUMERIC. Las dos columnas que se tomaron de la tabla ASIGNACIÓN (FECHA_INICIO y NÚM_DÍAS) ilustran otros dos tipos de datos: DATE (fecha) e INTERVAL (intervalo).

Un tipo de dato numérico significa que los valores de los datos de las columnas deben ser números, posiblemente con punto decimal. El tipo de datos de ID_TRABAJADOR es IDENTIFICADOR, un dominio definido por el usuario que fue definido como NUMERIC (4), que quiere decir que tiene 4 dígitos significativos y *no* tiene dígitos después del punto (no tiene fracción). TARIFA_HR tiene un tipo de dato NUMERIC (5, 2) que significa que tiene cinco dígitos significativos, dos de los cuales están a la derecha del punto decimal.

Un tipo de dato CHARACTER (carácter) significa que los valores de las columnas consisten en cadenas de caracteres, formadas por caracteres alfanuméricos, posiblemente combinados con caracteres especiales. La longitud máxima de la cadena de caracteres se indica entre paréntesis. De este modo, NOMB_TRABAJADOR puede tener valores de cadenas de caracteres de longitnd no mayor de 12.

Un tipo de datos fecha (como el que tiene FECHA_INICIO) significa que los valores de los datos de la columna serán fechas, dadas con formato de año (cuatro dígitos), mes (dos dígitos) y día (dos dígitos). Un tipo de datos intervalo (como el de NÚM_DÍAS) es dado en años y/o meses o en días, minutos, horas y/o segundos. Se ha escogido el intervalo como tipo de dato DAY (3), lo que significa que puede ser cualquier cantidad de días de 0 a 999. Los tipos de datos fecha e intervalo tienen la ventaja que pueden sumarse o restarse. De modo que se le puede sumar NÚM_DÍAS a FECHA_INICIO y obtener la fecha en la que termina la asignación de un trabajador a un edificio (si se asume que después de la fecha de comienzo de un edificio el trabajador no toma vacaciones ni fines de semana y no trabaja en ningún otro edificio).

ID_TRABAJADOR está sujeto a dos restricciones: NOT NULL (no nulo) y PRIMARY KEY (clave primaria). La restricción NOT NULL se hereda de su dominio de defi-

nición y significa que ID_TRABAJADOR no puede tener un valor nulo. PRIMARY KEY significa que dos filas en la tabla TRABAJADOR no pueden tener el mismo valor en ID_TRABAJADOR y que para propósitos de referencias a clave externa ID_TRABAJADOR se considera una clave primaria. La restricción NOT NULL refuerza la regla de integridad de entidad que establece que una columna clave no puede ser nula.

Las columnas TIPO y CATEGORÍA en la tabla EDIFICIO tienen valores DEFAULT (por defecto) definidos. Si se añade una tupla a la tabla EDIFICIO y por ejemplo no se introduce ningún valor para la columna CATEGORÍA, el sistema pondrá automáticamente un 1 como valor de esa columna en esa tupla. Similarmente TIPO se pone automáticamente a 'Oficina' si no se le introduce ningún valor. Si en el esquema no se especifica un valor por defecto para una columna, se tomará como valor por defecto el valor nulo. Obsérvese que no se ha definido un valor por defecto para ID_TRABAJADOR puesto que éste toma el valor por defecto (0) de su dominio de definición.

Las columnas TIPO y CATEGORÍA también tienen definidas restricciones CHECK. Estas restricciones limitan los posibles valores que pueden introducirse en estas columnas. En particular, TIPO debe tener uno de los valores del conjunto ('Oficina', 'Almacén', 'Comercio', 'Residencia') y CATEGORÍA debe estar entre 1 y 3.

Ya que se ha explicado la sección de definición de columnas del esquema queda sólo por estudiar las restricciones sobre las tablas. En este esquema, las restricciones de tablas son:

```
FOREIGN KEY ID_SUPV REFERENCES TRABAJADOR
                    ON DELETE SET NULL
```

que restringe la tabla TRABAJADOR y:

```
PRIMARY KEY (ID_TRABAJADOR, ID_EDIFICIO),
FOREIGN KEY ID_TRABAJADOR REFERENCES TRABAJADOR
                          ON DELETE CASCADE,
FOREIGN KEY ID_EDIFICIO REFERENCES EDIFICIO
                          ON DELETE CASCADE
```

que restringe a la tabla ASIGNACIÓN. Veamos cada una de éstas. La restricción de clave externa de la tabla TRABAJADOR:

```
FOREIGN KEY ID_SUPV REFERENCES TRABAJADOR
                          ON DELETE SET NULL
```

clave externa recursiva. Una clave externa que referencia a su propia relación.

indica que ID_SUPV es una **clave externa recursiva** (ver Capítulo 5). Esto es, una clave externa que referencia a su propia relación. Observe que en esta instrucción las palabras que están encima de "ON DELETE ..." son idénticas a las que se usaron para definir las claves foráneas en el Capítulo 5. Simplemente se identifica la(s) columna(s) de la clave externa en la relación y se establece a cuál relación apuntan esta(s) columna(s). En este caso, la columna clave externa está en TRABAJADOR y apunta a la propia relación TRABAJADOR. La cláusula ON DELETE SET NULL le dice al sistema que si se borra la tupla a la que apunta la clave externa entonces el valor de ésta se debe poner a cero. Por ejemplo, supongamos que borramos la tupla con ID_TRABAJADOR 1311 en la relación TRABAJADOR de la Figura 7.1. Entonces el ID_SUPV de la primera tupla (el que tiene ID_TRABAJADOR 1235) apuntaría a un trabajador inexistente, es decir, a una tupla que ya no existe más en la base de datos. Esto violaría la integridad referencial. Para evitar esto, la cláusula ON DELETE SET NULL instruye al SGBD para que ponga el valor de la clave externa (ID_SUPV en la tupla 1235) a cero, si se borra la tupla a la que ésta apunta (la tupla 1311). Por tanto, antes de dicho borrado la tabla se ve en la forma:

TRABAJADOR

ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID-SUPV
1235	M. Faraday	12,50	Electricista	1311
...	...	...	...	...
1311	C. Coulomb	15,50	Electricista	1311

y después del borrado queda como:

TRABAJADOR

ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID-SUPV
1235	M Faraday	12,50	Electricista	[null]

Las restricciones de la tabla ASIGNACIÓN son:

```
PRIMARY KEY (ID_TRABAJADOR, ID_EDIFICIO),
FOREIGN KEY ID_TRABAJADOR REFERENCES TRABAJADOR
                          ON DELETE CASCADE,
FOREIGN KEY ID_EDIFICIO REFERENCES EDIFICIO
                          ON DELETE CASCADE
```

Con la excepción de la cláusula ON DELETE CASCADE en las dos restricciones de claves externas, que se estudiarán en breve, estas restricciones son elementales. La restricción de PRIMARY KEY (ID_TRABAJADOR, ID_EDIFICIO) le dice al sistema que ID_TRABAJADOR e ID_EDIFICIO constituyen una clave externa compuesta para la tabla. Por lo tanto, los valores combinados de estas dos columnas deben ser únicos para cada tupla en la tabla.

ON DELETE CASCADE es similar a ON DELETE SET NULL, ya que la acción se emprende siempre que se borra la tupla referida por la clave externa. Considérense las siguientes tuplas en la base de datos de la Figura 7.1:

ASIGNACIÓN

ID_TRABAJADOR	ID_EDIFICIO	FECHA_INICIO	NÚM_DÍAS
1235	312	10/10	5
1235	515	17/10	22

TRABAJADOR

ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID-SUPV
1235	M. Faraday	12,50	Electricista	1311

ID_TRABAJADOR en la tabla ASIGNACIÓN es una clave externa que apunta a la tabla TRABAJADOR. ON DELETE CASCADE significa que si se borra la tupla referida en la relación TRABAJADOR (1235 en el ejemplo), significa un borrado en "cascada" de todas las tuplas de la tabla ASIGNACIÓN que hacían referencia a éste. En este ejemplo el ID_TRABAJADOR 1235 en la tabla TRABAJADOR se refiere por dos tuplas de la tabla ASIGNACIÓN (tal como se mostró anteriormente). Si se borra la tupla 1235 de TRABAJADOR, entonces el sistema borrará automáticamente las dos tuplas correspondientes en la tabla ASIGNACIÓN. Este es el manejo correcto de esta clave externa, porque si se borra la tupla de TRABAJADOR, las tuplas de ASIGNACIÓN carecen de sentido y también deben borrarse.

La cláusula ON DELETE es similar a la cláusula ON UPDATE y ambas cláusulas tienen las opciones siguientes:

- CASCADE
- SET NULL
- SET DEFAULT

Si la opción es SET DEFAULT, el sistema pone el valor de clave externa al valor por defecto de la columna. Naturalmente, como señalan Date y Darwen (1994), la clave externa apuntará ahora a una tupla en la relación a la que hace referencia, teniendo este valor por defecto, por lo que debe existir una tupla tal en dicha relación. Si se omiten tanto ON DELETE como ON UPDATE, las acciones de borrar o actualizar serán prohibidas por el sistema si tales acciones pudieran violar la integridad referencial.

Antes de dejar esta definición de esquema se debe mencionar que, a diferencia de las relaciones en el modelo relacional, no se requiere que cada tabla SQL tenga una clave primaria. En otras palabras, si no se indica ninguna clave, entonces dos filas de la tabla pueden tener valores idénticos. Una tabla SQL se denomina **multiconjunto** (*multiset*) si puede tener entradas duplicadas. De esta manera, SQL se aparta un poco del modelo relacional. Por supuesto, una tabla que no tenga clave primaria no puede ser referida mediante clave externa desde otra tabla (Date y Darwen, 1994).

multiconjunto. Un conjunto que puede tener entradas duplicadas.

Esta **definición de esquema** *describe* la base de datos para el SGBD, pero no causa que se introduzcan datos concretos. Los valores se introducen y se manipulan mediante el lenguaje de manipulación de datos de SQL, que se describirá en detalle en la próxima sección.

definición de esquema. Descripción de una base de datos para el SGBD.

Otras instrucciones de manipulación de esquemas. Junto con la instrncción CREATE TABLE que define una nueva tabla, SQL-92 ofrece otras instrucciones para cambiar las definiciones de las tablas (ALTER TABLE) o para borrar las tablas del esquema (DROP TABLE). ALTER TABLE puede usarse para añadir una columna a una tabla, cambiar la definición de una columna existente o eliminar una columna de una tabla. DROP TABLE borrará todas las filas de la tabla y quitará del esquema la definición completa de la tabla. Un esquema completo puede eliminarse mediante la instrucción DROP SCHEMA. Sin embargo, puesto que ésta es una operación bastante peligrosa, o bien CASCADE o RESTRICT se deben especificar junto con ella.

```
DROP SCHEMA nombre de esquema CASCADE
```

significa eliminar el esquema con ese nombre al igual que todas las tablas, datos y otros esquemas que aún existan.

```
DROP SCHEMA nombre-de-esquema RESTRICT
```

significa eliminar el esquema sólo si todos los restantes objetos del esquema ya han sido borrados.

▼ Manipulación de datos

SQL contiene una gran variedad de capacidades de manipulación de datos, tanto para consulta como para actualización de una base de datos. Estas capacidades dependen sólo de la estructura lógica de la base de datos, no de su estructura física, consistente con los requisitos del modelo relacional. Inicialmente, la estructura sintáctica de SQL fue modelada sobre el cálculo relacional de Codd. La única operación del álgebra relacional que era soportada fue la unión. Sin embargo, SQL-92 implementa directamente la unión, la intersección, la diferencia y la reunión (*join*) y además tiene una sintaxis similar a la del cálculo relacional que ya tenía desde antes. Las operaciones de selección (*select*), proyectar (*project*) y producto estaban y continúan estando soportadas de una forma elemental, mientras que la división y la asignación son soportadas, aunque de forma más engorrosa.

Primero se describirá el lenguaje de consulta de SQL y luego las operaciones para introducir y cambiar los datos. Por último se describirán las operaciones de cambiar los datos porque su estructura depende de alguna forma de la estructura del lenguaje de consulta.

Consultas simples

consulta simple. Una consulta que involucra una sola tabla de la base de datos.

Una **consulta simple** es una consulta que afecta a una sola tabla de la base de datos. Las consultas simples nos ayudan a ilustrar la estructura básica de SQL.

Consulta: *¿Quiénes son los fontaneros?*

```
SELECT NOMB_TRABAJADOR
FROM TRABAJADOR
WHERE OFICIO = 'Fontanero'
```

Resultado:

```
NOMB_TRABAJADOR
C. Nemo
H. Rickover
```

Esta consulta ilustra las tres cláusulas más usadas del SQL: la cláusula SELECT, la cláusula FROM y la cláusula WHERE. Aunque en el ejemplo anterior se han puesto en líneas separadas, pueden colocarse en la misma línea. También se les puede poner sangrías y los términos dentro de las cláusulas pueden separarse por un número arbitrario de espacios en blanco. A continuación se analizan las características de cada cláusula.

cláusula SELECT. Señala las columnas que se desean en la consulta.

Select. La **cláusula SELECT** lista las *columnas* que se desean en el resultado de la consulta. Siempre tienen que ser columnas de una tabla relacional. En el ejemplo anterior, la tabla resultante tiene una sola columna (NOMB_TRABAJADOR), pero podría tener varias columnas, o incluir valores calculados o valores literales. Se mostrarán ejemplos de cada uno de éstos. Si el resultado deseado contiene más de una columna éstas se indicarán en la cláusula SELECT separadas por comas. Por ejemplo, SELECT ID_TRABAJADOR, NOMB_TRABAJADOR causaría que ID_TRABAJADOR y NOMB_TRABAJADOR sean listados como columnas de la tabla resultante.

cláusula FROM. Lista las tablas que son referidas por la consulta.

From. La **cláusula FROM** lista una o más *tablas* que van a ser referidas en la consulta. Todas las columnas relacionadas en la cláusula SELECT o WHERE se deben encontrar en una de las tablas de la cláusula FROM. En SQL-92 estas tablas pueden definirse directamente en el esquema de la base de datos como tablas base o como vistas, o pudieran ser en sí mismas tablas anónimas que sean el resultado de consultas SQL. En este último caso, la consulta se da explícitamente en la cláusula FROM.

cláusula WHERE. Nos da la condición para seleccionar las filas de las tablas indicadas.

Where. La **cláusula WHERE** contiene una *condición* para seleccionar las filas de las tabla(s) que se dan en la cláusula FROM. En el ejemplo, la condición ha sido que la columna OFICIO debe tener el valor literal 'Fontanero', éste se pone entre comillas simples, que es como se denotan los valores literales en SQL. La cláusula WHERE es la más versátil de SQL y puede contener una gran variedad de condiciones. Gran parte de lo que sigue se dedica a ilustrar las diferentes construcciones que se permiten en la cláusula WHERE.

La consulta SQL anterior se procesa por el sistema en el orden FROM, WHERE, SELECT. Esto es, las filas de la tabla a la que se hace referencia en la cláusula FROM (TRABAJADOR en este caso) se "ponen" en un área de trabajo para su procesamiento. Luego se aplica la cláusula WHERE a cada fila, una por una. Las filas que no satisfagan la

cláusula WHERE no se toman en cuenta. Aquellas filas que satisfagan la cláusula WHERE se procesan por la cláusula SELECT. En el ejemplo se selecciona el NOMB_TRABAJADOR para cada una de estas filas y todos estos valores se muestran en pantalla como resultado de la consulta.

Consulta: *Relacione todos los datos de los edificios que sean oficinas.*

```
SELECT *
FROM EDIFICIO
WHERE TIPO = 'Oficina'
```

Resultado:

ID_EDIFICIO	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORÍA
312	123 Elm	Oficina	2	2
210	1011 Birch	Oficina	3	1
111	1213 Aspen	Oficina	4	1

El "*" en la cláusula SELECT significa "la fila completa". Esta es una forma abreviada que se empleará con frecuencia.

Consulta: *¿Cuál es la tarifa semanal de cada electricista?*

```
SELECT NOMB_TRABAJADOR, 'Tarifa semanal = ' , 40 * TARIFA_HR
FROM TRABAJADOR
WHERE TIPO = 'Electricista'
ORDER BY NOMB_TRABAJADOR
```

Resultado:

NOMB_TRABAJADOR

```
C. Coulomb     Tarifa semanal = 620,00
D. Faraday     Tarifa semanal = 500,00
```

literal cadena de caracteres. Literales formados por caracteres alfanuméricos y por caracteres "especiales".

Esta consulta ilustra el uso de los literales alfanuméricos y de los **literales cadenas de caracteres** (en este ejemplo 'Tarifa semanal =') y el cálculo deutro de la cláusnla SELECT. En la cláusula SELECT se pueden definir cálculos que involucran columnas numéricas y literales numéricos combinados con las operaciones aritméticas estándar (+, -, *, /) agrupadas con paréntesis cuando sea necesario. Se ha incluido aquí una nueva cláusula, la cláusula ORDER BY (ordenar por), que se usa para ordenar el resultado de la consulta en orden alfanumérico ascendente por la columna que se especifique. Si se desea en orden descendente entonces esto debe especificarse añadiendo a la orden "DESC". Se pueden especificar múltiples columnas en la cláusula ORDER BY, algunas pueden estar en orden ascendente y algunas en orden descendente. La primera columna por la que se quiere ordenar se especifica primero.

Consulta: *¿Quiénes tienen una tarifa por hora entre $10 y $12?*

```
SELECT *
FROM TRABAJADOR
WHERE TARIFA_HR >= 10 AND TARIFA_HR <= 12
```

Resultado:

ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID-SUPV
2920	R. Garret	10,00	Albañil	2920
1520	H. Rickover	11,75	Fontanero	1520

operadores de comparación
=, <>, <, >, <=, >=

conectores booleanos
AND, OR, NOT.

Esta consulta ilustra algunas características adicionales de la cláusula WHERE: Los operadores de comparación y el conector booleano AND. Los seis **operadores de comparación** (=, <> [no igual], <, >, <=, >=) se pueden usar para comparar columnas con otras columnas o con literales. Los **conectores booleanos** AND, OR y NOT pueden usarse para crear condiciones compuestas o para negar una condición. Los paréntesis también pueden

usarse en la forma acostumbrada de los lenguajes de programación para agrupar condiciones. Esta consnlta también hubiera podido resolverse usando el operador BETWEEN (entre):

```
SELECT *
FROM TRABAJADOR
WHERE TARIFA_HR BETWEEN 10 AND 12
```

BETWEEN puede usarse en la comparación de algún valor con otros dos valores, el primero menor que el segundo, para saber si el valor a comparar está entre estos dos valores.

Consulta: *Indique los fontaneros, albañiles y electricistas.*

```
SELECT *
FROM TRABAJADOR
WHERE OFICIO IN ('Fontanero', 'Albañil', 'Electricista')
```

Resultado:

ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID-SUPV
1235	M. Faraday	12,50	Electricista	1311
1412	C. Nemo	13,75	Fontanero	1520
2920	R. Garret	10,00	Albañil	2920
1520	H. Rickover	11,75	Fontanero	1520
1311	C. Coulomb	15,50	Electricista	1311

Esta consulta introduce e ilustra el uso del operador de comparación IN. La cláusula WHERE ha evaluado 'true' si el tipo del oficio de la fila se encuentra *en el conjunto* [1] indicado entre paréntesis, esto es, si el oficio es fontanero, albañil o electricista. Ya tendremos más ocasiones de utilizar el operador IN cuando veamos las subconsultas.

caracteres comodines. Símbolos especiales que valen por cualquier cadena de caracteres.

Supóngase que no podemos recordar exactamente el nombre exacto de un oficio. Es decir, ¿"Electricista", "Eléctrico" o "Electrónico"? Se pueden usar **caracteres comodines** (*wild card character*) que son símbolos especiales que valen como cualquier cadena no especificada de caracteres, lo que hace más fácil si no sabemos el nombre exacto en una consulta.

Consulta: *Encontrar todos los caracteres cuyo oficio comience con "Elec".*

```
SELECT *
FROM TRABAJADOR
WHERE OFICIO LIKE 'Elec%'
```

Resultado:

ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID-SUPV
1235	M. Faraday	12,50	Electricista	1311
1311	C. Coulomb	15,50	Electricista	1311

SQL tiene dos caracteres comodines el % (tanto por ciento) y el _ (subrayado). El subrayado vale por exactamente *un* carácter cualquiera. El tanto por ciento vale por cero o cualquier cantidad de caracteres no especificados. El operador LIKE se usa para comparar variables de caracteres con literales cuando se utilizan comodines. Otros ejemplos:

```
NOMB_TRABAJADOR LIKE '___Coulomb'
NOMB_TRABAJADOR LIKE '___C%'
```

[1] En inglés *in the set*, de ahí el nombre del operador IN (N. del T.).

El primer ejemplo evalúa verdadero si NOMB_TRABAJADOR consiste de tres caracteres seguido de 'Coulomb'. Por ejemplo, en la relación TRABAJADOR, todos los nombres de trabajadores comienzan por una inicial, seguida de un punto y luego un espacio. Por lo tanto, la condición encontrará a todos los trabajadores cuyo apellido sea "Coulomb". En el segundo ejemplo, la condición identificaría a los trabajadores cuyo apellido comience con "C".

Consulta: *Encuentre todas las asignaciones que comiencen en las dos próximas semanas.*

```
SELECT *
FROM ASIGNACIÓN
WHERE FECHA_INICIO BETWEEN CURRENT_DATE AND CURRENT_DATE + INTERVAL
'14' DAY
```

Resultado: (Suponiendo que CURRENT_DATE = 10/10)

ID_TRABAJADOR	ID_EDIFICIO	FECHA_INICIO	NÚM_DÍAS
1235	312	10/10	5
1235	515	17/10	22
3231	111	10/10	8
1412	435	15/10	15
3231	312	24/10	20
1311	460	23/10	24

Esta consulta ilustra el uso del operador BETWEEN con valores de fecha y el tipo intervalo de datos. CURREN_DATE (fecha actual) es una función que siempre devuelve la fecha de hoy. La expresión

```
CURRENT_DATE + INTERVAL '14' DAY
```

añade a la fecha actual un intervalo de dos semanas. De este modo se selecciona una tupla (asumiendo que la fecha actual es 10/10) si su columna FECHA_INICIO está en el rango 10/10 a 24/10. De aquí se puede ver que se pueden combinar fechas con intervalos de fecha mediante suma. También podría ser mediante resta. Es más se pueden multiplicar campos intervalos por valores enteros. Por ejemplo, snpongamos que se quiere identificar una fecha que está una cierta cantidad de semanas más adelante (dada por la variable NÚM_SEMANAS), se puede escribir

```
CURRENT_DATE + INTERVAL '7' DAY * NÚM_SEMANAS
```

Consultas multi-tablas

La capacidad para conectar datos sobre las fronteras de las tablas es esencial en cualquier lenguaje de base de datos. En el álgebra relacional esto se lleva a cabo con la *reunión* (*join*). Aunque una gran parte de SQL se ha modelado después del cálculo relacional, éste conecta los datos entre las tablas de la misma manera que la reunión (*join*) del álgebra relacional. A continuación se muestra cómo se hace esto. Considere la consulta:

Consulta: *¿Cuáles son los oficios de los trabajadores asignados al edificio 435?*

Los datos necesarios para responder a esta consulta se encuentran en dos relaciones: TRABAJADOR y ASIGNACIÓN. La solución SQL requiere poner ambas relaciones en la cláusula FROM junto con un tipo particular de condición de cláusula WHERE:

```
SELECT OFICIO
FROM TRABAJADOR, ASIGNACIÓN
WHERE TRABAJADOR.ID_TRABAJADOR = ASIGNACIÓN.ID_TRABAJADOR
      AND ID_EDIFICIO = 435
```

¿Qué es lo que está pasando aquí? Se deben considerar dos pasos en el procesamiento que hace el sistema para la consulta:

producto cartesiano. Resultado de aparear cada fila de una tabla con *todas* las filas de otra tabla.

1. Como es usual, la cláusula FROM se procesa primero. En este caso, sin embargo, puesto que hay dos tablas en la cláusula, el sistema crea el **producto cartesiano** de las filas en estas tablas. Esto significa que se crea (lógicamente) una tabla enorme que consiste de todas las *columnas* de ambas tablas, emparejando cada *fila* de una tabla con cada *fila* en la otra tabla. En este ejemplo, puesto que hay cinco columnas en TRABAJADOR y cuatro columnas en ASIGNACIÓN, habrá entonces nueve columnas en el producto cartesiano creado por la cláusula FROM. El total de filas en este producto cartesiano es $m * n$, donde m es el número de filas de TRABAJADOR y n es el número de filas en ASIGNACIÓN: Puesto que TRABAJADOR tiene 7 filas y ASIGNACIÓN tiene 19 filas, el producto cartesiano tendrá 7 * 19, o sea, 133 filas. Una ilustración de esto se da en el Capítulo 6 (Figuras 6.6(a) y 6.6(b). Si hay más de dos tablas en la cláusula FROM, el producto cartesiano se creará con *todas* las tablas en la cláusula.
2. Después de crear esta relación gigante se aplica la cláusula WHERE, como antes. Se examina cada fila de la relación creada por la cláusula FROM conforme a la cláusula WHERE. No se consideran aquellas que no la satisfacen. La cláusula SELECT se aplica a las filas restantes.

La cláusula WHERE en esta consulta contiene dos condiciones:

1. TRABAJADOR.ID_TRABAJADOR = ASIGNACIÓN.ID_TRABAJADOR
2. ID_EDIFICIO = 435

La primera de estas condiciones es la reunión (*join*). Observe que puesto que tanto TRABAJADOR como ASIGNACIÓN contienen una columna llamada ID_TRABAJADOR, el producto de ambas relaciones contendrá *dos* columnas con este nombre. Para distinguir entre ellas se pone como prefijo al nombre de la columna el nombre de la relación de la cual proviene la columna.

La primera condición establece que, para que una fila cualquiera sea seleccionada, el valor de la columna ID_TRABAJADOR, que proviene de la relación TRABAJADOR, debe ser igual al valor de la columna ID_TRABAJADOR que proviene de la relación ASIGNACIÓN. En efecto, se está haciendo una reunión de dos relaciones sobre la colum-

TRABAJADOR.ID_TRABAJADOR	*NOMB_TRABAJADOR*	*TARIFA_HR*	*OFICIO*	*ID_SUPV*
1412	C. Nemo	13,75	Fontanero	1520
2920	R. Garret	10,00	Albañil	2920
1311	C. Coulomb	15,50	Electricista	1311

ASIGNACIÓN.ID_TRABAJADOR	*ID_EDIFICIO*	*FECHA_INICIO*	*NÚM_DÍAS*
1412	435	15/10	15
2920	435	28/10	10
1311	435	08/10	12

na ID_TRABAJADOR. Todas las filas para las que estas dos columnas no son iguales se eliminan de la relación producto. Esto es lo que pasa precisamente en la reunión natural (*natural join*) del álgebra relacional. (Sin embargo, esto difiere de la reunión natural, ya que en SQL la columna redundante ID_TRABAJADOR *no* se elimina automáticamente en SQL.) En la Figura 7.2 se muestra la reunión completa de estas dos relaciones con la condición adicional de que ID_EDIFICIO = 435. La aplicación de la cláusula SELECT produce el resultado final siguiente de la consulta:

```
OFICIO
Fontanero
Albañil
Electricista
```

Ahora se mostrará cómo se puede hacer en SQL una reunión de una relación consigo misma.

Consulta: *Indicar los trabajadores con los nombres de sus supervisores.*

```
SELECT A.NOMB_TRABAJADOR, B.NOMB_TRABAJADOR
FROM TRABAJADOR A, TRABAJADOR B
WHERE B.ID_TRABAJADOR = A.ID_SUPV
```

alias. Un nombre alternativo que se le da a una relación.

La cláusula FROM en este ejemplo define dos "copias" de la relación TRABAJADOR y les da los alias *A* y *B*. Un **alias** es un nombre alternativo que se le da a una relación. Las copias *A* y *B* de TRABAJADOR se conectan en la cláusula WHERE al hacer que el ID_TRABAJADOR en *B* sea igual al ID_TRABAJADOR de *A*. De modo que cada fila en *A* se asocia con la fila en *B* que contiene la información sobre que la fila de *A* es su supervisor (Figura 7.3). Seleccionando los dos nombres de trabajador de cada una de las filas se obtiene la lista solicitada de pares de trabajadores con sus supervisores:

A.NOMB_TRABAJADOR	*B.NOMB_TRABAJADOR*
M. Faraday	C. Coulomb
C. Nemo	H. Rickover
R. Garret	R. Garret
P. Mason	P. Mason
H. Rickover	H. Rickover
C. Coulomb	C. Coulomb
J.	Barrister P. Mason

A.NOMB_TRABAJADOR representa al trabajador y *B*. NOMB_TRABAJADOR representa al supervisor. Observe que algunos trabajadores se supervisan a sí mismos, como se refleja en el hecho de que en esos casos ID_SUPV = ID_TRABAJADOR.

En SQL se puede hacer una reunión de más de dos relaciones a la vez:

Consulta: *Indique los nombres de los trabajadores asignados a edificios que sean oficinas.*

Se necesita hacer una reunión de las tres relaciones para obtener los datos. Esto se hace en la consulta siguiente:

```
SELECT NOMB_TRABAJADOR
FROM TRABAJADOR, ASIGNACIÓN, EDIFICIO
WHERE TRABAJADOR.ID_TRABAJADOR = ASIGNACIÓN.ID_TRABAJADOR AND
   ASIGNACIÓN.ID_EDIFICIO = EDIFICIO.ID_EDIFICIO AND
   TIPO = 'Oficina'
```

A.ID_TRABAJADOR	A.NOMB_TRABAJADOR	A.TARIFA_HR	A.OFICIO	A.ID_SUPV
1235	M. Faraday	12,50	Electricista	1311
1412	C. Nemo	13,75	Fontanero	1520
2920	R. Garret	10,00	Albañil	2920
3231	P. Mason	17,40	Carpintero	3231
1520	H. Rickover	11,75	Fontanero	1520
1311	C. Coulomb	15,50	Electricista	1311
3001	J. Barrister	8,20	Carpintero	3231

B.ID_TRABAJADOR	B.NOMB_TRABAJADOR	B.TARIFA_HR	A.OFICIO	A.ID_SUPV
1311	C. Coulomb	15,50	Electricista	1311
1520	H. Rickover	11,75	Fontanero	1520
2920	R. Garret	10,00	Albañil	2920
3231	P. Mason	17,40	Carpintero	3231
1520	H. Rickover	11,75	Fontanero	1520
1311	C. Coulomb	15,50	Electricista	1311
3231	P. Mason	17,40	Carpintero	3231

Resultado:

NOMB_TRABAJADOR
M. Faraday
C. Nemo
R. Garret
P. Mason
H. Rickover
J. Barrister

Obsérvese que si un nombre de columna (por ejemplo, ID_TRABAJADOR o ID_EDIFICIO) aparece en más de una relación se debe poner como prefijo de la columna el nombre de la relación de donde proviene para evitar ambigüedad. Pero si el nombre de la columna aparece en una sola relación, como es el caso de TIPO en este ejemplo, entonces no hay ambigüedad y no es necesario ningún prefijo.

Esta instrucción SQL origina la creación de una sola relación a partir de tres relaciones de la base de datos. Las dos primeras relaciones se reúnen con ID_TRABAJADOR, después de lo cual se hace una reunión sobre ID_EDIFICIO de la relación resultante con la tercera relación. La condición

```
TIPO = 'Oficina'
```

en la cláusula WHERE hace que se eliminen todas las filas, excepto aquellas que cumplen que los edificios son oficinas. Esto satisface los requisitos de la consulta.

Subconsultas

subconsulta. Una consulta dentro de una consulta.

Una **subconsulta,** o una consulta dentro de una consulta, puede ponerse dentro de la cláusula WHERE de una consulta. Esto produce una expansión de las capacidades de una cláusula WHERE. Considérese el ejemplo siguiente:

Consulta: *¿Cuáles son los oficios de los trabajadores asignados al edificio 435?*

Se usará este ejemplo para ilustrar la reunión (*join*). Las subconsultas nos dan una equivalencia parcial con una reunión.

```
SELECT OFICIO
FROM TRABAJADOR
WHERE ID_TRABAJADOR IN
   (SELECT ID_TRABAJADOR
   FROM ASIGNACIÓN
   WHERE ID_EDIFICIO = 435)
```

En este ejemplo la subconsulta es

```
(SELECT ID_TRABAJADOR
FROM ASIGNACIÓN
WHERE ID_EDIFICIO = 435)
```

consulta externa. La consulta principal que contiene a las subconsultas.

La consulta que incluye a esta subconsulta se denomina **consulta externa** (*outer query*) o consulta principal. La subconsulta provoca que se genere el siguiente conjunto de IDs de trabajadores:

```
ID_TRABAJADOR
2920
1412
1311
```

Este conjunto de IDs ocupa el lugar de la subconsulta para la consulta más externa. En este punto la consulta externa se ejecuta usando el conjunto generado por la subconsulta. Esta consulta externa provoca que cada fila de TRABAJADOR sea evaluada con respecto a la cláusula WHERE. Si en el ID_TRABAJADOR de la fila está (*IN*), el conjunto generado por la subconsulta se selecciona en el campo OFICIO de la fila y se muestra en pantalla como resultado de la consulta:

```
OFICIO
Fontanero
Albañil
Electricista
```

Es muy importante que la cláusula SELECT de la subconsulta contenga a ID_TRABAJADOR y *sólo* a ID_TRABAJADOR. De lo contrario, la cláusula WHERE de la consulta externa que establece si ID_TRABAJADOR está en (*IN*), un conjunto de IDs de trabajadores no tendría sentido.

Obsérvese que la subconsulta puede ejecutarse lógicamente antes de que *alguna* fila sea examinada por la consulta principal. En cierto sentido, la subconsulta es independiente de la consulta principal. Esta podría ejecutarse como una consulta propiamente dicha. Se dice que esta clase de **subconsulta no está correlacionada** con la consulta principal. Como se verá en breve, las subconsultas pudieran también estar correlacionadas.

subconsulta no correlacionada. Una subconsulta cuyos valores no dependen de ninguna consulta más externa.

A continuación se da un ejemplo de una subconsulta dentro de una subconsulta.

Consulta: *Indicar los nombres de los trabajadores asignados a edificios que sean oficinas.*

De nuevo vamos a trabajar con la consulta que se nsó para estudiar la reunión (*join*).

```
SELECT NOMB_TRABAJADOR
FROM TRABAJADOR
WHERE ID_TRABAJADOR IN
   (SELECT ID_TRABAJADOR
   FROM ASIGNACIÓN
   WHERE ID_EDIFICIO IN
     (SELECT ID_EDIFICIO
     FROM EDIFICIO
     WHERE TIPO = 'Oficina' ))
```

Resultado:

```
NOMB_TRABAJADOR
M. Faraday
C. Nemo
R. Garret
P. Mason
H. Rickover
J. Barrister
```

Observe que no se han puesto prefijos a ningún nombre de columna. Esto se debe a que cada subconsulta tiene que ver sólo con una relación, por lo que no da lugar a ambigüedad.

La evaluación de esta consulta procede de adentro hacia afuera. Por tanto, la subconsulta más interna (o la "más al fondo")[5] se evalúa primero, luego la subconsulta que contiene a ésta y más tarde la consulta externa.

Subconsultas correlacionadas. Las subconsultas que hemos estudiado hasta ahora son independientes de la consulta más externa que las usa. Por esto entendemos que las subconsultas podrían existir como consultas por derecho propio. Ahora vamos a ver una clase de subconsultas cuyos valores en ejecución dependen de la fila que está siendo examinada por la consulta principal. Tales subconsultas se llaman **subconsultas correlacionadas**.

subconsulta correlacionada. Una subconsulta cuyos resultados dependen de la fila que se está examinando por una consulta más externa.

Consulta: *Indicar los trabajadores que reciben una tarifa por hora mayor que la de su supervisor.*

La palabra pivote en esta consulta es *su*. Esto es, la fila del supervisor a examinar depende directamente de la fila del trabajador que está siendo examinada. Esta consulta puede resolverse usando una subconsulta correlacionada.

```
SELECT NOMB_TRABAJADOR
FROM TRABAJADOR A
WHERE A.TARIFA_HR >
   (SELECT B.TARIFA_HR
   FROM TRABAJADOR B
   WHERE B.ID_TRABAJADOR = A.ID_TRABAJADOR)
```

Resultado:

```
NOMB_TRABAJADOR
C. Nemo
```

Los pasos lógicos involucrados en la ejecución de esta consulta son los siguientes:

1. El sistema hace dos copias de la relación TRABAJADOR, la copia *A* y la copia *B*. Como ya se han definido, *A* se refiere a los trabajadores y *B* se refiere a los supervisores.

[5] En inglés, *bottom-most* (N. del T.).

2. El sistema examina entonces cada fila de *A*. Se selecciona una fila si satisface la condición de la cláusula WHERE. Esta condición establece que cada fila será seleccionada si su TARIFA_HR es *mayor que* la TARIFA_HR generada por la subconsulta.
3. La subcousulta selecciona la TARIFA_HR de las filas de *B* cuyo ID_TRABAJADOR sea igual al del ID_SUPV de la fila de *A* que está siendo examinada por la consulta principal. Esta es la TARIFA_HR del supervisor de la fila de *A*.

Obsérvese que puesto que *A*.TARIFA_HR sólo puede compararse con un valor simple, la subconsulta debe entonces necesariamente generar un único valor. Este valor *cambia dependiendo de la fila de A que está siendo examinada*. De este modo, la subconsulta está correlacionada con la consulta principal. Se verán otras aplicaciones de las subconsultas correlacionadas más adelante cuando se estudien las funciones integradas (*built-in*).

EXISTS y NOT EXISTS

Supongamos que se quiere detectar a todos los trabajadores que *no* están asignados a un determinado edificio. Aparentemente una consulta como ésta parecería resolverse fácilmente por la simple negación de la versión afirmativa de la consulta. Supongamos, por ejemplo, que el edificio de interés es el que tiene ID_EDIFICIO 435, considérese esta solución:

```
SELECT ID_TRABAJADOR
FROM ASIGNACIÓN
WHERE ID_EDIFICIO < > 435
```

Desgraciadamente, ésta es una mala formulación de la solución. Esta solución simplemente da los IDs de los trabajadores que están trabajando en otros edificios que no son el 435. Obviamente algunos de estos trabajadores pudieran también estar asignados al edificio 435. Una solución correcta podría utilizar el operador NOT EXISTS:

```
SELECT ID_TRABAJADOR
FROM TRABAJADOR
WHERE NOT EXISTS
   (SELECT *
    FROM ASIGNACIÓN
    WHERE ASIGNACIÓN.ID_TRABAJADOR = TRABAJADOR.ID_TRABAJADOR AND
      ID_EDIFICIO = 435 )
```

Resultado:

ID_TRABAJADOR

```
1235
3231
1520
3001
```

operador EXISTS. Evalúa verdadero si el conjunto resnltante es no vacío.

Los operadores **EXISTS** y **NOT EXISTS** siempre preceden a una subconsulta. EXITS evalúa verdadero si el subconjunto resultante de la subconsulta no es vacío. Si el conjunto resultante es vacío, entonces el operador EXISTS da valor falso. El operador NOT EXISTS naturalmente trabaja de modo opuesto. Este evalúa verdadero si el conjunto resultante es vacío y falso en caso contrario.

operador NOT EXISTS. Evalúa verdadero si el conjunto resultante es vacío.

En este ejemplo se ha usado el NOT EXISTS. La subconsulta selecciona todas aquellas filas en ASIGNACIÓN que tienen el mismo ID_TRABAJADOR que la fila que está siendo examinada por la consulta principal y que además tienen ID_EDIFICIO igual a

435. Si este conjunto es vacío entonces se selecciona la fila trabajador que se está examinando en la consulta principal, puesto que esto significa que el trabajador en cuestión *no* trabaja en el edificio 435.

La solución que se tiene aquí involucra a una subconsulta correlacionada. Si se usa IN en lugar de NOT EXISTS, se puede usar una subconsulta no correlacionada:

```
SELECT ID_TRABAJADOR
FROM TRABAJADOR
WHERE ID_TRABAJADOR NOT IN
   (SELECT ID_TRABAJADOR
   FROM ASIGNACIÓN
   WHERE ID_EDIFICIO =- 435)
```

Esta solución también es más simple que la solución usando NOT EXISTS. Por lo tanto, parece natural preguntarse por qué tendríamos que usar EXISTS y NOT EXISTS. La respuesta es simplemente que NOT EXISTS ofrece la única forma disponible de resolver consultas que contengan el cuantificador "todos" en su condición. En el Capítulo 6 se vio que tales consultas se resolvían con la división en el álgebra relacional y con el cuantificador universal en el cálculo relacional. El siguiente ejemplo ilustra una consulta que contiene el cuantificador "todos" en su condición:

Consulta: *Indicar los trabajadores que están asignados a todos los edificios.*

Esta consulta puede resolverse en SQL usando una doble negación. La consulta con esta doble negación es:

Consulta: *Indicar los trabajadores tales que NO hay un edificio al cual NO estén asignados.*

Se han enfatizado las dos negaciones. Debe estar claro que esta consulta es lógicamente equivalente a la anterior.

Formulemos ahora una solución en SQL. Para aclarar esta última solución, primero daremos una solución al problema preliminar: el problema de identificar los edificios a que un trabajador hipotético "1234" *no* ha sido asignado.

```
(I) SELECT ID_EDIFICIO
    FROM EDIFICIO
    WHERE NOT EXISTS
     (SELECT *
     FROM ASIGNACIÓN
     WHERE ASIGNACIÓN.ID_EDIFICIO = EDIFICIO.ID_EDIFICIO AND
       ASIGNACIÓN.ID_TRABAJADOR = 1234)
```

Se ha etiquetado esta consulta con (I) porque más adelante se hará referencia a la misma. Si no hay ningún edificio que satisfaga esta consulta, entonces el trabajador 1234 debe asignarse a *todo* edificio y satisface la consulta original. Para obtener una solución a la consulta original, el siguiente paso es generalizar la consulta (I) del trabajador específico 1234 a una variable ID_TRABAJADOR y hacer que esta consulta con dicha modificación pase a ser una subconsulta de una consulta más grande. Esto se logra de la siguiente forma:

```
(II) SELECT ID_TRABAJADOR
     FROM TRABAJADOR
     WHERE NOT EXISTS
       (SELECT ID_EDIFICIO
       FROM EDIFICIO
```

```
WHERE NOT EXISTS
  (SELECT *
  FROM ASIGNACIÓN
  WHERE ASIGNACIÓN.ID_EDIFICIO = EDIFICIO.ID_EDIFICIO AND
        ASIGNACIÓN.ID_TRABAJADOR = TRABAJADOR.ID_TRABAJADOR ) )
```

Resultado:

ID_TRABAJADOR
1412

Obsérvese que la subconsulta que comienza en la cuarta línea de la consulta (II) es idéntica a la consulta (I), excepto que se ha reemplazado el "1234" con TRABAJADOR.ID_TRABAJADOR. La consulta (II) se puede leer como sigue:

Seleccione ID_TRABAJADOR de TRABAJADOR si no existe un edificio al cual ID_TRABAJADOR no esté asignado.

Esto satisface el requisito de la consulta original.

Se ve entonces que el NOT EXISTS se puede usar para formular respuestas a tipos de consultas para los cuales se necesitaría la operación de división del álgebra relacional o el cuantificador universal del cálculo relacional (Capítulo 6). Sin embargo, en términos de facilidades de uso, el operador NOT EXISTS no parece ofrecer ninguna ventaja en particular. Es decir, no parece que las consultas SQL que usen dos veces el NOT EXISTS sean más fáciles de entender que las soluciones que usan la división en el álgebra relacional o el cuantificador universal en el cálculo relacional. Se necesitan investigaciones adicionales para desarrollar construcciones lingüísticas que permitan una solución más natural a estos tipos de consultas.

Funciones integradas (built-in functions)

Considere preguntas como éstas:

¿Cuáles son las tarifas menores y mayores por hora? ¿Cuál es el promedio de días que están asignados los trabajadores al edificio 435? ¿Cuál es el número total de días asignado a fontanería en el edificio 312? ¿Cuántos tipos diferentes de oficios hay?

Estas preguntas requieren de funciones estadísticas que examinen un conjunto de filas en una relación y produzcan un solo valor. SQL provee tales funciones que se denominan **funciones integradas** (*built-in*) o **funciones de conjuntos** (*set functions*). Las cinco funciones son SUM, AVG, COUNT, MÁX y MÍN.

función integrada. Una función estadística que opera sobre un conjunto de filas - SUM, AVG, COUNT, MÁX, MIN.

función de conjuntos. Una función integrada.

Consulta: *¿Cuáles son la tarifa por hora menor y mayor?*

```
SELECT MÁX(TARIFA_HR), MÍN(TARIFA_HR)
FROM TRABAJADOR
```

Resultado: 17,40, 8,20

Las funciones MÁX y MÍN operan sobre una sola columna de la relación. Seleccionan respectivamente el valor mayor y el menor de los que se encuentran en la columna. La solución a esta consulta no incluye una cláusula WHERE. Este no es necesariamente el caso en la mayoría de las consultas, como se muestra en el ejemplo siguiente.

Consulta: *¿Cuál es el promedio de días que los trabajadores están asignados al edificio 435?*

```
SELECT AVG(NUM_DIAS)
FROM ASIGNACIÓN
WHERE ID_EDIFICIO = 435
```

Resultado: 12,33

Para calcular este promedio sólo se han considerado las filas de ASIGNACIÓN correspondientes al edificio 435. Como es normal en SQL, la cláusula WHERE restringe las filas que se deben considerar.

Consulta: *¿Cuál es el número total de días asignados a fontanería en el edificio 312?*

```
SELECT SUM(NÚM_DÍAS)
FROM ASIGNACIÓN, TRABAJADOR
WHERE TRABAJADOR.ID_TRABAJADOR = ASIGNACIÓN.ID_TRABAJADOR AND
    OFICIO = 'Fontanero' AND
    ID_EDIFICIO = 312
```

Resultado: 27

Esta solución usa la reunión de ASIGNACIÓN y TRABAJADOR. Esto fue necesario puesto que OFICIO está en TRABAJADOR e ID_EDIFICIO está en ASIGNACIÓN.

Consulta: *¿Cuántos tipos de oficios diferentes hay?*

```
SELECT COUNT (DISTINCT OFICIO)
FROM TRABAJADOR
```

Resultado: 4

Puesto que el mismo oficio se repite varias veces en diferentes filas, es necesario usar la palabra clave "DISTINCT" en esta consulta. De esta manera, el sistema no cuenta el mismo tipo de oficio más de una vez. **DISTINCT** puede usarse con cualquiera de las funciones integradas, aunque es un operador redundante en el caso de las funciones MÁX y MÍN.

DISTINCT. Operador que elimina las filas duplicadas.

SUM y AVG deben emplearse con columnas que son numéricas. Las otras funciones pueden usarse con datos numéricos o con cadenas de caracteres. Todas las funciones, excepto COUNT, pueden aplicarse a expresiones que computen un valor. Por ejemplo:

Consulta: *¿Cuál es el salario medio semanal?*

```
SELECT AVG(40 * TARIFA_HR)
FROM TRABAJADOR
```

Resultado: 509,14

COUNT se puede referir a filas completas en lugar de a una sola columna:

Consulta: *¿Cuántos edificios tienen nivel de calidad 3?*

```
SELECT COUNT ( * )
FROM EDIFICIO
WHERE NIVEL_CALIDAD = 3
```

Resultado: 3

Como muestran todos estos ejemplos, si una función integrada aparece en una cláusula SELECT, entonces nada más que funciones integradas pueden aparecer en dicha cláusula SELECT. La única excepción ocurre en combinación con la cláusula GROUP BY, que se examinará a continuación.

GROUP BY y HAVING

Los dirigentes están a menudo interesados en conocer información estadística que se aplique a cada grupo dentro de un conjunto de grupos. Por ejemplo, consideremos la consulta siguiente:

Consulta: *Para cada supervisor, ¿cuál es la tarifa por horas más alta que se le paga a un trabajador que informe a este supervisor?*

Para resolver esta consulta se deben dividir los trabajadores en grupos, en los que en cada grupo estén los trabajadores que informen a un mismo supervisor. Luego se determina el pago máximo en cada grupo. Esto se hace en SQL de la manera siguiente:

```
SELECT ID_SUPV, MÁX(TARIFA_HR)
FROM TRABAJADOR
GROUP BY ID_SUPV
```

Resultado:

ID_SUPV	*MÁX(TARIFA_HR)*
1311	15,50
1520	13,75
2920	10,00
3231	17,40

Para procesar esta consulta, el sistema procede dividiendo primero en grupos las filas de TRABAJADOR, usando la siguiente regla: Las filas se ponen en un mismo grupo si y sólo si tienen el mismo ID_SUPV. Luego se aplica la cláusula SELECT a cada grupo. Puesto que un grupo dado sólo puede tener un valor para ID_SUPV, no hay ambigüedad con el valor de ID_SUPV para ese grupo. La cláusula SELECT muestra ID_SUPV y calcula y muestra MÁX(TARIFA_HR) *para cada grupo*. El resultado queda como se mostró arriba.

Sólo los nombres de columna que aparezcan en una cláusula GROUP BY pueden aparecer en una cláusula SELECT que tenga también una función integrada. Note que ID_SUPV puede aparecer en la cláusula SELECT puesto que aparece en la cláusula GROUP BY.

cláusula GROUP BY. Indica cuáles filas deben agruparse sobre un valor común de las columna(s) especificada(s).

La cláusula **GROUP BY** sugiere la posibilidad de hacer cálculos sofisticados. Por ejemplo, se puede querer conocer el promedio de todas estas tarifas máximas. Los cálculos dentro de las funciones integradas se restringen de modo que ninguna función integrada puede contener a su vez otra función integrada. De modo que una expresión como

```
AVG( MÁX (TARIFA_HR) )
```

es ilegal. Resolver una consulta como ésta requiere dos pasos. El primer paso sería poner las tarifas horarias máximas en una nueva relación y el segundo paso sería calcular el promedio de éstas.

Es válido usar una cláusula WHERE con un GROUP BY:

Consulta: *Para cada tipo de edificio, ¿cuál es el nivel de calidad medio para los edificios de categoría 1?*

```
SELECT TIPO, AVG(NIVEL_CALIDAD)
FROM EDIFICIO
WHERE CATEGORÍA = 1
GROUP BY TIPO
```

Resultado:

TIPO	AVG(NIVEL_CALIDAD)
Comercio	1
Residencia	3
Oficina	3,5

La cláusula WHERE se ejecuta antes de la cláusula GROUP BY. De este modo, ningún grupo puede tener una fila con una categoría que no sea 1. Las filas con categoría 1 se agrupan entonces por TIPO y la cláusula SELECT se le aplica a cada grupo.

cláusula HAVING. Una cláusula que impone condiciones a los grupos

También se le puede aplicar una condición a los grupos formados por la cláusula GROUP BY. Esto se hace con la **cláusula HAVING**. Supongamos que se quiere hacer más específica una de las consultas anteriores.

Consulta: *Para cada supervisor que dirige a más de un trabajador, ¿cuál es la tarifa por horas más alta que se le paga a un trabajador que informe a dicho supervisor?*

Esta consulta se podría plantear usando la cláusula HAVING:

```
SELECT ID_SUPV, MÁX(TARIFA_HR)
FROM TRABAJADOR
GROUP BY ID_SUPV
HAVING COUNT( * ) > 1
```

Resultado:

ID_SUPV	*MÁX(TARIFA_HR)*
1311	15,50
1520	13,75
3231	17,40

La diferencia entre la cláusula WHERE y la cláusula HAVING es que la cláusula WHERE se aplica a las *filas*, mientras que la cláusula HAVING se aplica a los *grupos*.

Una consulta puede contener ambas cláusulas, tanto una WHERE como una HAVING. En tal caso, la cláusula WHERE se aplica primero, puesto que se aplica antes de que los grupos estén formados. Consideremos, por ejemplo, la siguiente revisión a una consulta formulada previamente:

Consulta: *Para cada tipo de edificio, ¿cuál es el nivel de calidad medio de los edificios con categoría 1? Considere sólo aquellos tipos de edificios que tienen un nivel de calidad máximo no mayor de 3.*

```
SELECT TIPO, AVG(NIVEL_CALIDAD)
FROM EDIFICIO
WHERE CATEGORÍA = 1
GROUP BY TIPO
HAVING MÁX(NIVEL_CALIDAD) <= 3
```

Resultado:

TIPO	
Comercio	1
Residencia	3

Obsérvese que, empezando por la cláusula FROM, las cláusulas en las instrucciones SQL se aplican en orden y la cláusula SELECT se aplica al final. Por tanto, la cláusula WHERE es aplicada a la relación EDIFICIO, con lo que se descartan todas las filas que tienen CATEGORÍA diferente de 1. Las filas que quedan se agrupan por TIPO, quedando todas las filas de un tipo en un mismo grupo. Esto crea un número de grupos —uno por cada valor de TIPO—. La cláusula HAVING se aplica entonces a cada uno de estos grupos y aquellos que tienen un nivel de calidad máximo por encima de 3 se eliminan. Finalmente, la cláusula SELECT se aplica a los grupos restantes.

Funciones integradas con subconsultas

Una función integrada (*built-in*) puede aparecer sólo en una cláusula SELECT o en una cláusula HAVING. Sin embargo, una cláusula SELECT que contenga una función integrada puede ser parte de una subconsulta. A continuación se verá un ejemplo de una tal subconsulta:

Consulta: *¿Qué trabajadores reciben una tarifa por hora mayor que la del promedio?*

```
SELECT NOMB_TRABAJADOR
FROM TRABAJADOR
WHERE TARIFA_HR >
   ( SELECT AVG(TARIFA_HR)
   FROM TRABAJADOR )
```

Resultado:

NOMB_TRABAJADOR
C. Nemo
P. Mason
C. Coulomb

Observe que esta subconsulta es una subconsulta no correlacionada, que produce precisamente un valor, el promedio de las tarifas por hora. La consulta principal selecciona un trabajador sólo si su tarifa por hora es mayor que el promedio que ha sido calculado por la subconsulta.

Las subconsultas correlacionadas se pueden usar junto con las funciones integradas:

Consulta: *¿Qué trabajadores reciben una tarifa por hora mayor de la del promedio de los trabajadores que dependen del mismo supervisor que él?*

En este caso, en lugar de calcular un solo promedio para todos los trabajadores, se debe calcular el promedio para cada grupo de trabajadores que informan a un mismo supervisor. Es más, este cálculo se debe llevar a cabo de nuevo por cada trabajador que esté siendo examinado en la consulta principal[6].

```
SELECT A.NOMB_TRABAJADOR
FROM TRABAJADOR A
WHERE A.TARIFA_HR >
   (SELECT AVG(B.TARIFA_HR)
   FROM TRABAJADOR B
   WHERE B.ID_SUPV = A.ID_SUPV )
```

[6] Observe que puede estar en un grupo al que ya alguna vez se le calculó el promedio (N. del T.).

Resultado:

A.NOMB_TRABAJADOR

C. Nemo
P. Mason
C. Coulomb

La cláusula WHERE de la subconsulta contiene la condición crucial de la correlación. Esta condición garantiza que el promedio se le calcule sólo a aquellos trabajadores que tengan el mismo supervisor que el trabajador que está siendo examinado por la consulta principal.

Operaciones del álgebra relacional

Como se ha señalado anteriormente y como debe probablemente haber observado, SQL parece tener el sabor del cálculo relacional con una lista de salida (la cláusula SELECT) y una instrucción de calificación (la cláusula WHERE). Sin embargo, SQL-92 ha implementado un número de operaciones del álgebra relacional, que se estudiarán a continuación. Específicamente, la unión, la intersección, la diferencia y la reunión están disponibles como operadores explícitos en SQL-92. Examinemos por orden cada uno de ellos.

unión compatible. Dos o más relaciones que tienen columnas equivalentes, tanto en número como en dominios.

Operadores UNION, INTERSECT y EXCEPT. Como en el álgebra relacional, la unión, la intersección y la diferencia son operaciones que se aplican a dos relaciones a la vez, que deben ser **unión compatible**. Este término tiene un significado ligeramente diferente en SQL. Dos relaciones son unión compatible si tienen el mismo número de columnas y las columnas respectivas en cada relación tienen tipos de datos compatibles. Es decir, tipos de datos que se puedan convertir de manera directa de uno a otro. Por ejemplo, dos tipos numéricos no necesitan ser idénticos, pero uno deberá ser convertible al otro.

Para la discusión de la unión, la intersección y la diferencia usaremos los mismos ejemplos utilizados en al Capítulo 6 para estas operaciones en el álgebra relacional (Figura 7.4). Esta figura muestra dos relaciones de vendedores, una consistente en vendedores que tienen algún jefe y otra de los vendedores que son jefes de alguien.

VEND_SUBORDINADO

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

VEND_JEFE

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

VENDEDOR

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
10	Rodney Jones	27	Chicago	10
14	Masaji Matsu	44	Tokyo	11
23	François Moire	35	Brussels	9
37	Elena Hermana	12	B.A.	13
39	Goro Azuma	44	Tokyo	10
27	Terry Cardon		Chicago	15
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

UNION. Operación que crea el conjunto unión de dos relaciones.

UNION. Supongamos que se quiere obtener una única relación con todos los vendedores. Se usa la instrucción SQL.

```
(SELECT * FROM VEND_SUBORDINADO)
UNION
(SELECT * FROM VEND_JEFE)
```

o la forma alternativa

```
SELECT *
FROM (TABLE VEND_SUBORDINADO UNION TABLE VEND_JEFE)
```

El resultado se muestra en la Figura 7.5. Como en el álgebra relacional, la unión de dos relaciones es una relación que contiene todas las filas que están en una relación o en la otra.

En este ejemplo ninguna fila aparece más de una vez, aun cuando aparezca en ambas relaciones. Sin embargo, si se usa la forma

```
(SELECT * FROM VEND_SUBORDINADO)
UNION ALL
(SELECT * FROM VEND_JEFE)
```

o la forma

```
SELECT *
FROM (TABLE VEND_SUBORDINADO UNION ALL TABLE VEND_JEFE)
```

entonces aquellas filas que aparezcan en ambas relaciones aparecerán duplicadas en la relación unión.

INTERSECT. Operación que crea el conjunto intersección de dos relaciones

INTERSECT (intersección). Supongamos que se quieren detectar aquellos vendedores que son jefes, pero que a su vez tienen algún jefe. En otras palabras, se quiere

VEND_SUBORD_JEFE

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
44	Albert Ige	27	Tokyo	12
35	Brigit Bovary	27	Brussels	11
12	Buster Sánchez	27	B.A.	10

la intersección de las filas que están en ambas relaciones. Se utiliza la instrucción SQL

```
(SELECT * FROM VEND_SUBORDINADO)
INTERSECT
(SELECT * FROM VEND_JEFE)
```

o la forma alternativa

```
SELECT *
FROM (TABLE VEND_SUBORDINADO INTERSECT TABLE VEND_JEFE)
```

El resultado se da en la Figura 7.6.

Como en la UNIÓN, ninguna fila aparece más de una vez en el resultado de un INTERSECT. Sin embargo, si hay *m* copias de una fila en una relación y *n* copias de la misma fila en la otra relación y $m <= n$, entonces la intersección de las dos relaciones tendrá *m* copias si se usa la sintaxis:

```
( SELECT * FROM VEND_SUBORDINADO)
INTERSECT ALL
(SELECT * FROM VEND_JEFE)
```

Al igual que con UNIÓN, la palabra clave ALL indica que las filas duplicadas deben considerarse como si fuesen filas independientes.

EXCEPT. Operación que crea el conjunto diferencia entre dos relaciones.

EXCEPT. Supongamos que se quieren identificar aquellos vendedores que no son subordinados de nadie. Lo que se qnisiera es sustraer la relación VEND_SUBORDINADO de la relación VEND_JEFE. En SQL-92 esta diferencia se lleva a cabo con la operación EXCEPT. En el caso de este ejemplo habría que usar la instrucción

```
(SELECT * FROM VEND_JEFE)
EXCEPT
(SELECT * FROM VEND_SUBORDINADO)
```

El resultado se muestra en la Figura 7.7.

Si se usa la sintaxis

```
(SELECT * FROM VEND_JEFE)
EXCEPT ALL
(SELECT * FROM VEND_SUBORDINADO)
```

y hay *m* copias de una determinada fila en VEND_JEFE y *n* copias de la misma fila en VEND_SUBORDINADO, donde $m > n$, entonces habrá $m - n$ copias de la fila en el resultado. Si $m <= n$, entonces no habrá copias de la fila en el resultado.

VEND_JEFE_JEFE

ID_VENDEDOR	*NOMB_VENDEDOR*	*ID_JEFE*	*OFICINA*	*%_COMISIÓN*
27	Terry Cardon		Chicago	15

Ya que se han introducido las ideas básicas de los operadores de SQL UNIÓN, INTERSECT y EXCEPT, analicémoslos un poco más de cerca. La restricción de que los dos operandos deben ser unión compatible parece algo fuerte. Después de todo, ¿cuán a menudo tenemos que ver con relaciones que tienen exactamente las mismas columnas? Echémosle un vistazo a algunas modificaciones sintácticas y a algunos ejemplos en donde las restricciones no son exactamente de esta manera. Consideremos la siguiente consulta:

Consulta: *¿Qué fontaneros comenzaron trabajos el 9 de octubre?*

```
(SELECT * FROM TRABAJADOR
WHERE OFICIO = 'Fontanero' )
INTERSECT CORRESPONDING BY (ID_TRABAJADOR)
(SELECT * FROM ASIGNACIÓN
WHERE FECHA_INICIO = '09/10' )
```

Resultado:

```
ID_TRABAJADOR
1520
```

Se ha destacado la parte de la instrucción SQL, que es importante para esta discusión. Las dos instrucciones SELECT definen dos relaciones que no son unión compatibles. Sin embargo, se ha tomado la intersección de estas dos considerando sólo las columnas que están en *ambas* relaciones. Estas columnas se especifican en la cláusula CORRESPONDING BY (correspondiendo por). En este ejemplo se ha considerado sólo la columna ID_TRABAJADOR. El sistema reducirá el resultado de ambos SELECT a los valores en la columna ID_TRABAJADOR, hará la intersección de estos dos conjuntos y éste será el resultado de la consulta. Es decir, el resultado será los ID_TRABAJADOR de aquellos trabajadores que son fontaneros y que tienen un trabajo asignado con fecha de inicio el 9 de octubre (que es exactamente lo que se pedía).

El mismo enfoque se usa para los operadores UNIÓN y EXCEPT. La cláusula CORRESPONDING BY va a continuación del operador para indicar las columnas que son compatibles en ambas relaciones y que son las que deben considerarse en la operación. Veamos algunos otros ejemplos.

Consulta: *¿Qué edificios son oficinas o tienen al trabajador 1412 asignado a él?*

```
(SELECT * FROM EDIFICIO
WHERE TIPO = 'Oficina' )
UNION CORRESPONDING BY (ID_EDIFICIO)
(SELECT * FROM ASIGNACIÓN
WHERE ID_TRABAJADOR = 1412 )
```

Resultado:

```
ID_EDIFICIO
312
210
111
460
435
515
```

Consulta: *¿Qué edificios que son oficinas no tienen al trabajador 1235 asignado a él?*

```
(SELECT * FROM EDIFICIO
WHERE TIPO = 'Oficina' )
EXCEPT CORRESPONDING BY (ID_EDIFICIO)
(SELECT * FROM ASIGNACIÓN
WHERE ID_TRABAJADOR = 1235 )
```

Resultado:

```
ID_EDIFICIO
210
111
```

Probablemente, el lector ya habrá notado que estas consultas son relativamente fáciles de formular. Se han identificado los subconjuntos apropiados de tuplas de las dos relaciones y luego se ha aplicado el operador conveniente UNIÓN, INTERSECT o EXCEPT. Esto añade un poder práctico considerable al lenguaje, ya que mucha gente debe encontrar más fácil de formular las consultas usando este enfoque. Considere, por ejemplo, la consulta anterior usando el "viejo" enfoque de SQL:

```
SELECT ID_EDIFICIO
FROM EDIFICIO
WHERE TIPO = 'Oficina' AND
   NOT EXISTS
     (SELECT *
     FROM ASIGNACIÓN
     WHERE EDIFICIO.ID_EDIFICIO = ASIGNACIÓN.ID_EDIFICIO
      AND ID_TRABAJDOR = 1235)
```

Esta solución utiliza una subconsulta correlacionada con el operador NOT EXISTS. No hay que ir muy lejos para darse cuenta de que la mayoría de la gente encontraría mucha mayor dificultad en formular esta solución que en la que usa EXCEPT. Observe que la consulta *podría* resolverse con la antigua sintaxis SQL, lo que no quiere decir que sea fácil.

El operador JOIN. SQL-92 tiene un número de operaciones explícitas de reunión (*join*): la reunión natural (*natural join*), la reunión interior (*inner join*), varias reuniones externas (*outer join*), la reunión unión (*union join*) y la reunión cruzada (*cross join*). Se considerarán solamente la reunión natural y la interior.

NATURAL JOIN. Operación que conecta las relaciones cuando las columnas comunes tienen iguales valores.

Natural join (*reunión natural*). Semánticamente, la reunión natural tiene el mismo significado en SQL que el álgebra relacional. Supongamos que se quiere hacer una reunión de las tablas TRABAJADOR y ASIGNACIÓN. Esto se logra con

```
TRABAJADOR NATURAL JOIN ASIGNACIÓN
```

El resultado de esta instrucción será la misma tabla que la que obtendríamos como resultado de la instrucción

```
SELECT T.ID_TRABAJADOR, NOMB_TRABAJADOR, TARIFA_HR, OFICIO, ID_SUPV,
   ID_EDIFICIO, FECHA_INICIO, NÚM_DÍAS
FROM TRABAJADOR T, ASIGNACIÓN
WHERE T.ID_TRABAJADOR = ASIGNACIÓN.ID_TRABAJADOR
```

excepto que la primera columna podría llamarse ID_TRABAJADOR en lugar de T.ID_TRABAJADOR. En general, la reunión natural causa que las dos tablas hagan la reunión en todas las columnas comunes, pero esas columnas se incluyen en el resultado una sola vez. En la relación resultante, las columnas comunes aparecen primero seguidas de las restantes columnas de la primera relación y luego las restantes columnas de la segunda relación.

JOIN USING. Operación que conecta las relaciones cuando las columnas comunes designadas tieuen iguales valores

Join USING (*reunión usando*). Supongamos que se tienen dos relaciones *A* y *B* y que éstas tienen columnas comunes *K*, *L*, *M* y *N*. Supongamos también que no se quiere hacer una reunión con todas las columnas comunes, sino sólo con las columnas *L* y *N*. Esto se puede hacer con las instrucción

```
A JOIN B USING (L, N)
```

Esta instrucción tiene el mismo efecto que la instrucción SELECT si en la cláusula SELECT se listan *L* y *N* primero seguidos de las restantes columnas de *A* y las restantes columnas de *B*. La cláusula WHERE de esta instrucción tendría la forma

```
WHERE A.L = B.L AND A.N = B.N
```

JOIN ON. Operación que conecta las relaciones cuando ocurre una condición.

Join ON (*reunión cuando*). Si se desea usar una condición más generalizada para la reunión de dos relaciones, se puede usar esta forma. Por ejemplo, supongamos que queremos hacer la reunión de TRABAJADOR consigo misma conectando ID_TRABAJADOR con ID_SUPV para obtener información sobre el supervisor de cada trabajador. Se puede hacer lo siguiente

```
TRABAJADOR T JOIN TRABAJADOR SUPV
ON T.ID_SUPV = SUPV.ID_TRABAJADOR
```

En este ejemplo se han creado dos copias de la relación TRABAJADOR y se les han dado los alias T y SUPV, respectivamente. La cláusula ON contiene una instrucción de condición de que el ID_SUPV de la copia T debe ser igual al ID_TRABAJADOR de la copia SUPV.

Se verán ahora algunas variaciones del ejemplo precedente. En lugar de usar tablas concretas o nombres de vistas de tabla, en una instrucción de reunión se pueden usar tablas derivadas de otras operaciones. Ilustraremos este enfoque con varias consultas

Consulta: *Obtener la asignación y los datos de los trabajadores Fontaneros que comenzaron sus trabajos el 9 de octubre.*

```
(SELECT * FROM TRABAJADOR WHERE OFICIO = 'Fontanero' )
NATURAL JOIN
(SELECT * FROM ASIGNACIÓN WHERE FECHA_INICIO = '09/10' )
```

Se puede observar que esta consulta es casi idéntica a la que se usó para ilustrar el operador anterior de INTERSECT. La diferencia aquí es que, en lugar de tener una lista de ID_TRABAJADORs, obtenemos toda la información relevante porque las tuplas de TRABAJADOR se reúnen con las tuplas de ASIGNACIÓN.

Consulta: *Obtener los datos sobre asignaciones, trabajadores y edificios para aquellos fontaneros que comenzaron los trabajos el 9 de octubre en edificios para residencias.*

```
(SELECT * FROM TRABAJADOR WHERE OFICIO = 'Fontanero' )
NATURAL JOIN
(SELECT * FROM ASIGNACIÓN WHERE FECHA_INICIO = '09/10' )
NATURAL JOIN
(SELECT * FROM EDIFICIO WHERE TIPO = 'Residencia' )
```

En este caso, las columnas de la reunión ID_TRABAJADOR e ID_EDIFICIO aparecerán primero en el resultado, seguidas de las restantes columnas de las relaciones TRABAJADOR, ASIGNACIÓN y EDIFICIO en ese orden. Si esto nos diese más información que la deseada, entonces se puede escribir esta instrucción completa en la cláusula FROM de una instrucción SELECT que especifique las columnas que se deseen:

```
SELECT NOMB_TRABAJADOR, FECHA_INICIO, ID_EDIFICIO, DIR_EDIFICIO
FROM ( SELECT * FROM TRABAJADOR WHERE OFICIO = 'Fontanero' )
NATURAL JOIN
( SELECT * FROM ASIGNACIÓN WHERE FECHA_INICIO = '09/10' )
NATURAL JOIN
( SELECT * FROM EDIFICIO WHERE TIPO = 'Residencia' )
```

Operaciones de modificación de la base de datos

SQL brinda tres operaciones para cambiar la base de datos, INSERT, UPDATE y DELETE, que permiten añadir filas, cambiar los valores en las filas y borrar filas, respectivamente, de una determinada relación en la base de datos. Se estudiará cada una de ellas por separado.

INSERT. Operación que causa que se añadan filas a una relación.

INSERT. La operación INSERT (insertar) permite insertar en una relación una fila mediante la especificación de los valores de cada una de las columnas de la fila, o insertar un grupo de filas especificando una consulta que nos daría el grupo de filas a insertar.

```
INSERT INTO ASIGNACIÓN (ID_TRABAJADOR, ID_EDIFICIO, FECHA_INICIO)
VALUES (1284, 485, 13/05)
```

Esta instrucción inserta una sola fila en ASIGNACIÓN. Los nombres de las columnas de las que se darán los valores respectivos se dan entre paréntesis a continuación del nombre de la tabla en la que se va a hacer la inserción. Puesto que se ha omitido la columna NÚM_DÍAS, se pondrá un valor nulo como valor de dicha columna en la fila a insertar.

Supongamos que se ha creado una nueva relación de nombre EDIFICIO_2 consistente en las columnas ID_EDIFICIO, TIPO y NIVEL_CALIDAD y que se desea poblar esta relación con las filas provenientes de EDIFICIO que tengan categoría 2. En este caso utilizaríamos la segunda forma de la instrucción INSERT.

```
INSERT INTO EDIFICIO_2
   SELECT ID_EDIFICIO, TIPO, NIVEL_CALIDAD
   FROM EDIFICIO
   WHERE CATEGORÍA = 2
```

UPDATE. Operación que cambia los valores de las columnas en las filas.

UPDATE. Las operaciones UPDATE (actualizar) se aplican a todas las filas que satisfacen la cláusula WHERE de la instrucción UPDATE. Si se quiere incrementar en un 5 por 100 el salario de todos los trabajadores que trabajan para el supervisor 1520, se necesitaría la instrucción siguiente:

```
UPDATE TRABAJADOR
SET TARIFA_HR = 1.05 * TARIFA_HR
WHERE ID_SUPV = 1520
```

Si no hubiese cláusula WHERE, entonces la operación se aplica a todas las filas de la relación. Por ejemplo, si se quisiera incrementar en el 5 por 100 el salario de *cada* trabajador, lo que habría que hacer es omitir la cláusula WHERE de la instrucción UPDATE anterior.

DELETE. Operación que quita filas de una relación.

DELETE. Las operaciones de DELETE se aplican también a todas las filas que satisfacen la cláusula WHERE en la instrucción DELETE. Si no hay cláusula WHERE, se borran todas las filas de la relación. Supóngase que todos los trabajadores del supervisor 1520 han sido despedidos y se quieren entonces borrar de la base de datos. La instrucción siguiente hace este trabajo:

```
DELETE FROM TRABAJADOR
WHERE ID_SUPV = 1520
```

Usar SQL con lenguajes de procesamiento de datos

El enfoque relacional de usar una simple instrucción para manipular conjuntos de filas en una relación es un avance sobre los métodos de manipulación de un registro-de-cada-vez de los lenguajes tradicionales. Sin embargo, puesto que SQL intenta ser utilizado en grandes organizaciones que usan lenguajes tradicionales, es necesario permitir que se puedan integrar instrucciones SQL en programas escritos en lenguajes tradicionales. Esto se logra con el SQL empotrado (*embedded SQL*).

SQL empotrado (*embedded SQL*). Un conjunto de instrucciones que permite que SQL sea utilizado con lenguajes de programación tradicionales.

El **SQL empotrado** (*embedded SQL*, también se conoce como *SQL embebido*) nos da un conjunto de instrucciones que se usan para incluir instrucciones SQL dentro de programas escritos en lenguajes como COBOL, C y Pascal (que son denominados **lenguaje anfitrión**). Estas instrucciones incluyen instrucciones de señalización que le notifican al preprocesador que lo que sigue se debe reemplazar por llamados a las rutinas del SGBD. También se incluyen facilidades especiales, llamadas *cursores*, las cuales permiten un procesamiento tipo *simple-registro* sobre los resultados de una consulta.

El siguiente es un ejemplo de código SQL empotrado:

```
EXEC SQL
   DECLARE TRAB_ASIG CURSOR
   FOR
     SELECT *
     FROM ASIGNACIÓN
     WHERE ID_TRABAJADOR = ID-TRABAJADOR-SOLICITADO
END-EXEC
```

lenguaje anfitrión (*host language*). Lenguaje de los programas en los cuales se pueden escribir las instrucciones SQL.

instrucciones de señalización (*flag statements*). Instrucciones SQL que se incrustan en un programa de aplicación para señalar dónde comienza o termina un conjunto de instrucciones SQL.

Este código podría empotrarse en un programa escrito en COBOL como lenguaje anfitrión. La primera línea (EXEC SQL) y la última línea (END-EXEC) son **instrucciones de señalización**[7] (*flag statements*) que indican que las líneas contenidas entre ellas dos son código SQL. El programa COBOL que contiene código SQL sería procesado por un preprocesador antes de ser compilado. El precompilador reconoce la instrucción de señalización y reemplaza éstas con llamadas a los subprogramas del SGBD que durante la ejecución tratan las instrucciones SQL. Cuando el programa COBOL se compile, el compilador ignora estas instrucciones de llamado y compila el resto del programa.

El resto del ejemplo contiene una instrucción SQL para declarar un cursor. Un **cursor** es como un archivo, cuyo contenido se genera en tiempo de ejecución. Observe que la definición de un cursor incluye una instrucción SELECT. La cláusula WHERE referencia a una columna de ASIGNACIÓN (la columna ID_TRABAJADOR) y establece que una fila debería seleccionarse si ID_TRABAJADOR es igual a una variable del programa COBOL (en este caso, ID-TRABAJADOR-SOLICITADO). No se selecciona ningún dato hasta que el cursor se *abre* por una instrucción independiente. La instrucción **OPEN** (abrir el cursor)

cursor. Una facilidad del SQL incrustado en la que se almacenan los resultados de una consulta SQL para su ulterior procesamiento.

instrucción OPEN cursor. Instrucciones SQL que causan que el SGBD procese una consulta y "almacene" el resultado en el cursor.

```
OPEN TRAB_ASIG
```

causará que el SGBD ejecute la instrucción SELECT. Las filas individuales que se ponen en el cursor al ejecutar el OPEN se pueden recuperar ejecutando **instrucciones FETCH**, que es análoga a una instrucción READ. También se brindan otras instrucciones para permitir actualizar y borrar datos de las tablas de la base de datos. La instrucción CLOSE quita los datos que "están en el cursor", de modo que éste puede ser abierto de nuevo asociado con nuevos datos que reflejen otros contenidos de la base de datos.

instrucción FETCH. Una instrucción que toma una fila a partir de un cursor que ya ha sido abierto.

[7] También denominada *instrucción bandera* (N. del T.).

El SQL empotrado brinda la interfaz necesaria para un uso exitoso de SQL en el procesamiento por lotes de gran escala o en programas en línea de grandes organizaciones de procesamiento de datos. Con el tiempo estas necesidades probablemente decrezcan con el surgimiento de nuevos lenguajes que sean capaces de hacer un uso total de las posibilidades de procesamiento de *una relación-a-la-vez* de SQL.

▼ Definición de vistas

tabla base. Una tabla que contiene información básica o real.

Al principio de este capítulo se mostró cómo se definen las tablas en un esquema de base de datos. Estas tablas se llaman **tablas base** porque contienen los datos básicos de la base de datos. Partes de estas tablas, así como información derivada de las mismas, pueden definirse en vistas de la base de datos. Estas vistas también se definen como parte del esquema de la base de datos.

vista. Una definición de una porción restringida de la base de datos.

Una **vista** (*view*) es como una "ventana" en una porción de la base de datos. Las vistas son útiles para mantener la confidencialidad al restringir el acceso a partes seleccionadas de la base de datos y para simplificar tipos de consultas que sean utilizados con frecuencia. Por ejemplo, para preservar la confidencialidad, podemos querer crear una vista que muestre la información sobre los trabajadores, excepto su tarifa por horas.

```
CREATE VIEW B_TRABAJADOR
AS SELECT ID_TRABAJADOR, NOMB_TRABAJADOR, OFICIO, ID_SUPV
   FROM TRABAJADOR
```

En este ejemplo, B_TRABAJADOR es el nombre de vista que se ha creado. El nombre de la vista puede estar seguido de los nombres de las columnas en la vista encerrados entre paréntesis. En este caso se han omitido los nombres de columnas, por lo que las columnas de la vista tomarán los mismos nombres de las columnas de la relación de la cual se forma la vista. La parte de esta instrucción de vista que sigue a la palabra "AS" se denomina **especificación de consulta** (*query specification*). Cualquier consulta que sea legal puede aparecer en la definición de una vista.

especificación de consulta. Definición de una consulta que se usa en una definición de vista, declaración de cursor u otra instrucción.

El sistema no genera realmente los valores de los datos para B_TRABAJADOR hasta que no se haga acceso a éste. En ese momento se ejecuta la especificación que define a B_TRABAJADOR, creándose B_TRABAJADOR a partir de los datos que existen en TRABAJADOR en el momento de la ejecución. Por tanto, los datos de una vista cambian dinámicamente en la medida que cambien los datos en su tabla base subyacente.

Supongamos que con frecuencia estamos interesados en información sobre los electricistas, los edificios a los que están asignados y las fechas de comienzo de las asignaciones. Esta definición de vista servirá para estos intereses:

```
CREATE VIEW ELEC_ASIGNACIÓN AS
   SELECT NOMB_TRABAJADOR, ID_EDIFICIO, FECHA_INICIO
   FROM TRABAJADOR, ASIGNACIÓN
   WHERE OFICIO = 'Electricista' AND
     TRABAJADOR.ID_TRABAJADOR = ASIGNACIÓN.ID_TRABAJADOR
```

Si se hiciese acceso a ELEC_ASIGNACIÓN, el sistema primero generaría los valores de sus datos a partir de la definición anterior de la vista. Para la base de datos de nuestro ejemplo, la vista ELEC_ASIGNACIÓN se vería como sigue:

```
ELEC_ASIGNACIÓN
NOMB_TRABAJADOR    ID_EDIFICIO    FECHA_INICIO
M. Faraday         312            10/10
M. Faraday         515            17/10
C. Coulomb         435            08/10
C. Coulomb         460            23/10
```

Se pueden consultar las vistas. Supongamos que estamos interesados en conocer los electricistas asignados al edificio 435.

Consulta: *¿Quiénes son los electricistas asignados al edificio 435 y cuándo comenzaron a trabajar?*

```
SELECT NOMB_TRABAJADOR, FECHA_INICIO
FROM ELEC_ASIGNACIÓN
WHERE ID_EDIFICIO = 435
```

Resultado: C. Coulomb 08/10

El sistema primero crea la relación como se explicó y luego aplica la consulta a la relación.

También, para definir una vista, se puede usar el resultado de una operación de agrupación. Echémosle otro vistazo a la consulta definida al principio que nos daban los supervisores junto con la tarifa máxima de los trabajadores que informaban a estos supervisores:

```
CREATE VIEW SALARIO_MÁX (ID_SUPV, MÁX_TARIFA_HR) AS
   SELECT ID_SUPV, MÁX(TARIFA_HR)
   FROM TRABAJADOR
   GROUP BY ID_SUPV
```

Observe que en este ejemplo se han incluido los nombres de las columnas de la vista. Éstos siguen inmediatamente a continuación del nombre de la vista (SALARIO_MÁX). Fue necesario incluir nombres de columna en este caso porque una de las columnas es el resultado de un cálculo usando una función integrada (MÁX) y no tiene nombre.

Se puede consultar esta vista para determinar, por ejemplo, qué supervisores tienen trabajadores que reciben una tarifa por horas por encima de una cierta cantidad.

Consulta: *¿Qué supervisores tienen trabajadores que tienen una tarifa por hora por encima de los $12?*

```
SELECT ID_SUPV
FROM SALARIO_MÁX
WHERE MÁX_TARIFA_HR > 12
```

Al procesar esta consulta, el sistema crea primero el resultado de ejecutar la especificación de la vista:

```
SALARIO_MÁX
ID_SUPV          MÁX_TARIFA_HR

1311             15,50
1520             13,75
2920             10,00
3231             17,40
```

Luego le aplica la consulta como tal a este resultado:

```
ID_SUPV
1311
1520
3231
```

Restricciones sobre consultas y actualizaciones de vistas

Una vista que sea definida con una cláusula GROUP BY en la especificación de la consulta se denomina una *vista de grupos* (*grouped view*). En SQL ANSI estándar hay varias

limitaciones para las consultas sobre vistas de grupos. Estas limitaciones son (Melton y Simon, 1993, p. 196):

1. La cláusula SELECT de una consulta sobre una vista de grupos no puede tener una función integrada.
2. No se puede hacer una reunión (*join*) de una vista de grupos con ninguna otra vista o tabla.
3. Una consulta sobre una vista de grupos no puede tener a su vez cláusulas GROUP BY o HAVING.
4. Una subconsulta no puede referirse a una vista de grupos.

Los usuarios desean con frecuencia actualizar la base de datos haciendo referencia a las vistas. Una vista se puede actualizar sólo si su especificación de consulta cumple con (Date y Darwen, 1994, p.174):

1. Tiene una cláusula SELECT que contiene sólo nombres de columnas (es decir, no tiene ningún cálculo ni ningún valor literal) y no contiene la palabra clave DISTINCT;
2. Tiene sólo una referencia a tabla en la cláusula FROM (esta referencia no puede incluir ninguna de las operaciones del álgebra relacional UNIÓN, INTERSECT, EXCEPT, JOIN). Dicha tabla en la cláusula FROM debe ser o una tabla base o una vista actualizable;
3. No tiene subconsulta en su cláusula WHERE que refiera a la tabla en la cláusula FROM de la consulta principal;
4. No contiene una cláusula GROUP BY o HAVING.

Por supuesto que las tablas base sí pueden actualizarse sin estas restricciones. Si una vista no permite actualizaciones, entonces se puede acceder a las tablas base propiamente dichas para lograr la actualización deseada.

El esquema de información

metadato. Información descriptiva sobre las bases de datos.

esquema de información. Esquema en un catálogo que contiene metadatos.

Uno de los criterios de Codd para decir que un SGBD sea completamente relacional (Codd, 1985c) es que la información del sistema que describe la base de datos esté mantenida también en tablas relacionales, al igual que otros valores de datos. Esta información descriptiva, o **metadato**, se mantiene normalmente en un diccionario de datos. Una versión de este diccionario de datos se le llama **esquema de información**. El esquema de información de SQL-92 cumple, en sus elementos esenciales, con este requisito, puesto que está estructurado como un conjunto de vistas (posiblemente sólo hipotéticas) de tablas, cada fila de las cuales nos da información descriptiva sobre los objetos de la base de datos, tales como una tabla, una columna o un dominio (ver Date y Darwen, 1994). En esta sección se dará una descripción breve del esquema de información de SQL-92.

Como se señaló al principio del capítulo, SQL-92 define los catálogos como colecciones con nombre de esquemas de base de datos. Cada esquema en un catálogo tiene un cierto "dueño", quien posiblemente originó el esquema. Si un catálogo determinado tiene un cierto número de esquemas que a su vez están definidos y son posesión de una variedad de usuarios, entonces el catálogo también contiene un esquema especial, el INFORMATION_SCHEMA, que describe a todos los esquemas del catálogo, incluyéndolo a él mismo. De acuerdo a SQL-92 estándar, este esquema de información realmente necesita no más que un conjunto de vistas de algunos diccionarios de datos que dependen de la implementación. Puesto que hay muchos SGBD comerciales diferentes en el mercado, cada uno de los cuales usa diferentes enfoques para definir metadatos, tiene sentido (como señalan Date y Darwen) que el estándar requiera solamente que el INFORMATION_SCHEMA sea un conjunto de vistas de las tablas de metadatos que realmente el SGBD tenga definidas.

Si un SGBD intenta cumplir con el estándar de SQL-92 puede definir los metadatos a su manera y garantizar solamente definir el INFORMATION_SCHEMA en la forma estándar. Puesto que el usuario no puede, de ninguna manera, actualizar directamente las tablas de metadatos, sólo hace falta que el esquema de información ofrezca las vistas necesarias para que el usuario pueda consultar y ver cómo está estructurada la base de datos.

Las vistas en el esquema de información tienen nombres y funciones predefinidas. Antes de dar una lista de las vistas señalemos lo siguiente. Dado que el esquema es una colección de *vistas*, cualquier usuario que las acceda verá solamente el resultado de una consulta sobre un grupo invisible de tablas base. Por lo tanto, para dar más seguridad, la vista siempre estará definida con respecto al usuario actual del sistema, cuya identidad está en la variable CURRENT_USER. A continuación se dará una lista de estas vistas junto con una explicación breve de sus funciones (ver Date y Darwen, 1994, y Melton y Simon, 1993, para más detalles).

INFORMATION_SCHEMA_CATALOG_NAME. Realmente es una tabla base formada por una sola fila y una sola columna que contiene el nombre del esquema.

SCHEMATA. Contiene los nombres de todos los esquemas creados por el usuario actual.

DOMAINS. Da todos los dominios accesibles por el usuario actual.

TABLES. Da las tablas y las vistas accesibles por el usuario actual.

VIEWS. Lista las vistas accesibles por el usuario actual.

COLUMNS. Identifica las columnas de las tablas listadas en la vista TABLES.

TABLE_PRIVILEGES

COLUMN_PRIVILEGES

USAGE_PRIVILEGES

DOMAIN_CONSTRAINTS. Contiene las restricciones de todos los dominios creados por el usuario actual.

TABLE_CONSTRAINTS. Da las restricciones sobre las tablas bases creadas por el usuario actual.

REFERENCIAL_CONSTRAINTS. Lista las restricciones referenciales propiedad del usuario actual.

CHECK_CONSTRAINTS. Lista las restricciones de chequeo propiedad del usuario actual.

KEY_COLUMN_USAGE. Lista todas las columnas que se usan en claves candidatas o en claves externas en tablas creadas por el usuario actual.

ASSERTIONS

CHARACTER_SETS

COLLATIONS

TRANSLATIONS

VIEW_TABLE_USAGE. Identifica la tablas sobre las cuales dependen las vistas propiedad del usuario actual.

VIEW_COLUMN_USAGE. Identifica las columnas de las que dependen las vistas propiedad del usuario actual.

CONSTRAINT_TABLE_USAGE. Lista las tablas, propiedad del usuario actual, que están referidas por restricciones.

CONSTRAINT_COLUMN_USAGE. Lista las columnas de las tablas, propiedad del usuario actual, que están referidas en las restricciones.

COLUMN_DOMAIN_USAGE. Da las columnas que están definidas en términos de dominios que son propiedad del usuario actual.

SQL_LANGUAGES

Las ventajas de la estructura relacional del esquema del sistema de información deben quedar claras. Se puede usar un lenguaje de consulta muy potente para formular una gran variedad de preguntas sobre la estructura de las base de datos del sistema. Los primeros sistemas que no eran relacionales no tenían automáticamente estas ventajas.

En este capítulo se han estudiado aspectos de las implementaciones de bases de datos relacionales con el lenguaje de consulta SQL. Se revisó de forma concisa el desarrollo histórico de los sistemas SQL. Se mostró cómo se pueden definir los esquemas de base de datos SQL. También se analizó en detalle el lenguaje de manipulación de datos de SQL y se destacó la definición de vistas. Por último se vio una breve introducción al esquema de información del sistema.

SQL fue creado por IBM durante los años setenta como parte del proyecto System R. Desde aquel tiempo ha sido implementado por una gran variedad de empresas comerciales, como el lenguaje de sus SGDB relacionales. Está disponible tanto en entornos de grandes computadores como en entornos de microcomputadores y ha devenido en el estándar ANSI para lenguajes relacionales.

Los catálogos de SQL son colecciones con nombre de los esquemas, cada uno incluye un número de esquemas creados por el usuario, así como un esquema de información que contiene metadatos de los esquemas en el catálogo. Los esquemas de base de datos en SQL se definen con las órdenes CREATE SCHEMA, CREATE DOMAIN y CREATE TABLE. Estas órdenes definen los esquemas específicos y sus dueños, los dominios, las tablas y las columnas. Los dominios se usan como tipos de datos en las definiciones de las columnas. También se pueden definir restricciones tanto para columnas individuales como para tablas. Las restricciones incluyen restricciones de claves primarias y claves externas. También se pueden definir restricciones CHECK más generales.

La manipulación de datos de SQL incluye todas las capacidades del álgebra relacional definidas por Codd, aunque su formulación original está basada en la estructura del cálculo relacional. Sin embargo, SQL-92 incluye operaciones que son más directamente del álgebra relacional, tales como la reunión (join), la unión, la intersección y la diferencia. Adicionalmente, SQL brinda capacidades para agrupar las filas sobre valores en común de una determinada columna y proporciona también funciones que permiten hacer cálculos estadísticos sobre estos grupos. La manipulación de datos también permite la inserción, la actualización y el borrado de filas individuales, así como de grupos de filas. Por último, se pueden escribir instrucciones SQL dentro de programas tradicionales escritos en COBOL, C o Pascal. Mediante los cursores, las instrucciones SQL pueden generar los resultados de consultas, de modo que puedan ser procesados fila por fila por el programa escrito en el lenguaje anfitrión.

Las vistas se definen en el esquema de la base de datos usando el lenguaje de consulta de SQL. Las vistas pueden consultarse y, bajo determinadas restricciones, pueden también actualizarse.

El esquema de información del sistema en un sistema verdaderamente relacional debería estar estructurado como una base de datos relacional. Hemos visto cómo el esquema de información de SQL-92 satisface este requisito. Los sistemas que tienen los esquemas de información con esta estructura tienen distintas ventajas. En particular el hecho de que el esquema de información puede ser consultado usando el mismo lenguaje que se usa para las consultas normales a la base de datos.

1. Defina con sus propias palabras cada uno de los términos siguientes:
   a. catálogo
   b. propietario de esquema
   c. definición de dominio
   d. literal cadena de caracteres
   e. operaciones de comparación
   f. conectores booleanos
   g. consulta externa

h. consulta correlacionada
i. función integrada
j. DISTINCT
k. GROUP BY
l. INSERT
m. DELETE
n. SQL empotrado (*embebido*)
o. lenguaje anfitrión
p. cursor
q. FETCH
r. vista
s. especificación de consulta
t. esquema de información
u. valor por defecto
v. clave externa recursiva
w. unión compatible
x. UNIÓN
y. EXCEPT
z. NATURAL JOIN

2. Describa brevemente los primeros desarrollos de SQL. ¿Qué fabricante fue el primero en lanzar una versión comercial de SQL? Describa varios sistemas en los cuales está disponible SQL.

3.
   a. ¿Cuáles son las órdenes de SQL-92 que se usan para definir un esquema de una base de datos?
   b. ¿Cómo se puede definir una clave de una sola columna o de multi-columnas? ¿Cómo se define una clave externa? ¿Como se define un valor por defecto? ¿Cuál es la diferencia entre una restricción de columna y una restricción de tabla? ¿Cómo se pueden definir restricciones más generalizadas?
   c. Indique diez tipos de datos diferentes disponibles en SQL-92.

4. Describa lo que puede aparecer en cada una de las cláusulas siguientes de SQL:
   a. SELECT
   b. FROM
   c. WHERE
   d. GROUP BY
   e. HAVING
   f. ORDER BY

5. Describa la acción de cada una de las operaciones:
   a. UNIÓN
   b. INTERSECT
   c. EXCEPT
   d. NATURAL JOIN
   e. JOIN USING
   f. JOIN ON

6. ¿Cómo se usan las especificaciones de consulta en cada una de las operaciones de modificación de datos?

7. Describa cómo trabaja el cursor.

8. ¿Qué es una vista de una base de datos y cómo se puede usar?

9. Describa cómo se puede usar SQL con el esquema de información para obtener información sobre una base de datos.

1. Conecte cada término con su definición.

—*alias*	a.	Contiene datos básicos o reales
—*reunión ON*	b.	Nombre alternativo que se le da a una relación
—*cláusula FROM*	c.	Lista las tablas existentes que son referidas por la consulta
—*producto cartesiano*	d.	Evalúa verdadero si el conjunto resultante es *no* vacío
—*consulta no correlacionada*	e.	Da la condición para seleccionar filas de las tablas identificadas
—*operador EXISTS*	f.	Una función integrada
—*definición de esquema*	g.	Consulta que involucra a sólo una tabla de la base de datos
—*restricción*	h.	Operación que cambia los valores de las columnas en las filas
—*cláusula HAVING*	i.	Evalúa a verdadero si el conjunto resultante es vacío
—*reunión USING*	j.	Provoca que el SGBD procese la consulta de un cursor y almacene el resultado en el cursor
—*consulta simple*	k.	Resultado de parear cada fila en una tabla con *todas* las filas en otra tabla
—*operador NOT EXISTS*	l.	Su valor no depende de ninguna consulta externa
—*INTERSECT*	m.	Descripción de una base de datos al SGBD
—*instrucción de señalización*	n.	Pone condiciones sobre los grupos
—*tabla base*	o.	Señal de comienzo o fin de un conjunto de instrucciones SQL.
—*carácter comodín (wild char)*	p.	Puede tener entradas duplicadas
—*cláusula WHERE*	q.	Identifica las columnas que se desean en la consulta
—*abrir cursor*	r.	Información descriptiva sobre las bases de datos
—*multiconjunto*	s.	Una consulta dentro de una consulta
—*función de conjunto*	t.	Una regla que restringe los valores en una base de datos
—*metadato*	u.	Símbolos especiales que valen por una cadena de caracteres cualesquiera
—*UPDATE*	v.	Operación que conecta las relaciones cuando las columnas comunes designadas tienen iguales valores
—*subconsulta*	w.	Operación que conecta las relaciones cuando ocurre una condición
—*cláusula SELECT*	x.	Operación que crea el conjunto intersección de dos relaciones

2. Escriba órdenes SQL para definir los dominios siguientes:
   a. Un dominio de salario cuyos elementos tengan cuatro dígitos con dos a la derecha del punto decimal. Su valor por defecto es $6.00, el valor debe ser al menos $5,00, pero no mayor que $30,00.
   b. Un dominio de fechas con todas las fechas después del 1 enero de 1980, pero antes del 1 enero de 2000.
   c. Un dominio de horario cuyos elementos estén entre las 8 a.m. y las 5 p.m., inclusive

3. Escriba las órdenes SQL para crear un esquema de base de datos para el siguiente esquema relacional:

```
CLIENTE (ID_CLIENTE, NOMB_CLIENTE, RENTA_ANUAL, TIPO_CLIENTE)
ID_CLIENTE debe estar entre 100 y 10.000
RENTA_ANUAL tiene valor por defecto $20.000
TIPO_CLIENTE debe ser fabricante, mayorista, o minorista

EMBARQUE (EMBARQUE_#, ID_CLIENTE, PESO, CAMIÓN_#, DESTINO, FECHA
EMBARQUE)
Clave externa: ID_CLIENTE REFERENCIA A CLIENTE, borra en cascada
Clave externa: CAMIÓN_# REFERENCIA A CAMIÓN, al borrar pone a nulo
Clave externa: DESTINO REFERENCIA A CIUDAD, al borrar pone a nulo
PESO debe estar bajo 1000 y con valor por defecto 10

CAMIÓN (CAMIÓN_#, NOMB_CHÓFER)

CIUDAD (NOMB_CIUDAD, POBLACIÓN)
```

Use el esquema siguiente, tomado del ejercicio 3, para expresar en SQL las consultas que se expresan a continuación.

```
CLIENTE (ID_CLIENTE, NOMB_CLIENTE, RENTA_ANUAL, TIPO_CLIENTE)
EMBARQUE (EMBARQUE_#, ID_CLIENTE, PESO, CAMIÓN_#, DESTINO, FECHA
EMBARQUE)
CAMIÓN (CAMIÓN_#, NOMB_CHÓFER)
CIUDAD (NOMB_CIUDAD, POBLACIÓN)
```

4. *Consultas simples*
   a. ¿Cuál es el nombre del cliente 433?
   b. ¿Cuál es la ciudad destino del embarque #3244?
   c. ¿Cuáles son los números de los camiones que han llevado paquetes (embarques) por encima de las 100 libras?
   d. Dé todos los datos de los embarques de más de 20 libras.
   e. Cree una lista por orden alfabético de los clientes con renta anual de más de $10 millones.
   f. ¿Cuál es el ID del cliente Wilson Brothers?
   g. Dé los nombres y la renta promedio mensual de los clientes que tienen renta anual que excede los $5 millones, pero que es menor de los $10 millones.
   h. Dé los IDs de los clientes que han enviado paquetes a Chicago, St. Louis o Baltimore.
   i. Dé los nombres de los clientes que han enviado embarques a las ciudades cuyo nombre empieza con 'C'.
   j. Dé los nombres de los clientes que han enviado embarques a las ciudades cuyo nombre termina con 'City'.
   k. Dé los nombres de los clientes que tienen una 'D' como tercera letra del nombre.
   l. Dé los nombres de todos los clientes que sean minoristas.

5. *Reuniones (joins)*
   a. ¿Cómo se llaman los clientes que han enviado paquetes a Sioux City?
   b. ¿A cuáles destinos han hecho envíos las compañías con renta anual menor del millón?
   c. ¿Cuáles son los nombres y las poblaciones de las ciudades que han recibido embarques que pesen más de 100 libras?
   d. ¿Cuáles son los clientes que tienen más de $5 millones de renta anual y que han enviado embarques de menos de 1 libra?
   e. ¿Quiénes son los clientes que tienen sobre los $5 millones en renta anual y que han enviado embarques de menos de 1 libra o han enviado embarques a San Francisco?
   f. ¿Quiénes son los chóferes que han conducido embarques de clientes que tienen renta anual sobre los $20 millones a ciudades con población por encima del millón?

6. *Subconsultas*
   a. Indique las ciudades que han recibido embarques de clientes que tienen más de $15 millones de renta anual.
   b. Indique los nombres de los chóferes que han transportado embarques que pesan más de 100 libras.
   c. Indique el nombre y la renta anual de los clientes que han enviado embarques que pesan más de 100 libras.

7. *NOT EXISTS*
   a. Indique los clientes que han tenido embarques transportados por cada camión.
   b. Indique las ciudades que han recibido embarques de todos los clientes.
   c. Indique los chóferes que han transportado embarques a cada una de las ciudades.

8. *Funciones integradas*
   a. ¿Cuál es el peso promedio de los embarques?
   b. ¿Cuál es el peso promedio de los embarques que van para Atlanta?
   c. ¿Cuántos embarques ha enviado el cliente 433?
   d. ¿Cuáles ciudades de la base de datos tienen la menor y la mayor población?
   e. ¿Cuál es el peso total de los paquetes (embarques) transportados en el camión 82?
   f. Dé una lista de los clientes y de la renta anual para aquellos clientes que están en la base de datos (Sugerencia: Usar una subconsulta.)
   g. Dé una lista de los clientes para los que todos sus embarques han pesado por encima de las 25 libras.
   h. Dé una lista de los clientes que hacen todos sus envíos a una sola ciudad. (Note que la ciudad no tiene que ser la misma para cada cliente.)

9. *GROUP BY y HAVING*
   a. Para cada cliente, ¿cuál es el peso medio de los paquetes enviados por él?
   b. Para cada ciudad, ¿cuál es el peso máximo de un paquete que haya sido enviado a dicha ciudad?
   c. Para cada ciudad con población por encima del millón, ¿cuál es el peso menor de un paquete enviado a dicha ciudad?
   d. Para cada ciudad que haya recibido al menos diez paquetes, ¿cuál es el peso medio de los paquetes enviados a dicha ciudad?

10. *Operaciones del Álgebra Relacional*
    a. Los clientes que son fabricantes o que han enviado algún paquete a St. Louis.
    b. Las ciudades de población por encima del millón y que hayan recibido paquetes de 100 libras de parte del cliente 311.
    c. Camiones manejados por Jake Stinson que nunca hayan transportado un embarque hacia Denver.
    d. Clientes con una renta anual sobre los $10 millones y que hayan enviado paquetes de menos de 1 libra a ciudades con población menor de 10.000.

11. *Operaciones de modificación de la base de datos*
    a. Añada el camión 95 con el chófer Winston.
    b. Borre de la base de datos todas las ciudades con población por debajo de 5.000. No se olvide de actualizar también la relación EMBARQUE.
    c. Convierta el peso de cada envío a kilogramos, dividiendo el peso por 2.2.

12. *Cursores*

    Cree la declaración de un cursor que identifique toda la información sobre clientes que hayan enviado embarques a una ciudad con población mayor que 500.000.

Vistas

13. Cree vistas para cada uno de los casos:
    a. Clientes con renta anual por debajo del millón.
    b. Clientes con renta anual entre $1 millón y $5 millones.
    c. Clientes con renta anual por encima de los $5 millones.

14. Use las vistas anteriores para responder a las consultas siguientes:
    a. ¿Qué chóferes han transportado embarques a Los Angeles provenientes de clientes con renta sobre los $5 millones?
    b. ¿Cuál es la población de las ciudades que han recibido embarques de clientes con renta entre $1 millón y $5 millones?
    c. ¿Qué chóferes han transportado embarques de clientes con renta por debajo del millón y cuál es la población de las ciudades hacia las que se han enviado dichos embarques?

1. Escriba un artículo de investigación sobre el desarrollo de SQL desde sus primeras etapas en IBM. Discuta sus primeras versiones comerciales en Oracle e IBM, su categoría como estándar ANSI y las versiones más recientes para mainframe y microcomputadores.

2. Escriba un artículo de investigación que critique el estándar ANSI de 1992 para SQL. Compare este estándar con dos o tres productos comerciales.

3. Estudie en detalle el SQL empotrado. Determine cómo es su interfaz con uno o dos lenguajes anfitriones.

4. Estudie los esquemas de información de dos sistemas diferentes de gestión de base de datos relacionales. ¿Cómo podrían usarse con SQL para brindar información a los usuarios?

CAPÍTULO

8

IMPLEMENTACIÓN RELACIONAL CON LENGUAJES DE CONSULTA GRÁFICOS

Introducción
Manipulación de datos
- Consultas simples
- Consultas de múltiples tablas
- Funciones integradas o predefinidas
- GROUP BY
- Operaciones de modificación de la base de datos
  - Insertar (Insert)
  - Actualizar (Update)
  - Borrar (Delete)

PARADOX PARA WINDOWS
- Definición y entrada de datos
- Manipulación de datos
- Seleccionar atributos y filas
- Consultas conjuntivas y disyuntivas
  - Cálculos
  - Enlazando tablas
- Lenguaje de aplicación
  - Instrucciones condicionales
  - Instrucciones iterativas

Resumen
Preguntas de repaso
Problemas y ejercicios
Proyectos y cuestiones profesionales

"Irv, ¿cómo quieres trabajar con el sistema ahora que se ha añadido Query-by-Example?"

Annette Chang está interesada en las reacciones de los usuarios con el nuevo lenguaje de consulta de base de datos relacional que ha instalado recientemente la Compañía Constructora Premier. Está hablando con Irv Bernstein, inspector de la compañía.

"Annette, honestamente te digo que nunca había pensado que un lenguaje de computación fuese tan fácil de usar. Cuando comprendí cómo se despliegan las tablas de datos fue fácil ver cómo construir consultas. Por supuesto, por ahora sólo he estado utilizando consultas simples, pudiera que no lo encontrase tan fácil a medida que mis necesidades sean más sofisticadas."

"Yo creo que te parecerá que para la mayoría de las cosas que quieres hacer, el lenguaje es fácil de entender. Te ayudaré con mucho gusto con cualquier tipo de consultas que creas que no puedes resolver; pero pienso que en cualquier caso en poco tiempo podrás valerte por tu cuenta."

Query-by-Example[1] es una versión gráfica del lenguaje de datos relacional SQL. Particularmente encaja bien en las necesidades de los usuarios típicos. En este capítulo nos familiarizaremos con la mayoría de los aspectos de este lenguaje, así como con una implementación del mismo para microcomputadoras: PARADOX para Windows. Después de leer este capítulo se podrá:

- Analizar algunos hechos históricos relativos a la teoría y las implementaciones comerciales del *Query-by-Example* (QBE).
- Desarrollar las soluciones en QBE para una amplia variedad de consultas.
- Hacer modificaciones a una base de datos mediante QBE.
- Evaluar los puntos fuertes y débiles del QBE como lenguaje relacional.
- Ser capaz de desarrollar aplicaciones usando PARADOX para Windows.

▼ Introducción

Query-by-Example[1] (QBE) fue desarrollado en los años setenta en el centro de investigación Thomas J. Watson de IBM (Zloof, 1975; Date, 1986). El tiempo de su desarrollo transcurre en paralelo con el de SQL, y los dos lenguajes son lógicamente similares. QBE fue lanzado como producto comercial desde hace ya algún tiempo y es comúnmente utilizado como herramienta de consulta por usuarios finales. Típicamente se soporta QBE cuando se soporta también SQL y las tablas que se acceden por QBE fueron definidas mediante las facilidades de definición de tablas de SQL.

Ambos lenguajes están soportados por el *Query Management Facility* (*Facilidad para Manipulación de Consultas*) (QMF) que ofrece IBM. QMF es un programa guión *(shell)* que da una interfaz de usuario al QBE y al SQL. Los usuarios interactúan con QMF a través de terminales en línea usando menús en pantalla y teclas de función. El usuario selecciona uno de los dos lenguajes y formatos y ejecuta las consultas en el lenguaje escogido. QMF proporciona medios para imprimir los resultados de estas consultas en informes con formatos que incluyen encabezamientos y otros aspectos estándares.

QBE también está soportado por otras compañías. Borland, por ejemplo, ofrece PARADOX como una versión con la funcionalidad completa del QBE, que incluye la definición de tablas. Aunque la sintaxis es ligeramente diferente de la de IBM, el lenguaje es básicamente el mismo. Lotus 1-2-3 también nos da un lenguaje simple de consulta de base de datos, qne usa un enfoque similar al del QBE. Debido a que el QBE permite que los

[1] Pudiera traducirse como *Consultando-por-ejemplos*. Por ser un nombre ya establecido en la industria y la literatura, se dejará el original en inglés (N. del T.).

usuarios no técnicos formulen fácilmente sus consultas, éste deberá formar una parte importante de los sistemas comerciales de base de datos por algún tiempo.

En este capítulo se estudiarán los recursos de manipulación del QBE en algún detalle. Se utilizará como estándar la implementación comercial de IBM. También se verá PARADOX para Windows, que es una implementación importante de QBE ampliamente utilizada en los sistemas microcomputadoras.

▼ Manipulación de datos

lenguaje textual. Un lenguaje de computadoras cuyas instrucciones consisten de cadenas de caracteres.

Los lenguajes tradicionales de computadores son **textuales**, es decir, la formulación de las soluciones se expresa en textos formados por cadenas de caracteres. Sin embargo, QBE es un lenguaje **gráfico** que estructura las soluciones a las consultas por medio de representaciones pictóricas de las tablas de las bases de datos. Ubicando símbolos en los lugares adecuados en las columnas de las tablas, el usuario puede especificar las condiciones de selección de la consulta, la agrupación, el despliegue de los datos y las operaciones de actualización de la base de datos.

lenguaje gráfico. Un lenguaje de computadoras que usa representaciones pictóricas para resolver los problemas.

Aunque QBE es un lenguaje gráfico, tiene muchas similitudes estructurales con SQL. Ambos lenguajes soportan similares expresiones condicionales, funciones integradas, agrupación, consulta multi-tablas y otras por el estilo. También hay áreas donde los dos lenguajes son diferentes. En particular, algunos recursos del SQL no aparecen en el QBE. En la medida en que la presentación avance se señalarán las similitudes y diferencias más importantes. Salvo que se señale explícitamente, se usarán las mismas consultas para ilustrar QBE que las que se usaron para SQL. Los valores resultantes de los datos serán por tanto los mismos.

Puesto que la mayoría de los sistemas con QBE usan bases de datos que han sido definidas usando las órdenes de definición de SQL, no se analizará la definición de datos en QBE.

Consultas simples

Al igual que con SQL, primero se estudiarán las **consultas simples** que afectan a una sola tabla de la base de datos.

consulta simple. Una consulta que involucra a una sola tabla de la base de datos.

Como lenguaje gráfico, QBE debe proporcionar medios para que el usuario interactúe gráficamente con la base de datos. Esto se hace desplegando un esqueleto de tablas o, en la terminología de QBE, **tablas ejemplos**. Ejecutando una orden, el usuario puede provocar que se despliegue en pantalla una tabla ejemplo para cada tabla en la base de datos. En la Figura 8.1 se muestra un ejemplo simple de base de datos. La Figura 8.2 muestra tablas ejemplos para cada una de las tres tablas en la base de datos. En cada caso el nombre de la tabla aparece en la primera columna, seguido en orden por el nombre de cada una de las columnas de la tabla. Un espacio en blanco se deja debajo de cada uno de estos nombres de tabla y columnas para permitir la entrada de la consulta del usuario. A continuación se ilustra el uso de las tablas ejemplos a través de consultas simples.

tabla ejemplo. El QBE es una tabla esqueleto que muestra el nombre de la tabla y los nombres de las columnas arriba de espacios en blanco que se usan para la entrada de las condiciones de consulta.

Consulta: *¿Quiénes son los fontaneros?*

La solución en QBE aparece en la Figura 8.3. Se ha puesto una "P" debajo de la columna NOMB_TRABAJADOR para indicar que se quiere mostrar en pantalla (o imprimir) el valor de dicha columna [2]. Esto equivale a poner NOMB_TRABAJADOR en la cláusula SELECT de una instrucción SQL. Poniendo el valor literal "Fontanero" en la columna OFICIO se está indicando la condición de que sean seleccionados sólo aquellos trabajadores cuyo OFICIO sea igual a "Fontanero".

[2] La "P" se debe al término *Print* (Imprimir) del inglés (N. del T.).

TRABAJADOR

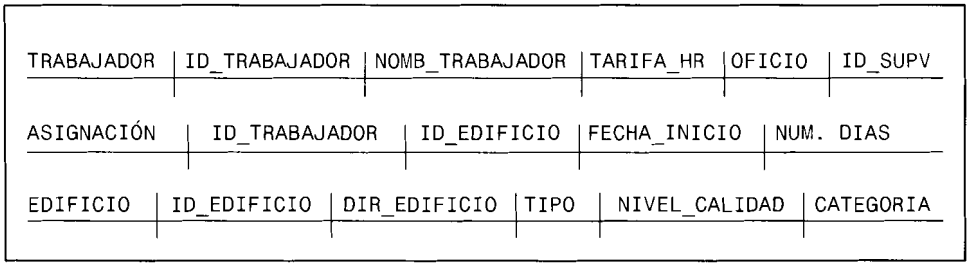
ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
1235	M. Faraday	12,50	Electricista	1311
1412	C. Nemo	13,75	Fontanero	1520
2920	R. Garret	10,00	Albañil	2920
3231	P. Mason	17,40	Carpintero	3231
1520	H. Rickover	11,75	Fontanero	1520
1311	C. coulomb	15,50	Electricita	1311
3001	J. Barrister	8,20	Carpintero	3231

ASIGNACIÓN

ID_TRABAJADOR	ID_EDIFICIO	FECHA_INICIO	NUM. DIAS
1235	312	10/10	5
1412	312	01/10	10
1235	515	17/10	22
2920	460	05/10	18
1412	460	08/12	18
2920	435	28/10	10
2920	210	10/11	15
3231	111	10/10	8
1412	435	05/11	15
1412	515	05/11	8
3231	312	24/10	20
1520	515	09/10	14
1311	435	08/10	12
1412	210	15/11	12
1412	111	01/12	4
3001	111	08/10	14
1311	460	23/10	24
1520	312	30/10	17
3001	210	27/10	14

EDIFICIO

ID_Edificio	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORIA
312	123 Elm	Oficina	2	2
435	456 Maple	Comercio	1	1
515	789 Oak	Residencia	3	1
210	1011 Birch	Oficina	3	1
111	1213 Aspen	Oficina	4	1
460	1415 Beech	Almacén	3	3



TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		P.		Fontanero	

NOMB_TRABAJADOR
C.Nemo
H. Rickover

Observe que en este ejemplo la orden QBE va seguida de un punto. De este modo, "P." es la orden de escribir en pantalla. El valor literal "Fontanero" indica una condición de selección de datos. Las órdenes de QBE siempre terminan en un punto y los valores literales aparecen exactamente como se escriben normalmente. El resultado de esta consulta se muestra en la Figura 8.4. Observe que el resultado es una tabla relacional, como es siempre el caso con los lenguajes relacionales.

Consulta: *Indique todos los datos de los edificios que sean oficinas.*

En la solución QBE (Figura 8.5) aparece una "P." debajo del nombre de la tabla, lo que indica que la fila completa para cada edificio oficina debe mostrarse en pantalla. "Oficina" aparece bajo TIPO como condición para la selección de las filas.

Resultado:

ID_EDIFICIO	DIRECCION_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORIA
312	123 Elm	Oficina	2	2
210	1011 Bich	Oficina	3	1
111	1213 Aspen	Oficina	4	1

Consulta: *¿Cuál es la tarifa semanal de cada electricista?*

EDIFICIO	ID_EDIFICIO	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORIA
P.			Oficina		

tabla de resultado. En QBE una tabla sin encabezamientos por columna. Se usa para definir la salida de una consulta.

En esta solución de QBE (Figura 8.6) se ha introducido una segunda tabla que no tiene encabezamientos por columna. Una tabla tal se llama **tabla de resultado** (*target table*) porque contiene información sobre el resultado final o resultado de salida de la consulta. En esta consulta esto es el equivalente de la cláusula SELECT de SQL. Observe que "P." aparece en la primera columna de la tabla de salida. Esto significa que se mostrará cada columna de la tabla de salida. La segunda y la cuarta columna contienen los elementos ejemplo _NT y _TH. Un **elemento ejemplo** es una variable que representa un valor típico no específico en una columna de una tabla. En este caso se definen los elementos ejemplos

elemento ejemplo. Una variable en QBE que representa un valor no especificado en una columna de una tabla

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		_NT	_TH	Electricista	
P.	_NT	Tarifa semanal = '		40* _TH	

en las columnas 3 y 4 de la tabla ejemplo que se muestra arriba de la tabla de resultado. Observe que las variables QBE comienzan con subrayado (_). Puesto que _NT está en la columna NOMB_TRABAJADOR éste equivale a cualquier valor posible en esta columna para la relación TRABAJADOR. Similarmente _TH toma los valores de la columna correspondiente TARIFA_HR. En esta consulta se muestra como resultado un cálculo (la columna 4 de la tabla de salida) y un valor literal (la columna 3 de la tabla de salida). El resultado será:

Resultado:

NOMB_TRABAJADOR	
C. Coulomb	Tarifa salarial semanal = 620,00
D. Faraday	Tarifa salarial semanal = 500,00

Consulta: *¿Quién tiene una tarifa por hora menor de $10?*

Con esta consulta se ilustrará cómo especificar condiciones de comparación simples en QBE (Figura 8.7). Observe la condición

```
< 10
```

en la tabla ejemplo.

Resultado:

NOMB_TRABAJADOR
J. Barrister

Consulta: *¿Quiénes son los fontaneros que ganan más de $12 por hora?*

La solución (Figura 8.8) a esta nueva consulta muestra cómo se expresa en la tabla ejemplo una condición compuesta. Las condiciones en esta tabla son equivalentes a

```
TARIFA_HR > 12  AND  OFICIO = 'Fontanero'
```

Es decir, cuando dos condiciones aparecen en la misma línea en una tabla ejemplo deben ocurrir *ambas* condiciones para que se seleccione una fila. En este caso se ha escogido mostrar todos los datos de las filas seleccionadas, por eso es que se ha puesto una "P." en la primera columna.

Resultado:

ID_TRABAJADOR	*NOMB_TRABAJADOR*	*TARIFA_HR*	*OFICIO*	*ID-SUPV*
1412	C. Nemo	13,75	Fontanero	1520

Consulta: *Indicar los trabajadores que sean fontaneros o que ganen más de $12 por hora.*

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		P.	< 10		

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
			> 12	Fontanero	

La solución (Figura 8.9) a esta nueva consulta ilustra que las condiciones que se especifican en dos líneas diferentes de una consulta en QBE equivalen a condiciones conectadas por un OR booleano. Observe la ubicación de la orden "P.". Aparece en ambas líneas y en cada una de las columnas que se desea mostrar.

Resultado:

NOMB_TRABAJADOR	*TARIFA_HR*	*OFICIO*
M. Faraday	12,50	Electricista
C. Nemo	13,75	Fontanero
P. Mason	17,40	Carpintero
H. Rickover	11,75	Fontanero
C. Coulomb	15,50	Electricista

Consulta: *¿Quién tiene una tarifa por horas entre $10 y $12?*

La solución se muestra eu la Figura 8.10

ID_TRABAJADOR	*NOMB_TRABAJADOR*	*TARIFA_HR*	*OFICIO*	*ID-SUPV*
2920	R. Garret	10.00	Albañil	2920
1520	H. Rickover	11.75	Fontanero	1520

caja de condición. En QBE es una caja en la cual se puede expresar una condición compleja de una consulta.

Para resolver este tipo de consulta se introduce una **caja de condición** (*condition box*). Esta caja, con la etiqueta CONDITIONS (CONDICIONES), contiene las restricciones que el usuario desea imponer sobre los valores. En este ejemplo, en el que se aplican dos condiciones simultáneamente en una misma columna, es conveniente crear una sola condición, utilizando un AND booleano y ponerla en la caja de condición.

Consultas de múltiples tablas

consnlta multi-tabla. Una consulta que afecta a más de una tabla.

El aspecto más importante de las **consultas de múltiples tablas** (consultas multitabla) es que implica la conexión de los datos *a través* de varias tablas, en otras palabras, haciendo una reunión (*join*) de las tablas. A continuación se verá cómo se hace esto en QBE.

Consulta: *¿Cuáles son los oficios de los trabajadores asignados a la construcción del edificio 435?*

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		P.	P. > 12	P.	
		P.	P.	P. Fontanero	

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
P.			_TH		

CONDICIONES
_Th > = 10 AND_TH < = 12

Lo importante a destacar en la solución QBE a esta consulta (Figura 8.11) es el mismo elemento _IT que aparece en ambas tablas TRABAJADOR y ASIGNACIÓN. Así es como se expresa una reunión en QBE. Esto significa que un *par* seleccionado de filas de las dos tablas debe tener *los mismos valores* para las dos respectivas columnas (en las que aparece el _IT). Se ha escogido _IT como el elemento ejemplo, ya que es una abreviatura adecuada para el nombre de la columna ID_TRABAJADOR. Podría haberse escogido cualquier otro elemento ejemplo, como _X, pero éste no sería de ninguna ayuda en este contexto. El valor literal 435 se pone en la columna ID_EDIFICIO de ASIGNACIÓN para indicar que se deben considerar solamente las filas correspondientes al edificio 435. Por último, la P. en la columna OFICIO quiere decir que se quiere mostrar en pantalla el valor de dicha columna.

Resultado:

OFICIO

Fontanero
Albañil
Electricista

Consulta: *Indicar los trabajadores y los nombres de sus supervisores.*

La solución a esta consulta (Figura 8.12) nos ilustra la facilidad con la que se puede hacer en QBE la reunión (*join*) de una relación con ella misma. Observe que usando el elemento ejemplo _IS en ambas tablas ejemplos estamos exigiendo que el valor de ID_SUPERVISOR en la primera copia de TRABAJADOR sea igual a ID_TRABAJADOR en la segunda copia de TRABAJADOR. Se muestra entonces el valor de la columna NOMBRE_TRABAJADOR para ambas tablas (como se ve en la tabla de salida). La segunda columna representa el nombre del supervisor. Se ha asumido en esta base de datos que los trabajadores que no tienen supervisores son sus propios supervisores.

Resultado:

NOMBRE_TRABAJADOR	*NOMBRE_TRABAJADOR*
M. Faraday	C. Coulomb
C. Nemo	H. Rickover
R. Garret	R. Garret
P. Mason	P. Mason
H. Rickover	H. Rickover
C. Coulomb	C. Coulomb
J. Barrister	P. Mason

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
	_IT			P.	

ASIGNACIÓN	ID_TRABAJADOR	ID_EDIFICIO	FECHA INICIO	NÚM_DÍAS
	_IT	435		

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		_NT			_IS

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
	_IS	_NS			

P.	_WN	_NS			

Consulta: *Indique los nombres de los trabajadores asignados a edificios que sean oficinas.*

La solución (Figura 8.13) a esta consulta ilustra la reunión de tres tablas. Es difícil imaginar que una reunión tal se pueda hacer de modo más fácil que la que se puede hacer aquí con QBE. El poder de la programación gráfica se hace obvio con este ejemplo.

Resultado:

NOMBRE_TRABAJADOR

M. Faraday
C. Nemo
R. Garret
P. Mason
H. Rickover
J. Ballester

Consulta: *Indique los trabajadores que tienen una tarifa por hora mayor que la de su supervisor.*

De nuevo hacemos una reunión de TRABAJADOR consigo mismo (Figura 8.14). Sin embargo, en esta ocasión se necesita una caja de condición para seleccionar a aquellos trabajadores que tienen una tarifa salarial mayor que la de su supervisor. La caja de condición es el único medio disponible para comparar valores en dos columnas diferentes. En SQL este tipo de consulta se resolvía con una subconsulta correlacionada. El resultado queda:

Resultado:

NOMBRE_TRABAJADOR

C. Nemo

Funciones integradas o predefinidas (Built-In Functions)

función integrada o predefinida. Una función estadística que opera sobre un conjunto de filas - SUM, AVG, MÁX, MÍN, CNT.

Al igual que SQL, QBE también tiene **funciones integradas** (*built-in functions*). Estas pueden usarse por ejemplo para encontrar el mayor o el menor valor de una columna, calcular el valor promedio o el total de una columna, y para contar el número total de elementos en

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
	_IT	P.			

ASIGNACION	ID_TRABAJADOR	ID_EDIFICIO	FECHA_INICIO	NUM_DIAS
	_IT	_IE		

ASIGNACION	ID_EDIFICIO	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORIA
	_IE		Oficina		

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		_NT P.	_TT		_IS

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
	_IS		_TS		

CONDICIONES
_TT > _TS

una columna. En QBE estas fuucioues integradas tienen los nombres MAX, MIN, AVG[3], SUM, y CNT[4].Estas pueden aparecer en las tablas de salida o en las cajas de condición.

Consulta: *¿Cuáles son las tarifas por hora más alta y más baja?*

La solución aparece en la Figura 8.15.

Resultado: 18,40, 8,20

Este resultado consiste en dos valores que se pueden imaginar como una tabla de dos columnas y uua sola fila. Las columnas con las cabeceras TARIFA_MAXIMA y TARIFA_MINIMA. De este modo, el resultado de una consulta cualquiera sigue siendo una tabla relacional.

Consulta: *¿Cuál es el número promedio de días que los trabajadores están asignados al edificio 435?*

La solución aparece en la Figura 8.16.

Resultado: 12,33

Consulta: *¿Cuál es el número total de días asignados a los Fontaneros en el edificio 312?*

La solución aparece en la Figura 8.17.

Resultado: 27

Consulta: *¿Cuántos tipos de oficios diferentes hay?*

El operador UNQ[5] en la Figura 8.18 garantiza que se contará sólo una sola ocurrencia por oficio. Es decir, los duplicados no se tomarán en cuenta a la hora de contar.

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
			_TH		
P.	MÁX._TH	MÍN._TH			

ASIGNACIÓN	ID_TRABAJADOR	ID_EDIFICIO	FECHA INICIO	NUM_DIAS
		435		_ND
P.	AVG. _ND			

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
	_IT			Fontanero	
ASIGNACIÓN	ID_TRABAJADOR	ID_EDIFICIO	FECHA INICIO	NUM_DIAS	
	_IT	312		_ND	
P.	SUM. _ND				

[3] Abreviatura de *average* (promedio) (N. del T.).
[4] Abreviatura de *count* (cantidad) (N. del T.).
[5] Abreviatura de *unique* (único) (N. del T.).

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
				_OF	
P.	CNT. UNQ_OF				

Resultado: 4

Consulta: *¿Cuál es la media de la tarifa semanal?*

La solución aparece en la Figura 8.19.

Resultado: 509,14

Consulta: *¿Cuántos edificios tienen nivel de calidad 3?*

La solución aparece en la Figura 8.20.

Resultado: 3

GROUP BY

Al igual que en SQL, se pueden agrupar las filas que tienen una valor en común en una o más columnas. Esto significa que las filas de una relación se dividen en grupos, un grupo para cada valor de la columna designada. A cada grupo se le pueden aplicar las funciones estadísticas.

Consulta: *Para cada supervisor, ¿cuál es la tarifa por hora mayor que se le paga a un trabajador controlado por este supervisor?*

"G." indica la columna por la que se agrupa (Figura 8.21). En este caso se está agrupando por ID_SUPV, puesto que queremos conocer la tarifa máxima de los trabajadores por supervisor. La tabla de resultado (*target table*) se usa para indicar que se quiere mostrar la colnmna por la que se agrupa junto con la función integrada que se aplica a los grupos.

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
			_TH		
P.	AVG. (40 * _TH)				

EDIFICIO	ID_EDIFICIO	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORIA
	_ID			3	
P.	CNT. _IE				

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
			_TH		G. _IS
P.	_SI	MAX. _TH			

Resultado:

ID_SUPV

```
1311     15,50
1520     13,75
2920     10,00
3231     17,40
```

Por supuesto, la agrupación se puede combinar con la especificación de condiciones.

Consulta: *Para cada tipo de edificio ¿cuál es el nivel de calidad promedio de los edificios de categoría 1?*

La solución puede verse en la Figura 8.22.

Resultado:

TIPO

```
Comercio     1
Residencia   3
Oficina      3,5
```

Consulta: *Para cada supervisor que controle a más de un trabajador, ¿cuál es la tarifa por hora más alta que se le paga a un trabajador de los supervisados por él?*

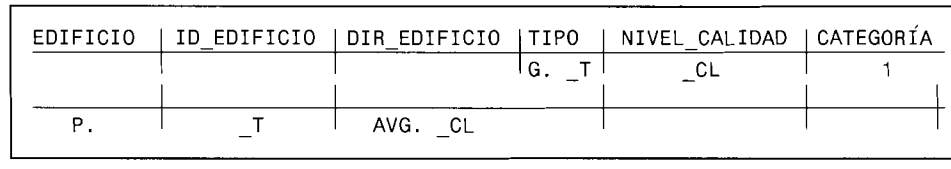
En la solución (Figura 8.23), la condición en la caja de condición determina que cada grupo debe tener más de una fila. De modo que la caja de condición en este ejemplo es equivalente a una cláusula HAVING de SQL.

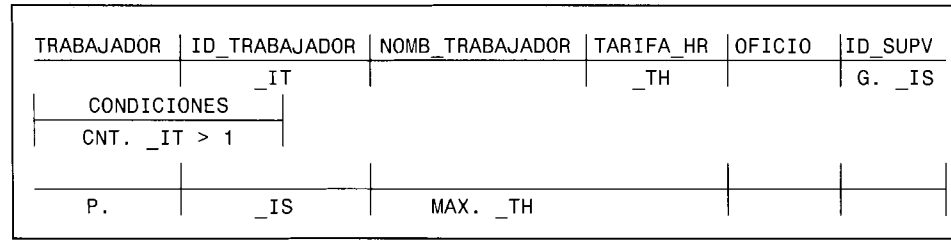
Resultado:

ID_SUPV

```
1311     15,50
1520     13,75
3231     17,40
```

Consulta: *¿Qué trabajadores ganan por encima de la media?*





Se necesitan dos copias de la tabla TRABAJADOR para esta consulta y dos variables *diferentes* para designar a TARIFA_HR (Figura 8.24). Una variable (_TH1) se usa para calcular la tarifa por hora promedio para todos los trabajadores, mientras que la otra variable (_TH2) se usa para designar a la TARIFA_HR para cada trabajador en específico.

Resultado:

NOMB_TRABAJADOR

C. Nemo
P. Mason
C. Coulomb

Consulta: *¿Qué trabajadores tienen una tarifa por hora más alta que la del promedio de los trabajadores que informan al mismo supervisor que él?*

En esta solución (Figura 8.25) simplemente se ha modificado la consulta anterior añadiendo una agrupación por la columna ID_SUPV de la primera tabla y conectando las tablas mediante ID_SUPV.

Resultado:

NOMB_TRABAJADOR

C. Nemo
P. Mason
C. Coulomb

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
			_TH1		
TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		P. _NT	_TH2		

CONDICIONES
_TH2 > AVG. _TH1

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
			_TH1		G. _IS
TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
		P. _NT	_TH2		_IS

CONDICIONES
_TH2 > AVG. _TH1

Operaciones que modifican la base de datos

QBE soporta tres tipos de operaciones que cambian la base de datos: insertar (*insert*), actualizar (*update*) y borrar (*delete*).

Insert. Al igual que en SQL, se pueden insertar filas especificando simplemente el valor individual de cada columna, o se puede insertar un grupo de filas mediante la especificación de una consulta. Como ejemplo del primer método puede considerarse la consulta QBE de la Figura 8.26. Esta consulta es una instrucción de inserción, como lo indica la "I." de la primera columna. Esto provoca que se añada una fila, con los valores que se muestran, a la tabla ASIGNACIÓN. Puesto que la columna NÚM_DÍAS se ha dejado en blanco, dicha columna contendrá un valor nulo al insertar la fila.

ASIGNACION	ID_TRABAJADOR	ID_EDIFICIO	FECHA INICIO	NUM_DIAS
I.	1284	485	13/05	

La siguiente consulta (Figura 8.27) añadirá filas a la relación de nombre EDIFICIO_2, la cual consiste en las columnas ID_EDIFICIO, TIPO y NIVEL_CALIDAD. Las filas que se añaden son aquellas que en EDIFICIO tienen categoría 2.

Obsérvese que se ha puesto una condición (CATEGORÍA = 2) sobre las filas de EDIFICIO, mientras que la orden insertar (I.) está en la tabla EDIFICIO_2. De este modo se lleva a cabo una consulta sobre EDIFICIO en la que las filas seleccionadas se insertan en EDIFICIO_2. Una tabla como EDIFICIO_2 podría usarse, por ejemplo, como tabla temporal, sobre la cual se podría llevar a cabo una gran variedad de consultas.

Actualización (*update*). La consulta QBE de la Figura 8.28 nos da que a todos los que están trabajando para el supervisor 1520 se les aumente su tarifa por hora en un 5 por 100.

Se necesita mostrar la columna TARIFA_HR dos veces: una para definir la variable _T, y otra para indicar la orden de actualización (U.) y la fórmula (_T * 1.05). Incidentalmente, esta facilidad de mostrar múltiples copias de una misma columna en una tabla ejemplo puede usarse también en consultas (por ejemplo, si se aplican múltiples condiciones a una misma columna mediante un AND).

Borrar (*delete*). Si lo que se desea es borrar a todos los que trabajan para el supervisor 1520, entonces se puede usar la consulta que se muestra en la Figura 8.29, donde D. indica la orden borrar[6].

EDIFICIO	ID_EDIFICIO	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORIA
	_IE		_T	_CL	2

EDIFICIO2	ID_EDIFICIO	TIPO	NIVEL_CALIDAD
I.	_IE	_T	_CL

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	TARIFA_HR	OFICIO	ID_SUPV
			_T	U. _T*1.05		1520

TRABAJADOR	ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
D.					1520

▼ Paradox para Windows

PARADOX PARA WINDOWS. Una base de datos para microcomputadoras cuyo lenguaje de consulta es QBE.

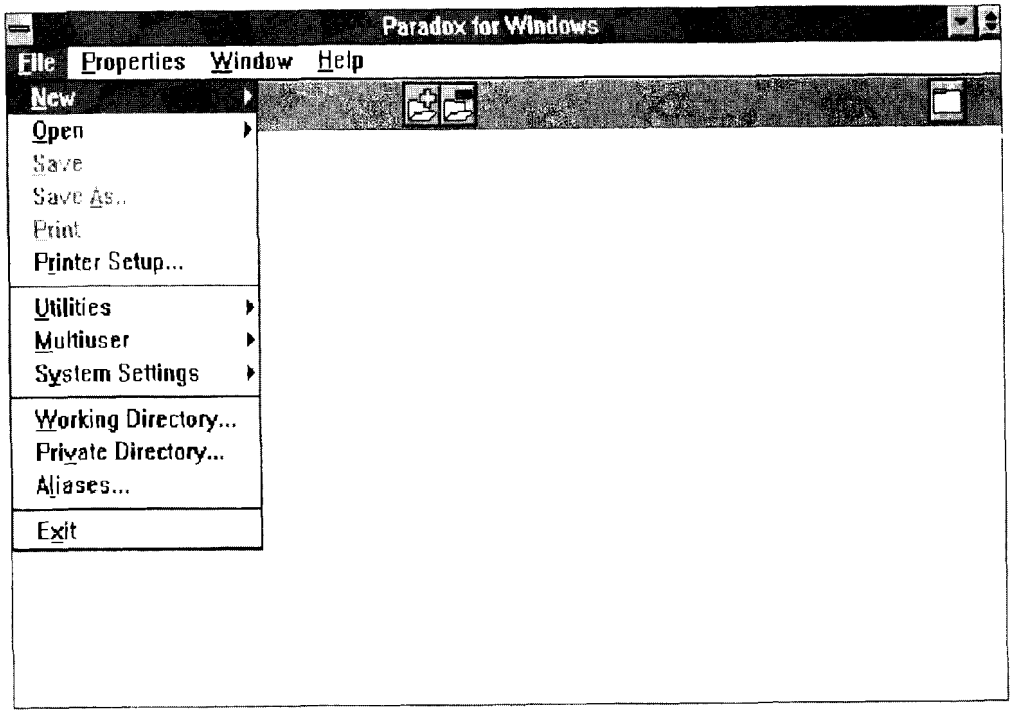
Una base de datos poderosa que incorpora QBE para entornos con microcomputadoras es **PARADOX PARA WINDOWS**. Puesto que muchos usuarios de este libro deberán tener acceso a laboratorios de microcomputadores, en el resto de este capítulo se tratarán algunas de las características y capacidades más importantes de PARADOX PARA WINDOWS.

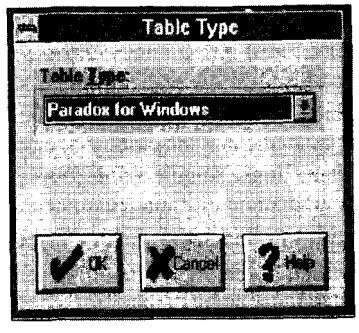
[6] Del inglés *Delete* (borrar) (N. del T.).

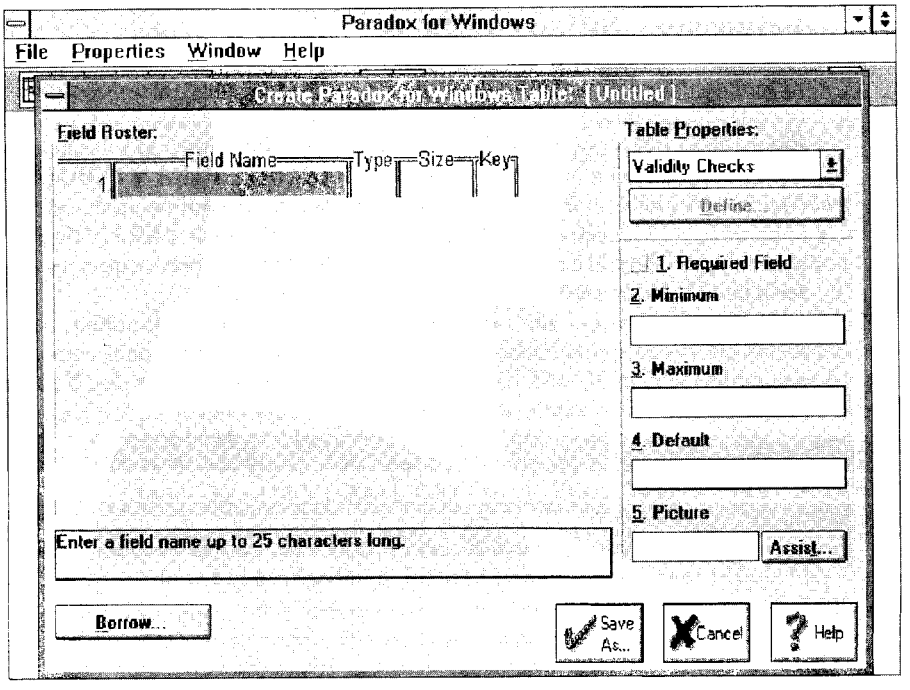
Definición y entrada de datos

El lenguaje de definición de datos de PARADOX es muy fácil de aprender. Pulsando la opción *File* (archivo), en el menú se obtienen a su vez las opciones que se muestran en la Figura 8.30 (Observe que todas las opciones también pueden seleccionarse por teclado. Esto puede verse en cualquier manual de PARADOX PARA WINDOWS). Se selecciona *New* (nuevo) del menú de File y luego *Table* (tabla) del submenú de éste. Esto produce la ventana que se muestra en la Figura 8.31. Se puede elegir PARADOX PARA WINDOWS como el tipo de tabla a crear, pulsando el botón de OK, se verá entonces una pantalla similar a la de la Figura 8.32.

Supongamos que se desea crear una base de datos de manufacturas que se compone de las relaciones que se muestran en la Figura 8.33. Explicaremos cómo hacer la creación de la relación PERSONAL y se dejará el resto como ejercicio. Teclee **Númemp** en la colum-



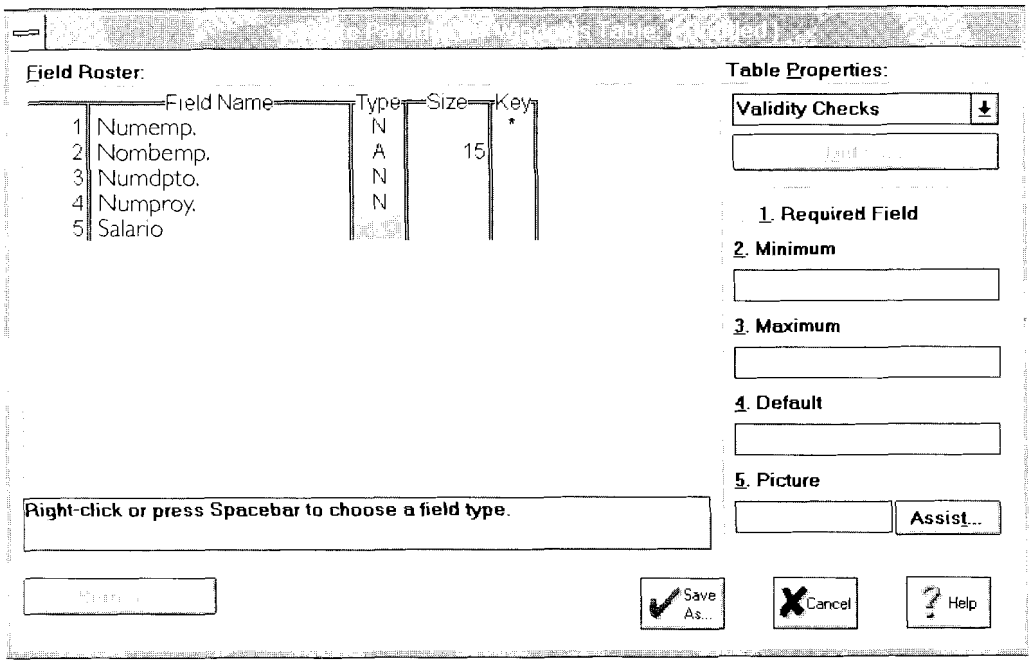


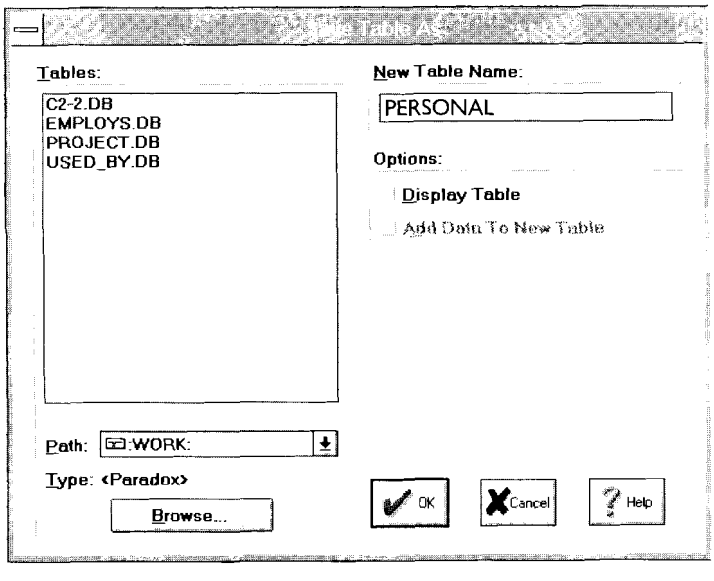


(a) PERSONAL (*NUMEMP*, NOMBEMP, NUMDPTO, NUMPROY, SALARIO)
(b) DPTO (*NUMDPTO*, NOMBDPTO)
(c) PROYECTO (*NUMPROY*, CLIENTE, LOCALIDAD)
(d) PARTEPR (*NUMPROY*, *NUMPARTE*, CANTIDAD)
(e) PARTE (*NUMPARTE*, VENDEDOR, COSTO)

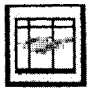
na de nombre Field Name (nombre del campo). Muévase para la columna Type presionando las teclas *Tab*, *Enter* o Flecha Derecha (*right-arrow*). Añada el tipo del campo (**A** para alfanumérico, **N** para numérico, **D** para fecha (*date*), **$** para dinero). Para NúmMemp (número del empleado) el tipo de dato debe ser **N**. Dado que el Númemp puede ser una clave, debemos movernos para la columna **Key** (clave) y presionar cualquier tecla para indicar que Númemp es una clave. Aparecerá un asterisco que indicará que éste es una clave. Presionando de nuevo las teclas *Tab*, *Enter* o Flecha Derecha se avanza al próximo atributo. Entre **Nombemp** en la columna Field Name (Nombre de Campo). En la columna Type (tipo) ponga **A** y en la columna Size (Tamaño) ponga **15**. Luego avance al próximo atributo, **Númdpto**. Las entradas restantes siguen un patrón similar y se muestran en la Figura 8.34. Observe que los campos que sean de tipo fecha (*date*), numérico, o dinero, no requieren que se especifique una longitud para el campo. Observe la disponibilidad de un cierto número de comprobaciones de edición sobre los datos de los campos, tales como limitar el rango de los valores a introducir en el campo, la validación a través de tablas, y otros.

Cuando se haya definido el último atributo se pulsa el botón Save As (salvar como) para guardar y dar un nombre a la tabla. Se verá una pantalla similar a la de la Figura 8.35. Teclee **PERSONAL** como nombre de la tabla en la caja texto *New Table Name* (Nuevo nombre de tabla) y luego pulse el botón OK. Los nombres de tabla están limitados a ocho caracteres (que no pueden ser espacios).





Botón Open Table (*Abrir tabla*).



El resto de las relaciones se crean de manera similar. Una vez que las relaciones (tablas) y sus atributos han sido definidos, se pueden introducir los datos usando los recursos del menú (esto es, usando las opciones del menú *File* | *Open* | *Table* o pulsando el botón **Open Table** (Abrir Tabla) localizado sobre la barra rápida *(speedbar)*. Después se debe pulsar la tabla deseada y luego F9 para comenzar la entrada de datos.

Manipulación de datos

Al lenguaje de manipulación de datos de PARADOX puede accederse escogiendo las siguientes opciones de menú: *File* | *New* | *Query*. Puesto que el lenguaje de consulta es visual a la manera de QBE, el usuario no tiene que memorizar órdenes complejas de consulta. Todo lo que se necesita hacer es pasar a la ventana de consulta. Se puede especificar las tablas a ser consultadas y los ejemplos de los datos requeridos.

Seleccionar atributos y filas

Botón Run Query (Ejecutar consulta).

Para seleccionar los atributos de la consulta primero se selecciona *File* | *New* | *Query*, luego se pulsa el nombre de la tabla y sobre OK. Esto causa que aparezca una tabla en blanco (ver Figura 8.36). A continuación se debe pulsar la caja de comprobación (*check box*), que está al lado del atributo deseado (lo cual pone la marca "√"). Después que se presione el botón **Run Query** (ejecutar consulta), PARADOX produce una tabla respuesta que contiene los valores de los atributos que fueron marcados tal y como están en la tabla.

Para renombrar un atributo durante una consulta, de modo que el nombre del campo que aparezca en la tabla de respuesta sea diferente del nombre del atributo en la tabla de consulta, lo que hay que hacer es simplemente teclear "**as**" [7] seguido del nombre con el que se quiere que aparezca dicho atributo en la tabla de respuesta. En la Figura 8.37 se ejemplifica esto. En este caso se están viendo los nombres de los empleados del Departamento 100 y se ha cambiado el nombre del atributo "Nombemp" por el de "Nombre" sólo a los propósitos de la salida. Como se puede observar, los atributos han sido escogidos usando el "√", mientras que las filas se seleccionan basándose en algún criterio que hay que suministrar (en este caso "Númdpto = 100").

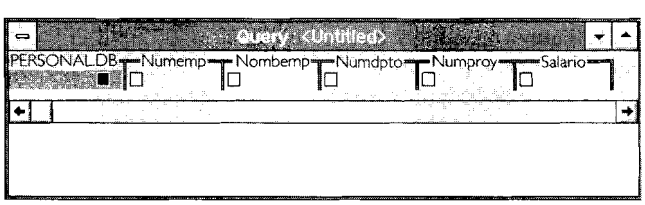
consulta disyuntiva. Una consulta cuyas condiciones están conectadas por un "or" , o algún símbolo equivalente.

Consultas conjuntivas y disyuntivas

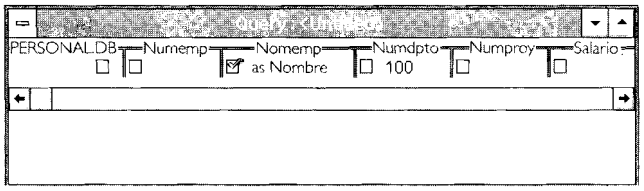
La Figura 8.38 extiende la consulta de la Figura 8.37 para pedir los nombres de todos los empleados del Departamento 100 que ganan más de $30.000 al año. Esta es una consulta **conjuntiva** ("and") en la que estamos preguntando por la lista de empleados que están en el departamento 100 *y* (*and*) que ganan al año más de $30.000. La conjunción se puede extender a más de dos condiciones simplemente entrando todas las condiciones "and" en una línea de la consulta. Las conjunciones pueden comprender varios atributos, así como varios valores para un simple atributo.

Supongamos que se quiere una conjunción con varios valores de un mismo atributo, tal como que el salario sea mayor de $30.000 y menor que $50.000. Esto es lo que se hace en la Figura 8.39. La *coma* entre los valores deseados es la que establece la conjunción requerida.

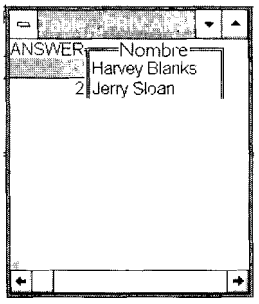
La condición **disyuntiva** ("or") se puede indicar de dos maneras. Dos o más filas en una consulta indican una conexión "or" entre cada fila, tal como se muestra en la Figura 8.39. Aquí se ha pedido una lista con los nombres de todos los empleados que están en el Departamento 100, *o* que ganan más de $30.000 al año. En la Figura 8.41 se ilustra una



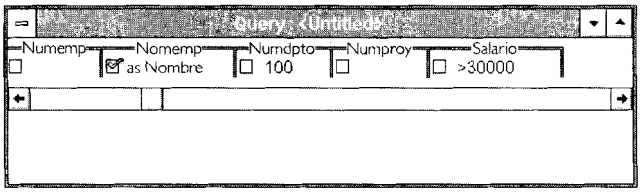
[7] *Como, poner como* (N. del T.).

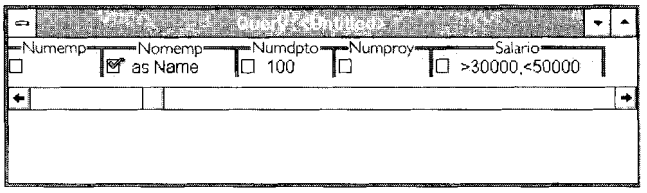


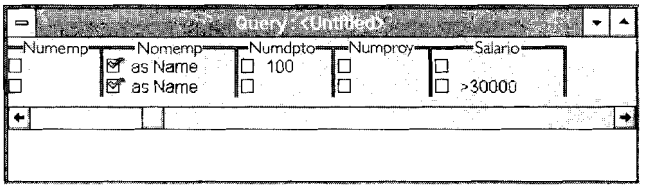
(a) La consulta



(b) La respuesta



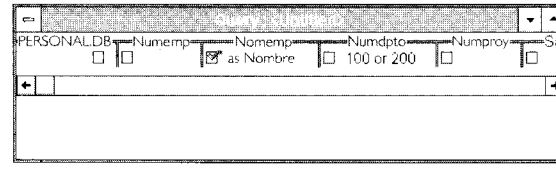


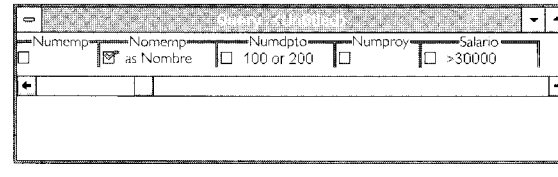


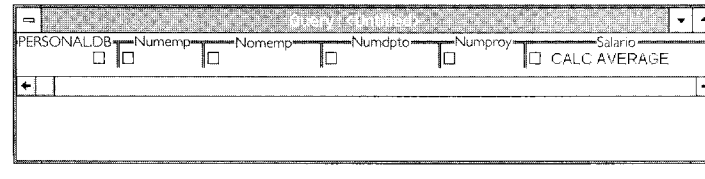
segunda forma de hacer esto, donde el "or" se ha insertado entre las condiciones. Este método puede utilizarse sólo si la condición "or" se aplica al contenido de un único atributo. En la Figura 8.41 se han solicitado los nombres de los empleados asignados al Departamento 100 o (*or*) al Departamento 200.

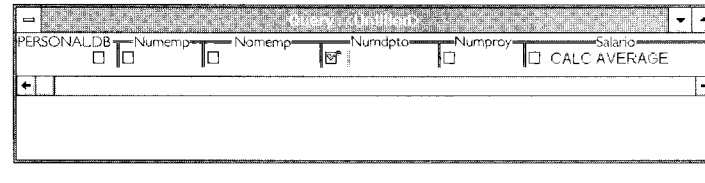
Al igual que en cualquier lenguaje de consulta, las condiciones "and" y "or" se pueden combinar en una misma consulta. En la Figura 8.42 se muestra un ejemplo. Aquí se han solicitado los nombres de los empleados asignados a los departamentos 100 *o* (or) 200 *y* (and) que ganan más de $30.000 al año.

Cálculos. Se pueden llevar a cabo cálculos dentro de una consulta. Todos los cálculos deben estar precedidos del operador CALC. Se puede ver un ejemplo en la Figura 8.43. Se ha solicitado el cálculo del salario promedio. En la Figura 8.44 se observa un ejemplo similar, sólo que aquí lo que se ha solicitado es el salario promedio por departamento. Poniendo una marca en la columna Departamento se lleva a cabo el cálculo indicado para los grupos de filas que tienen el mismo valor en el atributo Departamento. En estos dos ejemplos se ha usado el operador CALC junto con el operador AVERAGE (promedio). Otras operaciones son SUM, COUNT, MAX, y MIN. También se pueden llevar a cabo cálculos especializados.







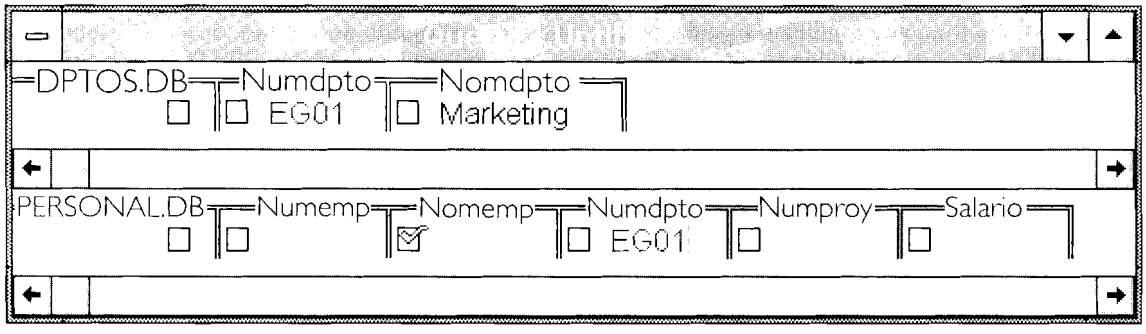


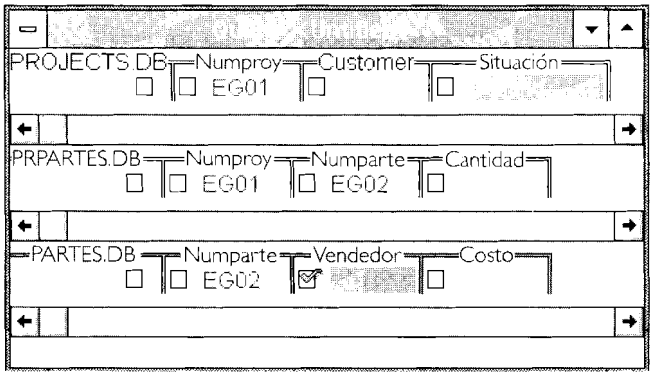
Tablas enlazadas *(linking tables)*. Las consultas que afectan a una sola tabla son sencillas puesto que sólo requieren que se identifiquen los atributos que se quieren en la salida y las condiciones que se quiere sean satisfechas. Consultas más complejas implican por lo general enlazar los datos de una tabla con los datos de otra (o de varias). Esto se lleva a cabo de forma elemental en PARADOX.

En la Figura 8.45 se muestra un ejemplo. Lo que se desea aquí es producir una lista de nombres de empleados para aquellos individuos asignados al departamento de marketing. El "√" denota los atributos cuyos valores se desea imprimir. Oprimiendo el botón **Join Tables** (reunión de tablas) de la barra rápida y luego pulsando en los atributos de las dos tablas que se utilizarán como enlace, PARADOX pone una única variable "EG" que muestra que el enlace es válido (EG viene del latín, *exempli gratia*, que significa "por ejemplo"). La variable "EG01" que se introduce en la columna NUMDPTO de ambas tablas DPTOS y PERSONAL actúa como variable mediante la cual se enlazan las tablas (lo mismo podría haberse usado "x" o "y"). Sin embargo, obsérvese que la "EG01" en la tabla DPTOS está restringida al valor de NUMDPTO asociado con el nombre "Marketing". De este modo, si suponemos que el número del departamento de marketing es 300, entonces el único "EG01" permisible en DPTOS es 300. Cuando se encuentra un empleado asociado con el NUMDPTO 300 en la tabla PERSONAL, se recupera el nombre como parte de la solución a la consulta.

Botón Join Tables.

En la Figura 8.46 se muestra un ejemplo algo más complicado. Aquí se ha solicitado una lista de todos los vendedores que suministran piezas a los proyectos localizados en "Outland". Vemos que las tablas PRPARTES y PARTES están enlazadas por la variable de nombre "EG02". Siempre que se encuentra una correspondencia en NUMPARTE se enla-





zan las dos filas respectivas. Mirando más allá en la tabla PRPARTES se ve que esas piezas se enlazan con los proyectos en "Outland" a través de la variable "EC01". Esta variable conecta las filas en la tabla PRPARTES con las filas apropiadas en la tabla PROYECTS[8].

Lenguaje de aplicación

El lenguaje de aplicación de PARADOX (*OBJECTPAL*) es un lenguaje de programación de base de datos de alto nivel, orientado a objetos y estructurado. Éste proporciona al usuario un método para integrar las órdenes interactivas de PARADOX dentro de una aplicación. La escritura de programas usualmente se ve como una serie de instrucciones, compuestas de aseveraciones simples, de instrucciones condicionales y de instrucciones iterativas. A continuación se verá brevemente la sintaxis general de estas dos últimas estructuras.

instrucción condicional. Una instrucción que pregunta por una condición y determina el procesamiento a seguir según el cumplimiento de la condieión.

Instrucciones condicionales. La sintaxis general de las instrucciones IF-THEN-ELSE es como sigue

```
IF <condición>
    THEN <instrucciones>
    ELSE <instrucciones>
ENDIF
```

instrucción SWITCH Una instrucción que permite preguntar por una serie de condiciones.

instrucción CASE. Se usa con el SWITCII para indicar cuál es el procesamiento a seguir cuando una condición sea verdadera.

Las instrucciones **SWITCH** son similares a múltiples instrucciones IF-THEN-ELSE. Mientras que cada IF-THEN-ELSE contiene sólo dos ramas, una instrucción SWITCH contiene una instrucción CASE para cada rama posible. La instrucción **CASE** contiene una condición seguida de dos puntos y un conjunto de órdenes. Se ejecuta sólo una de las instrucciones CASE cada vez que se ejecute una instrucción SWITCH. A continuación se muestra un ejemplo:

```
SWITCH
    CASE  opcion  =  "Datos del cliente" :  <lista de órdenes - A>
    CASE  opcion  =  "Transacciones" :  <lista de órdenes - B>
          .
          .
          .
    OTHERWISE :  <lista de órdenes - K>
ENDSWITCH
```

La primera instrucción CASE que satisfaga una condición (que evalúe verdadero) será ejecutada. Si ninguna de las condiciones CASE es verdadera entonces se ejecuta la instrucción OTHERWISE.

instrucción iterativa. Una instrucción que se puede repetir una cierta cantidad de veces.

Instrucciones iterativas. La sintaxis general de una instrucción de iteración es la siguiente:

```
WHILE <condición>
    < lista de instrucciones a ejecutar >
ENDWHILE
```

Se comprueba la condición del WHILE (mientras) y si ésta es verdadera se ejecuta la lista de instrucciones. Cada vez que la condición se cumpla, se ejecuta la lista. La instrucción ENDWHILE hace que la ejecución vuelva de nuevo a la parte WHILE. Cuando la condición del WHILE evalúe falso, el control del programa se pasa a la instrucción que siga al ENDWHILE.

---

[8] Recuerde que los nombres no pueden tener más de ocho caracteres (N. del T.).

1. Relacione cada término con su definición

—*elemento ejemplo*	**a.** En QBE, una tabla sin encabezamiento en las columnas que es usada para definir la salida de la consulta.
—*función integrada (built-in)*	**b.** Una consulta que afecta a más de una tabla.
—*tabla ejemplo*	**c.** En QBE, una tabla esqueleto que muestra el nombre de la tabla y el nombre de las columnas encima de espacios en blanco que se usan para introducir las condiciones de la consulta.
—*lenguaje gráfico*	**d.** En QBE, una variable que representa un valor no especificado de una columna de la tabla.
—*tabla resultado*	**e.** Una consulta que afecta sólo una tabla de la base de datos.
—*consulta simple*	**f.** Un lenguaje de computadores que usa representaciones pictóricas para resolver los problemas.
—*caja de condición*	**g.** Un lenguaje de computadores cuyas instrucciones consisten en cadenas de caracteres.
—*consulta multi-tabla*	**h.** En QBE, una caja en la cual se pueden expresar condiciones complejas de una consulta.
—*lenguaje textual*	**i.** Funciones estadísticas que actúan sobre un conjunto de filas: SUM, AVG, MAX, MIN, CNT.

Formule soluciones en QBE o en PARADOX a las consultas siguientes. Use el esquema relacional siguiente:

```
CLIENTE   (ID_CLIENTE,  NOMB_CLIENTE,  RENTA_ANUAL)
EMBARQUE  (EMBARQUE_#,  ID_CLIENTE,  PESO,  CAMION_#,  DESTINO)
Clave externa:  DESTINO que referencia a CIUDAD
CAMION    (CAMIÓN_#,  NOMB_CONDUCTOR)
CIUDAD    (NOMB_CIUDAD,  POBLACION)
```

2. *Consultas simples:*
   - **a.** ¿Cuál es el nombre del cliente 433?
   - **b.** ¿Cuál es la ciudad destino del embarque #3244?
   - **c.** ¿Cuáles son los números de los camiones que han llevado paquetes (embarques) por encima de las 100 libras?
   - **d.** Dé todos los datos de los embarques de más de 20 libras.
   - **e.** Cree una lista por orden alfabético de los clientes con renta anual de más de $10 millones.
   - **f.** ¿Cuál es el ID del cliente Wilson Brothers?
   - **g.** Indique los nombres y la renta promedio mensual de los clientes que tienen renta anual que excede los $5 millones, pero que es menor de los $10 millones.

3. *Consultas multi-tablas:*
   - **a.** ¿Cuáles son los nombres de los clientes que han enviado paquetes (embarques) a Sioux City?
   - **b.** ¿A qué destinos han enviado embarques las compañías con renta anual menor de $1 millón?
   - **c.** ¿Cuál es el nombre y la población de las ciudades que han recibido embarques que pesen más de 100 libras?
   - **d.** ¿Quiénes son los clientes que tienen más de $5 millones de renta anual y que han enviado embarques con peso de menos de 1 libra?
   - **e.** ¿Quiénes son los clientes que tienen sobre los $5 millones en renta anual y que han

enviado embarques de menos de 1 libra o que han enviado embarques a San Francisco?

f. ¿Quiénes son los conductores que han transportado embarques de los clientes con renta anual sobre los $20 millones a ciudades con población de más de 1 millón de habitantes?

4. *Funciones integradas (built-in)*
   a. ¿Cuál el peso medio de un embarque?
   b. ¿Cuál es el peso medio de los embarques que van para Atlanta?
   c. ¿Cuántos embarques ha enviado el cliente 433?
   d. ¿Cuáles son las ciudades de la base de datos que tienen la mayor y la menor población?
   e. ¿Cuál es el peso total de los embarques transportados en el camión 82?

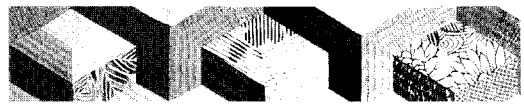
5. *GROUP BY*
   a. Para cada cliente, ¿cuál es el peso promedio de los paquetes (embarques) enviados por dicho cliente?
   b. Para cada ciudad, ¿cnál es el peso máximo de un embarque destinado a dicha ciudad?
   c. Para cada ciudad con población sobre el millón de habitantes, ¿cuál es el peso mínimo de un embarque enviado a dicha ciudad?

6. *Operaciones de modificación de la base de datos*
   a. Añada el camión 95 con el conductor Winston.
   b. Borre de la base de datos todas las ciudades con población por debajo de 5.000. No se olvide de actualizar también la relación EMBARQUE.
   c. Convierta el peso de cada envío a kilogramos dividiendo el peso por 2.2.

1. Usando la definición original de Zloof de QBE (Zloof, 1975, 1977), lleve a cabo una comparación comprensiva de SQL y QBE. Examine cómo los siguientes tipos de consultas SQL se pueden tratar en QBE:
   a. Subconsultas no correlacionadas
   b. Subconsultas correlacionadas
   c. NOT EXISTS

   Para cada uno de los tipos anteriores use consultas específicas del Capítulo 7 y defina cómo se resolverían en QBE.

2. Analice y compare QBE y PARADOX PARA WINDOWS.



CAPÍTULO

9

SISTEMAS DE BASES DE DATOS CLIENTE/SERVIDOR

Introducción: Un resumen de los conceptos de cliente/servidor
Definición de las tablas de la base de datos en el sistema de base de datos servidor
Sistemas de gestión
  Creación de tipos de datos definidos por el usuario
  Definición de tablas individuales
    Definición de claves primarias y claves externas
    Restricciones CHECK
  Lenguaje de definición de datos: Sumario y ejemplo final
Manipulación y programación del servidor de datos
  Lenguaje de flujo-de-control
    Instrucción de bloque:
    BEGIN ... END
    Instrucción condicional IF...ELSE
    Ejecución iterativa: Instrucción WHILE
    BREAK y CONTINUE
    DECLARE y variables locales
  Procedimientos almacenados
    Valores por defecto
    Uso del RETURN
  Introducción a los disparadores (Triggers)
    Uso de los disparadores en SQL Server y Oracle
Desarrollo de aplicaciones cliente
  El enfoque de Power Builder
  Uso de Power Builder
  Construir una aplicación
  Crear DataWindows
  Crear Windows
Resumen
Preguntas de repaso
Problemas y ejercicios
Proyectos y cuestiones profesionales

"Pero, ¿qué es exactamente cliente/servidor?"

Susan Broadbent y Sanford Mallon están discutiendo las posibilidades de cambiar sus aplicaciones de oficina desde un gran computador tradicional a una plataforma cliente/servidor. Susan está reticente en continuar hasta que vea una clara ventaja en hacer el cambio.

Sanford le contesta: "En lugar de tener una única y potente computadora que maneje todas nuestras aplicaciones principales, una plataforma cliente/servidor es una red que enlaza las computadoras de sobremesa con potentes computadoras llamadas servidores. Nuestro sistema de gestión de base de datos funcionará en el servidor y las otras computadoras accederán a la base de datos a través de la red mediante el servidor."

"¿Y qué ventajas nos ofrece este enfoque?"

"Toda nuestra gente ya tiene computadoras en sus escritorios, que usan para sus aplicaciones independientes, tal como procesamiento de texto y hojas de cálculo. Conectándolos tenemos todas las ventajas de una red, tales como compartir software y correo electrónico, pero además podemos tener distintas ventajas cliente/servidor".

¿Como cuáles?

"En el servidor funcionará un SGBD potente y dedicado, mientras que las máquinas clientes tendrán cada una su propia interfaz gráfica de usuario (GUI)[1] para darle más poder y facilidad de uso a la presentación de los datos, los informes, etc. Además, nuestro equipo de sistemas tendría herramientas potentes para desarrollar sistemas en las máquinas cliente y a la vez poder usar todas las herramientas disponibles en el SGBD. Tendremos una mayor cantidad de funciones en nuestras computadoras que las que tenemos ahora, lo que compensaría nuestra inversión de capital."

"Ya esto me parece interesante. Vamos a verlo en más profundidad."

En capítulos anteriores se han estudiado algunos aspectos iniciales de los sistemas cliente/servidor. En este capítulo se ampliará el estudio, se ilustrarán las funciones que están disponibles en los sistemas cliente/servidor tanto en el lado del SGBD o servidor como en el desarrollo de la aplicación o lado del cliente. Después de leer este capítulo será capaz de:

- Comprender la forma de división del trabajo que es posible hacer con el enfoque cliente/servidor y ver cómo esa división beneficia a las organizaciones porque le permite una utilización completa de la capacidad de sus sistemas.
- Definir los esquemas de la base de datos en el SGBD servidor, utilizando muchas capacidades potentes.
- Utilizar el lenguaje de manipulación del SGBD servidor para escribir programas que saquen ventaja del poder único de los conceptos de las bases de datos relacionales.
- Comprender los entornos modernos de aplicación en el cual se producen sistemas clientes que pueden interactuar con SGBDs servidores.

▼ Introducción: Un resumen de los conceptos cliente/servidor

plataforma cliente/servidor. Una red local consistente de computadoras clientes que reciben servicio de una computadora servidor.

En el Capítulo 1 se analizaron algunas características y ventajas de los sistemas cliente/servidor. Se señaló que una **plataforma cliente/servidor** es una red de computadoras

[1] Siglas del término en inglés *Graphical User Interface;* por ser siglas establecidas, se han dejado en inglés (N. del T.).

en la que una o más ofrecen servicios y de ahí el nombre de *servidores*. Los restantes computadores de la red son *clientes* de estos servidores. Cada tipo de máquina se optimiza para que lleve a cabo sus funciones específicas dentro del sistema de la red. De esta manera se optimizan los servicios para brindar servicios a muchos clientes. Los servicios de bases de datos suelen ser de los servicios más frecuentes. Por otro lado, los sistemas clientes se optimizan para la entrada y presentación de los datos. Por tanto, éstos incluyen normalmente **interfaces gráficas de usuario (GUIs)** que brindan medios orientados al usuario para facilitar la entrada de los datos y para dar formato y mostrar los datos de salida.

Interfaz Gráfica de Usuario (GUI). Pantallas y funciones que brindan medios gráficos para que un usuario final acceda a un sistema de cálculo.

En este capítulo se verán los sistemas cliente/servidor desde el lado del servidor o base de datos y desde el lado de la presentación o del cliente. En la primera parte del capítulo se explicarán las características de dos sistemas servidores de bases de datos, Oracle y SQL Server. En particular nos interesaremos en aquellas características de los SGBD que no hayan sido vistas previamente en este libro, o que se hayan visto sólo desde un planteamiento teórico. En la última parte del capítulo se examinará un entorno para el desarrollo de aplicaciones, PowerBuilder, que se usa para desarrollar sistemas que funcionarán en las máquinas clientes de una plataforma cliente/servidor. Estos sistemas son potentes y populares en los entornos comerciales actuales. Todos los nuevos conceptos que se presentarán son vitales para una buena comprensión de la teoría y la práctica de las bases de datos modernas.

Antes de continuar debemos señalar que puesto que cualquiera de estos sistemas tiene muchos más recursos que los que podemos describir en sólo un capítulo, o incluso en un único libro, se han seleccionado solamente aquellos aspectos más relevantes para el contexto de esta exposición. Para más detalles se deben consultar a los proveedores de estos sistemas o de otros similares. Debemos notar que la presentación será autocontenida y que por lo tanto no debe ser necesario que el lector tenga que tener acceso a estos sistemas para que se pueda beneficiar de esta presentación. Los principios implicados aquí se pueden comprender por sí solos y se pueden aplicar en las preguntas, problemas y ejercicios que se ofrecen al final del capítulo.

Como se mencionó en el Capítulo 7, el lenguaje SQL estuvo disponible comercialmente por primera vez a finales de los años 70 en un sistema de gestión de base de datos (SGBD) desarrollado por la empresa Oracle. Como ya dijimos anteriormente, Oracle será estudiado en este capítulo. Por supuesto, Oracle ha experimentado muchos desarrollos y mejoras desde su versión original. La versión del SGBD Oracle que se analizará aquí se llama Oracle7 Server, y está concebida para operar en un entorno cliente/servidor. También veremos otro SGBD comercial llamado SQL Server. SQL Server fue desarrollado inicialmente por Sybase, pero también se comercializa por Microsoft. Analizaremos aquí aspectos del Microsoft SQL Server, Versión 4.2. Ambos productos brindan lenguajes de órdenes orientados a texto. También brindan interfaces de usuario que visualizan y responden a las entradas de los usuarios. Los ejemplos de pantallas en las interfaces de usuario que se usarán provienen de SQL Server. También se verán ejemplos en el lenguaje de órdenes de Oracle.

Ambos productos son SGBD poderosos, que brindan tanto lenguajes de definición de datos (LDD) y lenguajes de manipulación de datos (LMD) considerablemente sofisticados. Este capítulo discute características importantes de ambos lenguajes. Se verá cómo se pueden usar sus LDDs para definir tablas de la base de datos y cómo usar sus LMDs para realizar operaciones avanzadas sobre los datos.

Esta porción del capítulo está dividida en dos partes principales: (1) definición de tablas de la base de datos y (2) manipulación de los datos y programación. Aunque se irán examinando ambos sistemas al mismo tiempo, no es nuestro propósito hacer una comparación detallada de sus capacidades. En lugar de esto, estamos interesados en los principios que puedan derivarse para ser usados en la gestión de bases de datos. Sólo en algunas ocasiones se verá cuándo uno de los sistemas tiene alguna capacidad significativa que el otro no tiene.

▼ Definición de las tablas de la base de datos en el sistema de base de datos servidor

En los sistemas que se examinarán en esta sección, el proceso de definir las tablas de la base de datos consiste en los pasos siguientes:

1. Crear los tipos definidos por el usuario
2. Definir las tablas individuales
   a. Para cada columna definir su nombre, su tipo de datos y posibles restricciones y valores por defecto.
   b. Definir las claves primarias y las claves externas.
   c. Definir las restricciones CHECK.
3. Crear los disparadores (*triggers*)

En esta sección se describirán los pasos 1 y 2. El paso 3, que describe los disparadores, se dejará para más tarde, porque los disparadores implican el uso del lenguaje de manipulación de datos y éstos se describen en la próxima sección.

Creación de tipos de datos definidos por el usuario

La base de datos de la Compañía Constructora Premier consiste en tres tablas: trabajador, edificio y asignación (Figura 9.1). Para relacionar estas tablas entre sí, se ha definido que algunas de sus columnas sean las mismas en varias tablas. Específicamente, id_trabajador aparece en las tablas trabajador y asignación. Además, los tipos de datos de estas dos columnas son similares, son identificadores numéricos en ambos casos. Supongamos que, además de definir estas columnas como numéricas, queremos especificar que deben ser enteros positivos menores que 100.000. Y que, por supuesto, también queremos que ninguno de los dos pueda ser nulo, ya que siempre son utilizados como clave (o clave externa). Esto sugiere la definición de un tipo de datos a la medida, un **tipo de datos definido por el usuario** (*user-defined datatype*), que tenga todas estas características y que sirva para ambas columnas. En esta sección se verá cómo se hace esto.

tipo de datos definido por el usuario. Un subtipo de uno de los tipos de datos suministrados por el sistema, el cual es adaptado a las necesidades del esquema de la base de datos.

```
trabajador (id_trabajador, nomb_trabajador, tarifa_hr, oficio, id_supv)
asignacion (id_trabajador, id_edificio, fecha_inicio, Num_dias)
edificio ( id_edificio, dir_edificio, tipo, nivel_calidad, categoría)
```

SQL Server y Oracle brindan una variedad de tipos de datos definidos por el sistema, tales como carácter (*character*), entero (*integer*), y fecha-hora (*datetime*). Además, SQL Server brinda facilidades para crear tipos de datos definidos por el usuario. Un tipo de datos definido por el usuario no es más que un subtipo de uno de los tipos de datos del sistema. Es decir, es un tipo de datos del sistema que ha sido restringido de alguna mane-

ra. Esto se ilustrará definiendo un tipo de datos *identificador*, que se llamará "tipo_id", y que será un entero positivo, no nulo y menor que 100.000.

Todas las actividades de definición de datos en SQL Server tienen lugar usando SQL Object Manager. Su pantalla principal se muestra en la Figura 9.2. Como se puede apreciar, ésta tiene una barra de menú en el tope y debajo de ésta y hacia la derecha un conjunto de cinco botones. Estos botones permiten conectarse a otros servidores, manejar objetos, transferir datos, escribir guiones (*script*) y consultar la base de datos. En la misma línea, a la izquierda, vemos que está actualmente conectada a un servidor de archivos llamado "SQL SERVER", y nuestra base de datos de nombre "construcción". Podríamos cambiar la base de datos actual a alguna otra disponible si se quisiera. Observe también que el nombre del usuario de la base de datos "John Doe" se muestra como una identificación de entrada (*login*) en el fondo de la pantalla. No se explicará toda la variedad de opciones dada por todos los menús y botones. Sólo se necesita saber que las pantallas que se verán en este capítulo están en el contexto del SQL Object Manager.

Se crea un tipo de datos definido por el usuario accediendo al cuadro de diálogo *Manage Datatypes* (*Manejar Tipos de Datos*) que se muestra en la Figura 9.3. Como se muestra en el medio del diagrama, se pueden especificar las siguientes características para el tipo de datos:

- Nombre (*name*)
- Propietario (*owner*)
- Tipo (*type*)
- Longitud (*length*)
- Por defecto (*default*)
- Regla (*rule*)

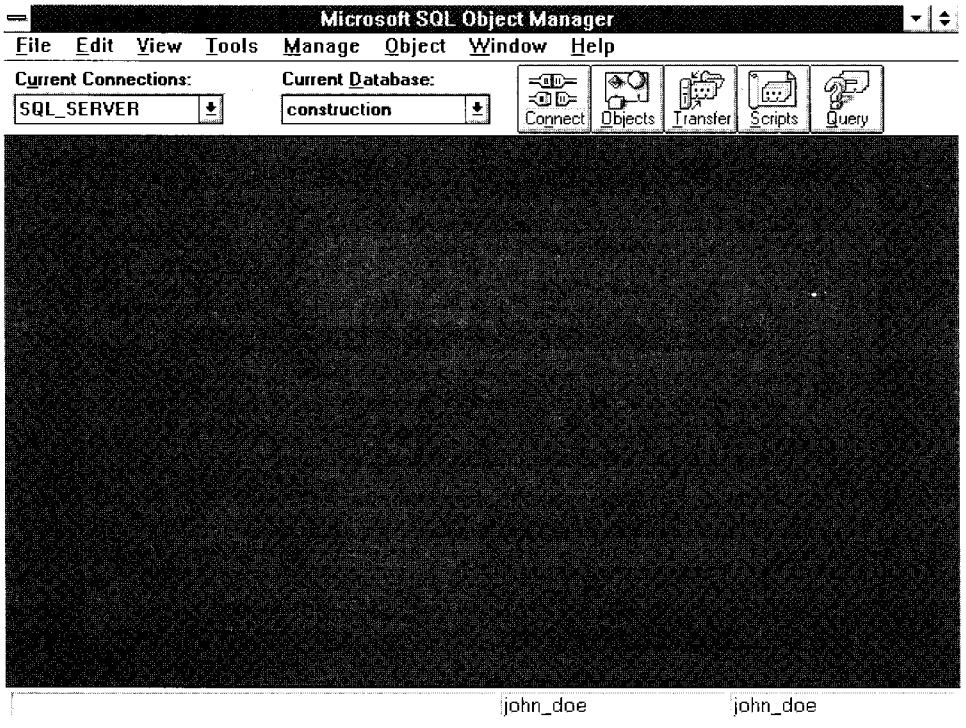
Éstas se irán explicando a medida que avancemos. Se introduce "tipo_id" como *Nombre*. Este será el nombre que se utilizará donde quiera que se definan columnas que tengan este tipo de datos. La caja *Owner* (propietario) automáticamente contendrá el ID login del usuario ("john_doe") y la caja *Type* (tipo) contiene por defecto "*int*" (por *integer*). Puesto que queremos que el tipo base de este tipo definido por el usuario sea *integer* (entero), entonces no se cambiará este campo, aunque podríamos haberlo cambiado por algún otro tipo soportado por el sistema. Ahora, dentro de SQL Server, los campos *integer* (entero) son siempre de la misma longitud, por tanto la longitud del campo no es significativa y podemos pasar sobre esto. Observe ahora que la caja *Nulls* (nulo) está vacía, lo que indica que no se permite el nulo en campos con este tipo de datos. Si quisiéramos permitir nulos, tendríamos que indicarlo en esta caja, pero como no queremos la dejamos así. Las dos últimas entradas son para el valor por defecto y para una regla. Éstas se explicarán con más detalle posteriormente. Por ahora simplemente indicaremos que no queremos ningún valor por defecto para este tipo de datos, por tanto pasamos a la caja *Rule* (regla).

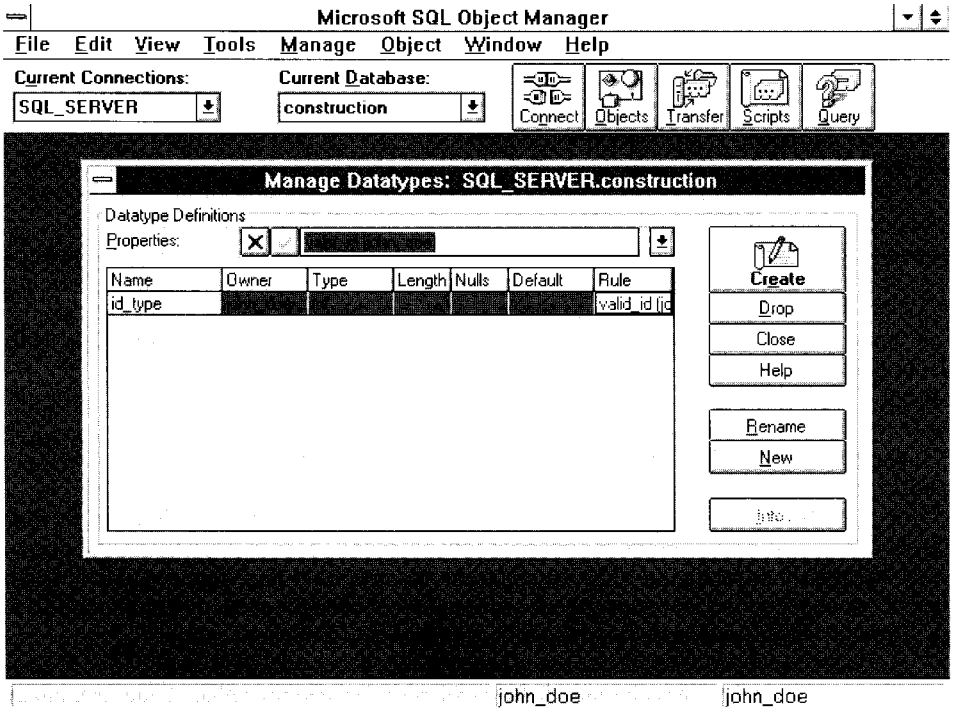
regla. Una restricción sobre el valor que se permite en una columna, establecida por una expresión condicional.

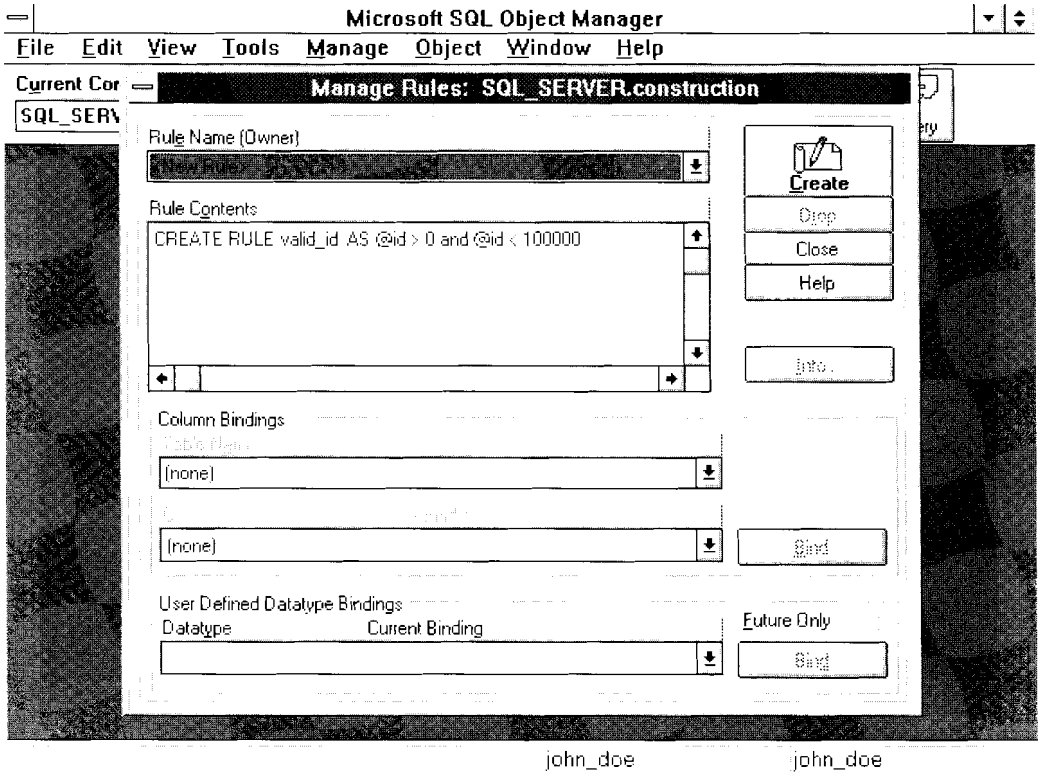
Queremos una **regla** para este tipo de datos para restringir que los valores posibles tengan que ser menores que 100.000. Esta regla se define separadamente y le damos el nombre "id_válido" . Esto se hace usando el cuadro de diálogo *Manage Rules* que se muestra en la Figura 9.4. En la caja *Rule Contents* (contenido de la regla) tecleamos:

```
CREATE RULE id_valido AS @id > 0 and @id < 100000
```

El nombre de la regla es "id_válido" y aparece en la caja *Rule* y en la caja *Properties* (propiedades) en la Figura 9.3. La regla establece que el valor del parámetro @id debe ser mayor que 0 y menor que 100.000. Este parámetro se toma del valor actual de la columna siempre que el tipo de datos se esté usando en una definición de columna. Por ejemplo, supongamos que se define id_trabajador en la tabla trabajador con el tipo de datos


Microsoft SQL Object Manager
File Edit View Tools Manage Object Window Help
Current Connections:
SQL_SERVER
Current Database:
construction
Connect
Objects
Transfer
Scripts
Query
john_doe
john_doe


Microsoft SQL Object Manager
File Edit View Tools Manage Object Window Help
Current Connections:
SQL_SERVER
Current Database:
construction
Connect
Objects
Transfer
Scripts
Query
Manage Datatypes: SQL_SERVER.construction
Datatype Definitions
Properties:
Name
Owner
Type
Length
Nulls
Default
Rule
id_type
valid_id
Create
Drop
Close
Help
Rename
New
john_doe
john_doe



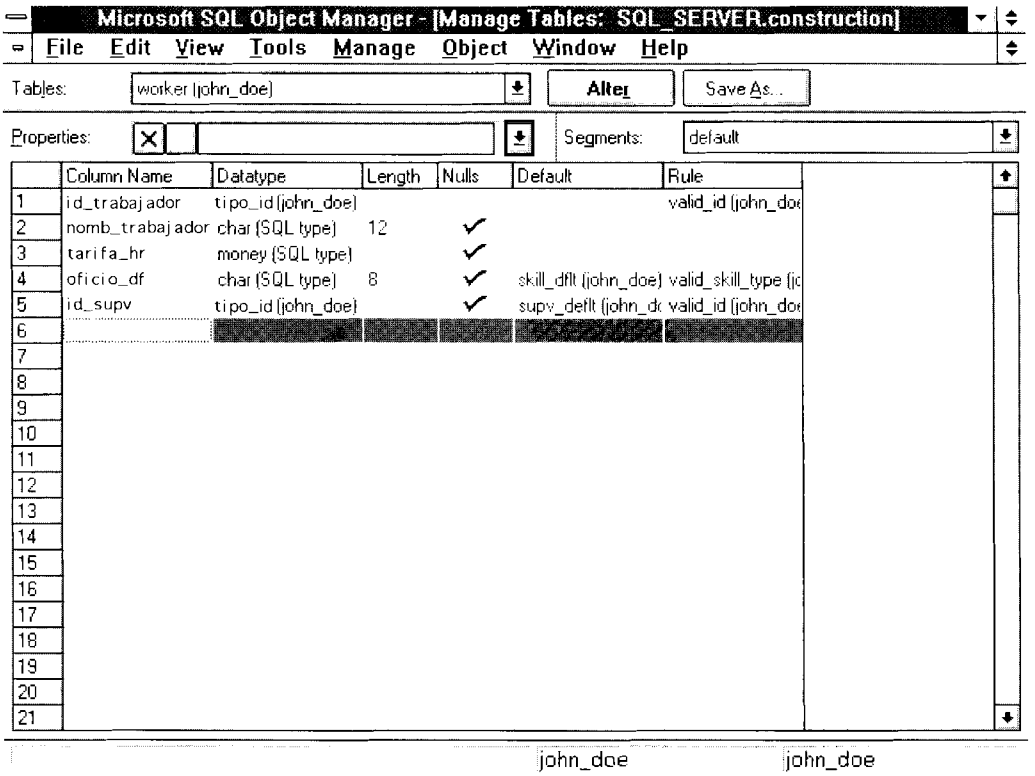
"tipo_id". Cualquier valor que se introduzca a la columna id_trabajador debe ser un entero, no nulo, y debe tener un valor entre 1 y 99.999, inclusive. En general, en la creación de reglas, la cláusula que sigue al "AS" puede ser la misma que cualquier cláusula WHERE válida de SQL, excepto que no puede incluir una subconsulta o hacer referencia a alguna otra columna. Su único parámetro se identifica por el símbolo @.

En resumen, hemos creado entonces un tipo de datos definido por el usuario, tipo_id, el cual es realmente un tipo de datos entero pero restringido a que no pueda ser nulo y a que debe estar entre 1 y 99.999. En SQL Server cualquier tipo de datos definido por el usuario tendrá un formato similar. Esto es, éste tendrá un tipo de datos soportado por el sistema que luego será restringido de alguna manera.

Definiendo tablas individuales

El próximo paso es definir las tablas con sus columnas individuales. Se ilustrará este proceso con la pantalla *Manages Tables* del SQL Server (Figura 9.5), al mismo tiempo analizarenos algunas de sus características interactivas. Luego se analizarán algunas características adicionales para la definición de datos que están disponibles en Oracle. El proceso de definición de la tabla se desarrolla de la forma siguiente:

1. Las columnas de la nueva tabla se definen en el área cuadriculada (*grid area*) que está debajo de la caja *Properties*. Observe que la definición de una columna ocupa toda una fila. Se muestran 21 filas de definición a la vez en la pantalla. El cuadriculado se puede mover hacia abajo si se necesitase definir más de 21 columnas.



2. La fila de definición de una columna consiste en *Column Name* (nombre de columna), *Datatype* (tipo de datos) y *Length* (longitud) y las indicaciones de las posibles restricciones: *Nulls* (nulos), un valor *Default* (por defecto), y *Rule* (regla) que se aplican a los valores de la columna.
3. Se define una tabla usando la orden Create Table (crear tabla), dándole un nombre a la tabla y definiendo cada uua de las columnas, incluyendo su tipo de datos, su longitud y las posibles restricciones.

Se comeuzará definiendo la tabla trabajador. En la caja *Colum Name* de la fila 1 entramos el nombre de la que será lenguaje la primera columna de la tabla, id_trabajador. se pulsa *tab* (tabulación) y se pasa al campo *Datatype* (tipo de datos). Puesto que el tipo de datos que se quiere para id_trabajador es el tipo de datos definido por el usuario tipo_id, se teclea esto aquí. Esto completa la definicióu de id_trabajador porque todas las otras cajas tendrán en cuenta lo que definimos para tipo_id. Podemos por tanto pasar a la próxima fila de la cuadrícula.

Ahora se introducirá nomb_trabajador y se designa que su tipo de datos es *character* (carácter) y se entra 12 como su longitud. Este campo puede ser nulo, luego se pone una marca en la caja *Nulls*. Esta columna no tendrá valor por defecto ni regla. El próximo nombre de columna es "tarifa_hr" y su tipo de datos es *money* (dinero), aquí no hay que especificar ninguna longitud. Esta columna también puede ser nula.

valor por defecto. Un valor que se entra automáticamente por el sistema cuando el usuario lo omita.

La próxima columna es "oficio" (character 8), que puede ser nula, y para la cual se quiere definir un valor por defecto y una regla. Expliquemos brevemente el concepto de valor por defecto para uua columna. Un **valor por defecto** se indica en la caja *Default*. Este valor se insertará automáticamente en la columna si el usuario falla al introducir un

valor. Por ejemplo, supongamos que tenemos una columna que nos da el número de teléfono de un trabajador. Si no se da ningún número al insertar un trabajador quisiéramos que el sistema ponga entonces automáticamente el número del conmutador central. Este sería en este caso el valor por defecto para la columna número de teléfono.

Obviamente, si se especifica un valor por defecto para una columna, *no* se introducirá automáticamente un valor nulo en el caso en que el usuario no introduzca explícitamente un valor. En su lugar se introducirá el valor por defecto. Por tanto, si se pone un valor en la caja *Default* (defecto), no importa lo que se haya especificado con relación al valor nulo. La entrada en la caja *Nulls* queda obsoleta si hay una entrada en la caja *Default*. Normalmente se pone una entrada en la caja *Nulls* cuando no se especifica ningún valor por defecto.

Definamos ahora un valor por defecto para la columna "oficio". Por defecto queremos que el oficio de un trabajador sea "Carpintero". Este valor por defecto se especifica a través de la caja *Manage Defaults* (gestionar valores por defecto), que es visualmente idéntica a la caja *Manage Rules* (Figura 9.4) y trabaja de la misma manera. Al introducir ahora:

```
CREATE DEFAULT oficio_df AS 'Carpintero'
```

se crea la definición de un valor por defecto. Este puede utilizarse en cualquier tabla en la que se quiera definir a 'Carpintero' como valor por defecto. En la pantalla de definición de la tabla se introdujo "oficio_df" en la caja *Default* para la columna oficio. En general, un valor por defecto debe ser una expresión constante. Es decir, no puede incluir variables, o tablas, o nombres de columna, sino solamente valores constantes, que pueden estar conectados con operadores aritméticos.

Ya estamos listos para definir una regla para esta columna que restrinja los posibles valores que puede introducir el usuario. Se quiere que el tipo de oficio esté limitado a los valores Electricista, Fontanero, Carpintero y Albañil. Por tanto, para ello hay que construir la regla:

```
CREATE RULE oficio_valido AS @oficio in ('Electricista', 'Fontanero',
'Carpintero', 'Albañil')
```

Después de crear esta regla, se entra su nombre, oficio_válido, en la caja *Rule* para terminar con la definición de la columna oficio.

La última columna de la tabla es id_supv. Ésta también tiene el tipo de datos tipo_id. Sin embargo, en este caso también queremos especificar un valor por defecto. Se deja como ejercicio especificar que el valor por defecto para id_supv es 3231 (ver Problema 3). Ya se puede crear la tabla. Las tablas restantes, Asignación y Edificio, se pueden crear de manera similar (ver Problema 5).

Definiendo claves primarias y externas. Para usar la posibilidad de que automáticamente el SGBD fuerce las reglas de integridad de entidades y de entidad referencial del modelo relacional, se deben definir las claves primarias y las claves externas. Este es un proceso bastante simple que se expone a continuación. Se usarán órdenes del lenguaje Oracle. Antes de empezar se debe señalar que las claves primarias y externas se definen como **restricciones** en Oracle. Esto es, son restricciones de los posibles valores de las tuplas y atributos en las tablas de la base de datos. Para este análisis se supone el esquema siguiente:

restricción. Una restricción de los valores posibles de tuplas y atributos en una base de datos.

```
trabajador (id_trabajador, nomb_trabajador, tarifa_hr, oficio,
            id_supv)
Foreign Key:          id_supv references trabajador
asignación (id_trabajador, id_edificio, fecha_inicio, Num_dias)
Foreign Keys:         id_trabajador references trabajador
                      id_edificio references trabajador
```

```
edificio (id_edificio, dir_edificio, tipo, nivel_calidad, categoria)
herramienta_asignacion (id_trabajador, id_edificio, herramienta,
fecha_inicio, fecha_terminacion)
Foreign Key:          id_trabajador, id_edificio, reference asignacion
```

Primero es necesario definir las claves primarias. En el lenguaje de órdenes de Oracle, las claves primarias se definen dentro del contexto de la definición de una tabla. Por ejemplo, supongamos que se está haciendo en la definición de la tabla trabajador:

```
CREATE TABLE trabajador
   (id_trabajador tipo_id CONSTRAINT llp_idtr PRIMARY KEY,
     ...
```

Esto nos muestra el comienzo de la definición de la tabla trabajador, donde se define la primera columna, id_trabajador, con el tipo de datos tipo_id. Se ha destacado en negrita la parte correspondiente a la definición de id_trabajador como una clave primaria. La palabra clave CONSTRAINT indica que lo que le sigue a continuación es la restricción de una columna. "llp_idtr" es el nombre que se le ha dado a esta restricción. Si luego hay algún intento de violar esta restricción, el sistema enviará un mensaje de error, identificando la restricción por su nombre. Después del nombre de la restricción, las palabras claves PRIMARY KEY indican que esta columna, id_trabajador, va a ser una clave primaria. Esto significa que su valor debe ser único dentro de la relación trabajador y que nunca puede ser nulo. Esta restricción se denomina **restricción de columna** porque se establece como parte de la definición de una columna.

restricción de columna. Una restricción que se establece en la definición de una columna en una tabla.

Ahora definamos la clave primaria de la tabla asignación. Esta clave consiste en dos columnas, id_trabajador e id_edificio, y por lo tanto no se puede definir como una restricción de columna. En este caso se debe definir como una restricción de tabla, como parte de la definición de la tabla, pero como parte de la definición de una columna. Esto se define en el contexto de la definición de la tabla:

```
CREATE TABLE asignacion
   (id_trabajador tipo_id,
   id_edificio tipo_id,
     ... ,
   CONSTRAINT llp_idtred PRIMARY KEY (id_trabajador, id_edificio)
```

Como en el ejemplo anterior, se ha destacado en negrita la definición de la restricción clave primaria. El nombre de la restricción es "llp_idtred". Las palabras reservadas, PRIMARY KEY, indican que lo que sigue entre paréntesis (id_trabajador, id_edificio) juntos constituyen una clave primaria para esta tabla. Observe que la definición no puede ser aplicable a una sola columna y que por lo tanto no es una restricción de columna. Esta es una **restricción de tabla** porque restringe los valores de estas columnas vistas como una unidad dentro de la tabla. Las columnas id_trabajador e id_edificio juntas deben tener un único valor en la tabla y ninguna de las dos puede tener valor nulo.

restricción de tabla. Una restricción que se aplica simultáneamente a múltiples columnas de una tabla.

Si asumimos que ya se han definido todas las claves primarias del esquema, entonces deben definirse las claves externas. Al igual que con las claves primarias, que pueden definirse a través de restricciones de columnas, si son claves externas de una columna, o como restricciones de tabla, si son claves externas multicolumnas. Para ver un ejemplo de clave externa de una sola columna, veamos de nuevo la definición de la tabla asignación:

```
CREATE TABLE asignacion
   (id_trabajador tipo_id, REFERENCES trabajador,
   id_edificio tipo_id, REFERENCES edificio,
    ... ,
```

Observe que esta vez no se ha usado la palabra clave CONSTRAINT. Podría haberse usado, pero en este caso es opcional. La palabra clave REFERENCES indica una definición de clave externa. La columna es una clave externa que referencia a la tabla que se

indica después de la palabra REFERENCES. Esto significa que su valor debe encontrarse como clave en dicha referida tabla. Por ejemplo, supóngase que la tabla asignación contiene la tupla

```
(1235, 515, ...)
```

Es decir, id_trabajador es 1235 e id_edificio es 515. Esto significa que 1235 debe ser una clave de alguna tupla en la tabla trabajador, y 515 debe ser una clave de una tupla en la tabla edificio. Si tales tuplas no existiesen en el momento en que se vaya a introducir la tupla de asignación, entonces el SGBD no permitirá que se añada esta tupla a la tabla asignación.

¿Qué pasa si se puede introducir esta tupla sin ningún problema, pero luego se elimina la tupla correspondiente en trabajador? En este caso, no se permite borrar la tupla de trabajador hasta que no se hayan borrado primero todas las tuplas en asignación que hacen referencia a dicha tupla en trabajador. Esto se conoce como la opción ON DELETE RESTRICT (restricción al borrar), y se aplica automáticamente a menos que se especifique otra cosa. Se *podría* especificar también ON DELETE CASCADE (borrar en cascada). Cuando está efectiva esta opción, las tuplas en asignación se borrarían automáticamente cuando se borren las tuplas a los que ellos se refieran en otras tablas. Veamos ahora cómo se define esto.

```
CREATE TABLE asignación
   (id_trabajador tipo_id, REFERENCES trabajador ON DELETE CASCADE,
   id_edificio tipo_id, REFERENCES edificio ON DELETE CASCADE,
     ... ,
```

Supongamos que se tiene en asignación la tupla

```
(1235, 515, ...)
```

en trabajador se tiene la tupla

```
(1235, ....),
```

y el usuario solicita borrar la tupla de trabajador. El SGBD borra automáticamente todas las tuplas en asignación que tengan 1235 como valor de id_trabajador. Pero si se hubieran omitido las palabras claves ON DELETE CASCADE, entonces el SGBD hubiera prohibido borrar la tupla de trabajador.

Como segundo ejemplo definamos la clave externa recursiva id_supv:

```
CREATE TABLE trabajador
   (id_trabajador tipo_id CONSTRAINT llp_idtr PRIMARY KEY,
     ...   id_supv tipo_id CONSTRAINT llf_idsp REFERENCES trabajador )
```

En este ejemplo se ha utilizado la palabra clave opcional CONSTRAINT y, por tanto, se exige también dar un nombre a la restricción (llf_idsp). Sin embargo, el aspecto más significativo aquí es que esta clave externa referencia a la propia relación; es decir, es recursiva. Como puede observarse, no hay ninguna diferencia entre definir una clave externa recursiva y definir otra clave externa cualquiera.

Todas estas claves externas se definen con restricciones de columna porque son claves externas de una sola columna. Se verá ahora una clave externa multicolumna. La tabla herramienta_asignación contiene una de estas claves. Esta tabla se utiliza para indicar cuáles herramientas usa un trabajador en un edificio. Puesto que un trabajador puede usar múltiples herramientas en un mismo edificio, estas asignaciones se indicarán en una tabla separada. Para ello se definirá, mediante una restricción de tabla, una clave externa que referencia a la relación asignación.

```
CREATE TABLE asignación_herramienta
   (id_trabajador tipo_id,
   id_edificio tipo_id,
    ... ,
   CONSTRAINT llf_idtred FOREIGN KEY (id_trabajador, id_edificio)
    REFERENCES asignación)
```

Probablemente ya habrá notado que todas estas definiciones de claves primarias y claves externas son muy elementales. Al igual que en la definición primera de clave externa, se puede tener la opción ON DELETE CASCADE.

Después que se definan todas las claves primarias y externas, el SGBD Oracle automáticamente fuerza que lo sean. Por tanto, cuando se entren datos, el SGBD se asegura que las claves primarias no sean nulas y que sean únicas, y que las claves externas tengan valores que correspondan a claves primarias válidas en las relaciones a las que refieren. Con SQL Server esta seguridad no es automática, sino que debe implementarse con mecanismos adicionales de disparadores (*triggers*). Estos disparadores se expondrán posteriormente.

Restricciones CHECK. En la definición de datos en Oracle se usan restricciones de columna y de tabla para definir las claves. También se pueden usar las restricciones de tabla y columna de una forma más general para definir otras reglas. Consideremos, por ejemplo, la relación siguiente:

```
empleado (id_emp, nomb_emp, dir_emp, salario, bonificacion)
```

Supongamos que la compañía tiene la política de que ningún empleado puede recibir una bonificación que exceda el 10 por 100 del salario. Esta sería una restricción general que quisiéramos que el SGBD garantice. Esto se puede hacer a través del mecanismo de una restricción CHECK:

```
CHECK (bonificacion <= 0.10 * salario)
```

restricción CHECK. Una restricción general, basada en una expresión condicional, que se impone sobre una columna o sobre una tabla.

La **restricción CHECK** puede ser parte de la definición de una columna o puede ser una restricción de la tabla. Observe, como en este caso, que aunque en la restricción se hace referencia a más de una columna, ella puede seguir formando parte de la definición de una columna. Una restricción CHECK puede afectar a cualquier columna de la tabla, y no solamente a la columna de la cual es parte de la definición. Sin embargo, tiene más sentido definir una restricción CHECK que se refiera a más de una columna, como una restricción de tabla, ya que ésta se aplica más bien a la tupla como un todo que a una sola columna de la tupla.

Las restricciones CHECK pueden aplicarse también a una sola columna. De esta manera ellas funcionarían como reglas del SQL Server. Por ejemplo, la regla siguiente

```
CREATE RULE oficio_valido AS @oficio in ('Electricista', 'Fontanero',
'Carpintero', 'Albañil')
```

En Oracle se puede lograr el mismo efecto definiendo una restricción CHECK como parte de la definición de la columna oficio:

```
CHECK (oficio IN ('Electricista', 'Fontanero', 'Carpintero',
'Albañil')
```

Las restricciones CHECK dan más poder que las reglas del SQL Server, puesto que pueden usarse con múltiples columnas. Sin embargo, éstas no son tan poderosas como los disparadores (como se verá posteriormente) porque están limitadas a comprobar los valores en un sola tupla a la vez. Es decir, ninguna restricción CHECK puede contener una

subconsulta que se refiera a valores en otras tuplas de la misma relación o en tuplas de alguna otra relación. Por tanto, la restricción CHECK

```
CHECK (bonificacion <= 0,10 * salario )
```

siempre comparará el atributo bonificación de una tupla con el atributo salario en la misma tupla.

Lenguaje de definición de datos: Sumario y ejemplo final

En esta sección sobre el lenguaje de definición de datos se ha visto el proceso de definición de tipos de datos, tablas y columnas. En esta discusión se observa que la mayor parte del esfuerzo se dedicó a los diferentes tipos de restricciones. Por consiguiente, vale la pena hacer un análisis final sobre las diferentes restricciones disponibles.

Cuando se define un tipo de datos o una columna en una tabla, por lo general siempre quisiéramos poner una restricción sobre los valores que se permitirán en él. Esto se hace en parte al escoger el tipo de datos básico, tal como *integer*, *datetime*, o *character*. Pero hay otros tipos de restricciones que a menudo se desean. Por ejemplo, es normal querer que una columna clave no sea nula. También con frecuencia se desea que un valor entero sea positivo.

Tanto SQL Server como Oracle brindan medios para definir una variedad de restricciones. SQL Server hace esto con las definiciones de restricciones para nulos, valores por defecto y reglas a través de la pantalla de *Manage Tables*. También proporciona medios para una forma más poderosa de definir restricciones a través de los disparadores (*triggers*), que no son más que procedimientos que se ejecutan cuando se haga una actualización en tablas específicas de la base de datos.

La caja *Nulls* se usa para indicar si se permiten valores nulos para los datos de una columna. La caja *Default* se usa para indicar un valor que el sistema debe poner cuando el usuario no introduzca ningún valor para dicha columna. La caja *Rule* indica que se debe aplicar una regla a los valores que se entren para la columna. Por ejemplo, si la columna representa la tarifa salarial por hora, y se desea prohibir que ésta sea menor que $5.00 por hora, podría especificarse una regla para tal efecto. Sin embargo, debe señalarse que una regla no puede hacer referencia a un valor de ninguna otra columna que no sea para la que se está especificando la regla. Por tanto, las reglas en SQL Server no tienen el poder de las restricciones CHECK en Oracle.

Con el lenguaje de órdenes de Oracle se pueden lograr características similares, así como otras adicionales, claves primarias y externas de una o más columnas, restricciones de unicidad y restricciones CHECK. En la Figura 9.6 se resumen todos estos tipos diferentes de restricciones. En general, la instrucción CREATE TABLE de Oracle a estos efectos tiene la estructura siguiente:

```
CREATE TABLE nombre_de_tabla
   (descripciones_de_columnas, restricciones_de_tablas)
```

donde una descripción_de_columna es (los elementos entre corchetes[ ] son opcionales):

```
descripción_de_columna:
   nombre_de_columna tipo_de_dato [DEFAULT expresión]
   [restricciones_de_columna]
```

y una restricción de columna es:

```
restricción_de_columna:
   [CONSTRAINT nombre_de_restricción] [NOT] NULL
```

```
SQL Server
    Null / Non-Null
    Defaults (por defecto)
    Rules (reglas)

Oracle
    Null / Non Null
    Defaults
    Uniqueness (unicidad)
    Primary Key (clave primaria)
    Foreign Key (clave externa)
    CHECK
    Table Constraints (restricciones de tabla)
```

```
(ó)             UNIQUE
(ó)             PRIMARY KEY
(ó) REFERENCES nombre_de_tabla [ON DELETE CASCADE]
(ó)             CHECK (condición)
```

y una restricción_de_tabla tiene la forma:

```
restricción_de_tabla:
   [CONSTRAINT nombre_de_restricción] UNIQUE (lista de
     nombres_de_columna)
              (ó)  PRIMARY KEY (lista de nombres_de_columna)
              (ó)  FOREIGN KEY (lista de nombres_de_columna)
                   REFERENCES nombre_de_tabla [ON DELETE CASCADE]
              (ó)  CHECK (condición)
```

Veamos un último ejemplo:

```
CREATE TABLE empleado
   (id_emp integer CONSTRAINT llp_id_emp PRIMARY KEY,
   nomb_emp char(15) UNIQUE,
   telef_emp char(7) DEFAULT '5551919',
   id_emp_supv REFERENCES empleado ON DELETE CASCADE,
   oficio_emp char(8) CONSTRAINT rest_oficio_emp NOT NULL
     CONSTRAINT ck_oficio_emp
    CHECK (oficio_emp IN ('Electricista', 'Fontanero',
    'Carpintero', 'Albañil' )
```

Este ejemplo muestra instancias de todos los tipos diferentes de restricciones de columnas (PRIMARY KEY, UNIQUE, FOREIGN KEY, NOT NULL, y CHECK), así como DEFAULT. Todas éstas han sido explicadas previamente con excepción de la restricción de unicidad (UNIQUE), que ha sido usada aquí con la columna nomb_emp. La **restricción UNIQUE** garantiza que los valores en la columna deben ser únicos. Esto es, no hay dos tuplas que tengan el mismo nombre de empleado. Sin embargo, esto no significa que la columna sea una clave, ya que sí es posible que la columna tenga valor nulo. Las dos restricciones NOT NULL y UNIQUE juntas son equivalentes a la restricción PRIMARY KEY. En este ejemplo también se muestra que una columna puede tener múltiples restricciones y que es opcional darle un nombre a una restricción.

restricción UNIQUE. Una restricción que garantiza que dos tuplas de una misma relación no tengan el mismo valor en una columna.

La restricción UNIQUE aplicada a múltiples columnas en una restricción de tabla y tiene el mismo significado que si se aplica a cada columna en una restricción de columna. Todos los restantes recursos de restricciones de tabla fueron ilustrados anteriormente con ejemplos en esta misma sección.

▼ Manipulación y programación del servidor de datos

SQL Server y Oracle soportan las características estándares de SQL, así como un número de mejoras útiles en sus lenguajes interactivos. Sin embargo, además de esto, ambos proporcionan estructuras de control que permiten los programas tipo lotes (*batch*), en el que se ejecutan sin interrupción un conjunto o *lote* de instrucciones. Estas estructuras de control son parte de los lenguajes de **flujo-de-control** del SQL Server y del **PL/SQL** de Oracle que incluyen instrucciones condicionales (IF) e instrucciones para hacer bucles. También se pueden usar variables para almacenar y calcular valores.

lenguaje flujo-de-control. Lenguaje de manipulación de datos de SQL Server.

PL/SQL. Lenguaje de manipulación de datos de Oracle.

Adicionalmente a estas características, estos lenguajes también brindan **procedimientos almacenados**, que son programas escritos que se compilan a lenguaje de máquina y se guardan para una ejecución repetida más eficiente. Los procedimientos almacenados también permiten el paso de parámetros de entrada y salida, por tanto le permiten al programador desarrollar sistemas modulares y definir módulos de utilidades que pueden ser utilizados en múltiples aplicaciones.

procedimiento almacenado. Un programa compilado en lenguaje de máquina y que se guarda para una ejecución repetida y más eficiente.

En esta sección se definirán y analizarán estos recursos lingüísticos. Aunque se usará la sintaxis y las ideas del SQL Server, debe observarse que en general en Oracle existen las mismas capacidades, aunque de forma diferente.

Lenguaje de flujo-de-control

Como lenguaje de consulta, SQL fue concebido originalmente para usarse interactivamente por usuarios relativamente sofisticados que estuviesen interesados en obtener información inmediata por la vía de las consultas a bases de datos. Sin embargo, el poder de un lenguaje relacional es tal que su utilización no debería confiarse solamente a esta aplicación. Esto tiene gran poder y utilidad, ya que pueden aplicarse a tareas más tradicionales de programación. Para hacer esto se debe mejorar el lenguaje para permitir la ejecución de varias instrucciones, una después de otra e incluir instrucciones de control condicionales e iterativas. Estos recursos se ofrecen mediante el lenguaje de flujo-de-control.

El lenguaje de flujo-de-control tiene las características siguientes (resumidas en la Figura 9.7), que se examinarán en esta sección:

1. Instrucciones BEGIN .. END que definen bloques de instrucciones SQL para ser tratados como unidades de ejecución.
2. Instrucciones IF ... ELSE para la ejecución condicional.
3. Instrucciones WHILE para la ejecución repetitiva o iterativa.
4. Instrucciones BREAK y CONTINUE para la salida anticipada de bucles WHILE.
5. Instrucciones DECLARE que permiten la definición de variables locales.

```
BEGIN ... END
IF-ELSE
WHILE con BREAK y CONTINUE
DECLARE con variables locales
RETURN
PRINT
Comentarios
```

6. Instrucciones RETURN que permiten la definición de módulos o subrutinas que se pueden llamar por otros módulos. (Las instrucciones RETURN se usan con los procedimientos almacenados que se analizarán posteriormente.)
7. Instrucciones PRINT que permiten enviar mensajes a los usuarios.
8. Comentarios que permiten incluir documentación interna en los programas.

Bloque de instrucciones BEGIN ... END. Las instrucciones condicional (IF) e iterativa (WHILE) controlan la ejecución de una instrucción simple o de un bloque de instrucciones. Un bloque de instrucciones se indica de la forma siguiente:

```
BEGIN
   instrucción SQL
   ...
   instrucción SQL
END
```

Se verán varios ejemplos que ilustran el uso de esta construcción. Por simplicidad se definirá un bloque de instrucciones como sigue:

bloque instrucción. O una sola instrucción SQL sin delimitadores o un conjunto de dos o más instrucciones SQL delimitadas por BEGIN ... END.

Un **bloque instrucción** es o una instrucción simple de SQL sin delimitadores o un conjunto de dos o más instrucciones SQL delimitadas por BEGIN y END.

Ejecución Condicional IF ... ELSE. La **instrucción IF** en SQL Server tiene la sintaxis siguiente:

instrucción IF. Una instrucción cuya ejecución depende de la veracidad de una condición.

```
IF <expresión condicional>
   <bloque de instrucciones>
[ ELSE
   <bloque de instrucciones> ]
```

Los corchetes que encierran la parte ELSE de la instrucción indican que ésta es opcional.

Esta estructura de la instrucción no parece inusual, pero de hecho lo es. La expresión condicional de esta instrucción IF la hace diferente de las instrucciones IF de otros lenguajes de programación. Particularmente, la expresión condicional, la cual debe poderse evaluar a verdadero o falso, contendrá a menudo subconsultas. De este modo, ésta puede utilizarse para determinar condiciones relativas a la relación en su totalidad antes de aplicar actualizaciones a la relación como un todo. Veamos un ejemplo:

```
IF (select avg(tarifa_hr) from trabajador) < 10,00
   update trabajador
     set tarifa_hr = tarifa_hr + .50
```

Se ha destacado la expresión condicional de esta instrucción para su explicación posterior. Observe que contiene una consulta que calcula la tarifa salarial media de los trabajadores y compara el resultado de la consulta con $10,00. La diferencia significativa entre esta condición y la que se usa normalmente en una instrucción IF es que en este caso se lleva a cabo un cálculo sobre la relación trabajador *completa* para determinar la tarifa salarial media antes de decidir si ejecutar la instrucción de actualización. Normalmente, una instrucción IF mira simplemente el valor de una o más columnas de una *sola* tupla. Permitiendo a la instrucción IF ver a la relación como un todo se incrementa significativamente la potencialidad del lenguaje. Por supuesto, usando la cláusula WHERE de instrucciones individuales de SQL podemos seguir tomando decisiones basadas en tuplas individuales y realizar actualizaciones sobre esas tuplas. Pero ahora

tenemos capacidades adicionales de usar cálculos sobre relaciones para tomar decisiones.

La expresión condicional que controla la ejecución de una instrucción IF puede incluir constantes, variables locales y subconsultas. Puede incluir nombres de columnas sólo como parte de las subconsultas. Cualquier otra referencia a un nombre de columna será ambiguo, ya que no estaría claro a qué tupla se estaría refiriendo. Una subconsulta sólo puede usarse si sigue a la palabra clave EXISTS o si produce un valor simple (como en el ejemplo precedente) y puede, por consiguiente, compararse con una constante o con una variable local.

Un segundo ejemplo nos muestra que se puede usar más de una instrucción select en una única expresión condicional. Supongamos que se quiere incrementar la tarifa salarial por horas de los fontaneros si su media actual está en más de un dólar por debajo de la media de los albañiles:

```
IF (select avg(tarifa_hr) from trabajador
    where oficio = 'Fontanero') <
  (select avg(tarifa_hr) from trabajador
    where oficio = 'Albañil') + 1,00
 update trabajador
    set tarifa_hr = tarifa_hr + .50
    where oficio = 'Fontanero'
```

De nuevo se ha destacado la expresión condicional. Como puede verse, ésta contiene dos consultas, cada una de las cuales produce una tarifa salarial media. Se hace una comparación entre las dos, y si la comparación se cumple, se ejecuta la actualización.

Como tercer ejemplo, para ilustrar el operador EXISTS, supongamos que se le quiere dar un aumento a los que son supervisados por H. Rickover:

```
IF EXISTS (select * from trabajador where id_supv in
       (select id_trabajador from trabajador
         where nomb_trabajador = 'H. Rickover'))
   BEGIN
     update trabajador
       set tarifa_hr = tarifa_hr + .50
     print '¡¡¡Esto hay que celebrarlo!!!'
   END
ELSE
   BEGIN
     update trabajador
       set tarifa_hr = tarifa_hr - .50
     print '¡Lo siento, muchachos, pero así están las cosas!'
   END
```

Este ejemplo es algo complejo, por lo que veremos algunas explicaciones adicionales. La condición contiene una consulta con una subconsulta:

```
select * from trabajador where id_supv in
       (select id_trabajador from trabajador
          where nomb_trabajador = 'H. Rickover')
```

La subconsulta

```
(select id_trabajador from trabajador
          where nomb_trabajador = 'H. Rickover')
```

nos da el id_trabajador de H. Rickover. La consulta principal seleccionará entonces todas las tuplas que tienen a H. Rickover como supervisor. Por último, la condición completa

```
EXISTS (select * from trabajador where id_supv in
       (select id_trabajador from trabajador
          where nomb_trabajador = 'H. Rickover') )
```

será verdadera sólo si hay algún trabajador que tenga a H. Rickover como supervisor. Esto corresponde con la condición que se estableció al principio. Ahora, si existen tales trabajadores, les queremos dar un aumento de $0,50.

En este ejemplo se ilustró también la parte ELSE de la instrucción IF, la orden print (imprimir) y el uso de BEGIN ... END para crear un bloque de instrucciones. El ELSE se ejecuta sólo si la expresión condicional es falsa. En este caso se le reduce la tarifa salarial en $0,50. La orden print le envía un mensaje al usuario. Este mensaje se delimita entre comillas simples. Si un apóstrofo es parte de un mensaje, entonces debe indicarse con dos apóstrofes seguidos.

Las instrucciones IF se pueden anidar dentro de otras instrucciones IF, siguiendo al IF o al ELSE. El número de niveles anidados no está limitado.

Las instrucciones SQL trabajan sobre relaciones como un todo, por tanto sólo es lógico que las instrucciones IF que controlen su ejecución se basen en condiciones que a su vez se aplican a relaciones como un todo. Los ejemplos previos ilustran esto.

instrucción WHILE. Una instrucción que usa una condición para controlar la ejecución iterativa de un bloque de instrucciones.

Instrucciones WHILE: Ejecución iterativa. La sintaxis de la **instrucción WHILE** es la que sigue:

```
WHILE <expresión condicional>
   <bloque de instrucciones>
```

La instrucción WHILE provoca la ejecución repetida del bloque de instrucciones, mientras que la expresión condicional evalúe verdadero. La expresión condicional en esta instrucción tiene la misma definición que en la instrucción IF. Veamos un ejemplo.

Supongamos que se desea duplicar el pago de todo el mundo mientras el salario medio de los fontaneros esté por debajo de los $20,00. Si la tarifa salarial media está por encima de los $20,00 no se hace nada, de lo contrario se le dobla la paga a todo el mundo:

```
WHILE (select avg(tarifa_hr) from trabajador
   where oficio = 'Fontanero') < 20,00
  BEGIN
   print '¡¡Le estamos doblando la paga!!'
   update trabajador
     set tarifa_hr = 2 * tarifa_hr
  END
```

De nuevo, como con la instrucción IF, la condición que controla la ejecución del lazo WHILE contiene una consulta que realiza el cálculo sobre la relacional trabajador como un todo. Mientras esta condición sea verdadera, se ejecutarán las instrucciones dentro del bucle WHILE. De este modo, supongamos que la tarifa salarial media de los fontaneros es $4,50. En este caso se doblará la tarifa salarial de todos y se imprimirá el mensaje "¡¡Le estamos doblando su paga!!". Ahora la tarifa media de los fontaneros será de $9,00. Puesto que sigue siendo menor de $20,00, todas las tarifas se duplicarán de nuevo, y el mensaje volverá a aparecer. Ahora el salario medio de los fontaneros es de $18,00. Una vez más éste continúa por debajo $20,00, por tanto la tarifa de todos se duplicará por tercera y última vez. Y, por supuesto, el mensaje se imprimirá una tercera vez también. (Bello programa, ¿no quisiera que *su* jefe tuviese uno igual?)

Una vez que el bloque de instrucciones comienza a ejecutarse dentro del bucle WHILE, éste continuará su ejecución hasta el final. Aún si el valor de la expresión

condicional cambia a falso antes del fin del bloque de instrucciones, la ejecución continúa. Después que el bloque de instrucciones se haya ejecutado completamente, la expresión condicional se evalúa de nuevo, y si es verdadera se repite la ejecución del bloque de instrucciones. Si la expresión evalúa falso, la ejecución comienza en la instrucción después del bloque de instrucciones del WHILE. Sin embargo, bajo determinadas circunstancias, podría preferirse abortar el bucle WHILE o ir al comienzo de éste sin terminar la ejecución del bloque. En la sección siguiente se ve cómo se puede hacer esto.

BREAK. Palabra clave que causa la salida de la ejecución de un bucle WHILE.

CONTINUE. Palabra clave que causa que el control de ejecución de un lazo WHILE retorne a la primera instrucción del lazo.

Usando BREAK y CONTINUE. La palabra clave **BREAK** le dice al sistema que salga inmediatamente de un bucle WHILE y continúe la ejecución en la instrucción siguiente al bloque de instrucciones del WHILE. **CONTINUE** le dice que ignore el resto del bloque de instrucciones, que vuelva evaluar la expresión condicional y, si es verdadera, vuelva a ejecutar el bloque de nuevo. Esto se aclarará con un ejemplo detallado. Supongamos que se quiere repetir un bloque de código mientras la tarifa salarial de todos esté por debajo de $40,00:

```
(I) WHILE (select max(tarifa_hr) from trabajador) < 40,00
    BEGIN
(II) update trabajador
       set tarifa_hr = 1.1 * tarifa_hr
(III) IF (select avg(tarifa_hr) from trabajador) < 20,00
       CONTINÚE
(IV) IF (select min(tarifa_hr) from trabajador) > 15,00
       BREAK
(V) update trabajador
       set tarifa_hr = tarifa_hr + 1,00
       where oficio = 'Fontanero'
    END
(VI) .......
```

Para facilitar las referencias se han etiquetado con números romanos las instrucciones principales de este ejemplo. Este bucle WHILE será ejecutado mientras la tarifa salarial máxima que se le pague a un trabajador esté por debajo de los $40,00 [instrucción (I)]. La instrucción (II) muestra que se quiere aumentar el salario en un 10 por 100 cada vez que se repita el bucle. La instrucción (III) comprueba si el salario medio de todos los trabajadores está por debajo de $20,00. En caso afirmativo se ejecuta el CONTINUE, lo que significa que nos saltamos el resto de las instrucciones del lazo y el control pasa de nuevo a la instrucción (I). Una vez que la media salarial haya alcanzado los $20,00 se ejecuta la instrucción (IV). Si la tarifa salarial mínima es mayor que $15,00 se ejecuta un BREAK pasándole el control a la instrucción (VI) (la cual no se especifica en el ejemplo). Si la tarifa mínima no sobrepasa los $15,00, entonces se ejecuta la instrucción (V) y se le añade con ello $1.00 a la tarifa de cada fontanero. Después de la instrucción (V), el control retorna de nuevo a la instrucción (I), donde el sistema comprueba de nuevo si la tarifa salarial máxima es menor que $40,00. Si lo es, el lazo se ejecuta de nuevo. En caso contrario, el control pasa a la instrucción (VI).

variable local. Una variable definida para usar dentro de un procedimiento y almacenar valores temporales de trabajo.

DECLARE y Variables locales. Ningún lenguaje de programación estaría completo sin **Variables locales** en las cuales poner temporalmente los valores de trabajo. En el lenguaje de SQL Server, tales variables se declaran con un tipo de datos del sistema o definido por el usuario, se les asigna un valor mediante instrucciones select y se usan dentro del mismo procedimiento.

Veamos ahora un ejemplo que usa una variable local para determinar la ejecución de un bucle (lazo) WHILE.

Una variable local se declara haciendo que el nombre comience con @:

```
declare @indice int /* "int" significa integer */
select @indice = 3
WHILE @indice > 0
   BEGIN
     delete trabajador where tarifa_hr > 20,00
     update trabajador set tarifa_hr = tarifa_hr * 1.1
     select @indice = @indice - 1
   END
```

Este ejemplo ejecuta el bucle WHILE exactamente tres veces. Cada vez se borran todos los trabajadores que hacen más de $20,00 por hora y se incrementa el salario de todos los demás en un 10 por 100. La primera línea también ilustra cómo se incorporan comentarios a los programas.

Procedimientos almacenados

Los procedimientos almacenados son programas SQL que fueron compilados la primera vez que se ejecutaron y luego se almacenaron para su uso posterior. Ofrecen una serie de ventajas sobre los programas que se escriben, se ejecutan inmediatamente y luego nunca más se usan: (1) Los procedimientos ya compilados se ejecutan muy rápidamente. (2) Estos pueden recibir y retornar **parámetros** —variables utilizadas para pasar datos hacia dentro y hacia afuera de un procedimiento almacenado— que hacen posible usar módulos utilitarios o dividir en módulos programas grandes y complejos. Veamos un ejemplo.

parámetro. Una variable que se usa para pasar o recibir datos de un procedimiento almacenado.

Supóngase que frecuentemente se desea contar el número de trabajadores en la relación trabajador. Después de definir una variable local @cant_trabajador, *podríamos* escribir la consulta:

```
select @cant_trabajador = count (*) from trabajador
```

O en su lugar se podría crear un procedimiento:

```
create procedure cant_trabajadores @cant_tr int output
```

como

```
select @cant_tr = count(*) from trabajador
```

La primera línea de esta instrucción

```
create procedure cant_trabajadores @cant_tr int output
```

da "cant_trabajadores" como nombre del procedimiento e inmediatamente después define los parámetros con sus tipos de datos y establece si son de entrada o de salida. En este ejemplo, @cant_tr es un parámetro de tipo integer ("int") que se usa como salida. Veamos ahora cómo trabajaría este procedimiento.

Cuando se llame al procedimiento se ejecuta su parte de consulta

```
select @cant_tr = count(*) from trabajador
```

y el resultado se pone en la variable de salida @cant_tr y se retorna al programa que llamó. Los parámetros de salida se identifican poniendo la palabra "output" o "out" después de la definición del tipo de datos del parámetro. Los parámetros que no se especifi-

quen como de salida se considerarán parámetros de entrada. La palabra clave "as" señala el final de la definición de los parámetros y el comienzo de la definición del procedimiento. Todo lo que esté después del "as" constituye la parte ejecutable del procedimiento.

Ahora veamos cómo se ejecuta este procedimiento. Primero se define la variable local @cant_trabaj como una variable entera que recibirá el valor de salida del procedimiento. Luego se ejecuta el procedimiento:

```
declare @cant_trabaj int
execute cant_trabajadores @ cant_trabaj output
```

Esta instrucción execute (ejecuta) provoca la ejecución del procedimiento cant_trabajadores definido anteriormente y que el resnltado de este procedimiento se ponga en la variable local @cant_trabaj.

Se puede ver que este enfoque es ligeramente más simple que escribir cada vez la consulta de salida

```
select @cant_tr = count (*) from trabajador
```

La ventaja de usar procedimientos almacenados es significativamente mayor en la medida en que el procedimiento sea más grande y más complejo.

Como un segundo y más complejo ejemplo supongamos que se quiere un procedimiento almacenado que calcule la tarifa salarial media de los trabajadores identificados por un oficio. Es decir, el programa que llama le pasa un tipo de oficio al procedimiento almacenado y éste retorna la tarifa salarial media de todos los trabajadores que tengan ese oficio. Este procedimiento calc_salario_prom se crearía como sigue:

```
create procedure calc_salario_prom @tarifa_prom money output,
   @ tipo_oficio char(8)
as
select @tarifa_prom = avg(tarifa_hr)
from trabajador
where oficio = @tipo_oficio
```

Este procedimiento almacenado tiene un parámetro de salida, @tarifa_prom y un parámetro de entrada @tipo_oficio. El programa que llama debe indicar una variable local de tipo *money* (dinero) para que reciba el valor de salida y debe indicar un valor de tipo oficio para pasar como parámetro de entrada. El debe indicar ambos parámetros en el mismo orden en que fueron listados cuando se creó el procedimiento. A continuación se da un ejemplo de ejecución de calc_salario_prom que hace todo esto:

```
declare @salario_prom money
exec calc_salario_prom @salario_prom output, Fontanero
```

Observe que "exec" se puede utilizar como abreviatura de "execute". Aunque el valor de entrada "Fontanero" es un dato de tipo character (carácter) no es necesario ponerlo entre comillas a menos que tenga un espacio en blanco, un signo de puntuación o que comience con un dígito. El procedimiento calc_salario_prom usará 'Fontanero' como valor para @tipo_oficio, por lo que esto se ejecutará como:

```
select @tarifa_prom = avg(tarifa_hr)
from trabajador
where oficio = 'Fontanero'
```

Esto calculará la tarifa por horas media de los fontaneros y cuando retorne al programa que llamó el resultado se dejará en la variable @salario_prom.

valor por defecto del parámetro. El valor de un parámetro que suministra el sistema si el programa que llama lo omite.

Valores por defecto. En la definición de un procedimiento almacenado es posible especificar **valores por defecto de los parámetros**. Si el programa que llama omite pasar un valor a un parámetro de entrada, entonces el programa usará el valor por defecto para dicho parámetro. El valor por defecto puede ser cualquier valor válido para el tipo de datos del parámetro, incluyendo nulo. Veamos un ejemplo que utiliza el nulo. Es meramente una modificación del ejemplo anterior. En este caso, si el programa que llama sólo especifica el parámetro de salida, pero no pone el tipo de oficio, se calculará la media salarial de *todos* los trabajadores. El procedimiento modificado quedaría en la forma:

```
create procedure calc_salario_prom @tarifa_prom money output,
   @ tipo_oficio char(8) = null
as
if @tipo_oficio = null
   select @tarifa_prom = avg(tarifa_hr)
   from trabajador
else
   select @tarifa_prom = avg(tarifa_hr)
   from trabajador
   where oficio = @tipo_oficio
```

Si se compara esta versión del procedimiento con la versión anterior se verá que el valor por defecto se define inmediatamente después que la definición del tipo de datos del parámetro:

```
@ tipo_oficio char(8) = null
```

Poniendo "= null" después de la definición del parámetro estamos queriendo decir que, si no se le pasa ningún valor al parámetro, se debe asumir que el valor de éste es nulo. La parte ejecutable del procedimiento se debe modificar para manejar la posibilidad de que el programa que llama no pasa ningún tipo de oficio.

Uso del RETURN. Cuando se ejecuta la última instrucción de un procedimiento, entonces éste termina y retorna el control a quien llamó. ¿Qué hacer si por la lógica del procedimiento se desea salir antes? La instrucción RETURN causa que el procedimiento almacenado termine inmediatamente y retorne el control al programa que lo llamó. Supóngase que se desea combinar una variedad de funciones en un simple programa almacenado. Por ejemplo, queremos permitirle al usuario solicitar la tarifa salarial máxima, mínima y media de trabajador. Un procedimiento para hacer esto sería:

```
create procedure funcs_calc_salario @tipo_func char(3),
       @ val_ret money output
as
   if tipo_func = "max"
     begin
       select @val_ret = max(tarifa_hr) from trabajador
       return
     end
   if tipo_func = "min"
     begin
       select @val_ret = min(tarifa_hr) from trabajador
       return
     end
   if tipo_func = "avg"
     begin
       select @val_ret = avg(tarifa_hr) from trabajador
       return
     end
```

En este ejemplo el programa que llama ha solicitado una de tres funciones. Si la función es máx se calcula este valor y se retorna inmediatamente, ya que no se desea calcular ninguno de los otros dos. Es fácil ver a partir de este ejemplo cómo se puede usar la instrucción RETURN en los procedimientos almacenados.

Una introducción a los disparadores

disparador. Un programa que se ejecuta automáticamente cuando se intenta hacer una determinada actualización sobre una tabla específica.

Un **disparador** (*trigger*) es un programa que se ejecuta automáticamente cuando se intenta hacer una actualización determinada sobre una tabla específica. Se pueden definir tres tipos de disparadores para cada tabla: de inserción, de actualización y de borrado. Puesto que lo que se está dando es una introducción al concepto de disparador, se continuará esta discusión con ejemplos de SQL Server. Los disparadores de Oracle son algo más sofisticados. Al final de la discusión se resumirán algunas de las diferencias que existen en los disparadores de Oracle.

Supóngase que se quiere mantener una columna en la base de datos que se deriva del cálculo de otras columnas. Por ejemplo, si tenemos una tabla vendedor en la base de datos, se puede querer tener en la base de datos el total de comisiones ingresadas por cada vendedor en el último mes. O en la base de datos de la constructora Premier podíamos querer mantener el estimado de cuántos días un trabajador ha sido planificado para un trabajo. O usando el concepto de días de ocho horas y la tarifa por hora de cada trabajador, calcular cuánto habría que pagarle al trabajador por esos días. Asumamos que se quiere mantener este último cálculo, se pondrá éste en una columna adicional en la tabla trabajador de nombre "pago_acumulado".

Antes de mostrar las instrucciones concretas que se usan para definir los disparadores (*trigger*), es necesario dar una mayor explicación de cómo trabajan los disparadores en SQL Server. Cada vez que se lleva a cabo una inserción, un borrado, o una actualización sobre una tabla de la base de datos se crean nuevas versiones de tablas controladas por el sistema. Estas **tablas de verificación de disparadores** (*trigger test tables*) se llaman *inserted* (insertados) y *deleted* (borrados)[2]. Si se insertan filas en una tabla entonces *inserted* consiste en las filas que han sido insertadas y *deleted* está vacía. Si se borran filas de una tabla, entonces *deleted* consiste en las filas que se han borrado e *inserted* es vacío. Si se actualizan filas en una tabla, entonces *deleted* consiste en las viejas versiones de las filas actualizadas e *inserted* está formado por las nuevas versiones de las mismas filas. Esta información es particularmente valiosa cuando se usa con el hecho de que en SQL Server un disparador se inicia *después* que ocurre un evento de actualización de fila. De este modo, cuando se ejecuta el disparador, los archivos *inserted* y *deleted* ya existen.

tablas de verificación de disparadores. Tablas actualizadas con las tuplas insertadas y borradas cada vez que ocurre una actualización en una tabla correspondiente de la base de datos.

Se verá ahora cómo implementar el disparador que se ha descrito anteriormente. Asumamos que la tabla trabajador ha sido mejorada para incluir la columna "pago_acumulado":

```
trabajador(id_trabajador, nomb_trabajador, tarifa_hr, oficio,
      id_supv, pago_acumulado)
```

La fórmula para calcular el pago acumulado es:

```
pago_acumulado = num_total_dias * 8 * tarifa_hr
```

donde núm_total_días se calcula para dicho trabajador a partir de la tabla de asignación.

Sólo tendremos que ver con las actualizaciones de la tabla asignación. Cada vez que se añada una nueva tupla, o se actualice o borre una tupla existente, se deseará actualizar el pago_acumulado del respectivo trabajador. Por tanto, queremos un disparador que responda a las actualizaciones de la tabla asignación actualizando una columna en la tabla trabajador.

¿Cómo trabaja este disparador? Recuérdese que cualquier actualización, sea una inserción, un borrado o una actualización, añade cero o más tuplas a *inserted* y *deleted*. Por lo tanto, se pueden usar estas dos tablas, independientemente de la actualización que se haya realizado, y hacer los ajustes necesarios en la tabla trabajador. Obsérvese que en

[2] Estos nombres de tabla son propios de SQL Server, por lo que se mantienen en inglés (N. del T.).

la siguiente solución no importa si se han añadido, borrado o cambiado una simple fila o múltiples filas a la vez. Esta solución funciona en ambos casos:

```
create trigger actualización_asignación
on asignacion
for insert, update, delete
as
update trabajador
set pago_acumulado = pago_acumulado + 8 *tarifa_hr *
        (select sum(num_dias) from inserted
        where inserted.id_trabajador = trabajador.id_trabajador )
update trabajador
set pago_acumulado = pago_acumulado - 8 * tarifa_hr *
        (select sum(num_dias) from deleted
        where deleted.id_trabajador = trabajador.id_trabajador)
```

Para entender la definición de este disparador se necesita mirar a sus partes separadamente. Comencemos con las primeras tres líneas:

```
create trigger actualización_asignacion
on asignacion
for insert, update, delete
```

La primera línea identifica el nombre del disparador como "actualización_asignación", mientras que la segunda línea indica que éste se aplica a la tabla asignación. La tercera línea establece que el disparador hará fuego en cada operación insert (insertar), update (actualizar) o delete (borrar).

La próxima línea del disparador, "as", introduce la parte código de la definición del disparador. Todo lo que venga después de esta línea se ejecuta cuando éste se dispare. Veamos ahora la última parte. Ésta consiste en dos instrucciones de actualización, ambas se aplican a la relación trabajador. La primera instrucción de actualización suma al pago acumulado usando la tabla *inserted*, la segunda instrucción resta del pago acumulado usando la tabla *deleted*.

```
update trabajador
set pago_acumulado = pago_acumulado + 8 *tarifa_hr *
        (select sum(num_dias) from inserted
        where inserted.id_trabajador = trabajador.id_trabajador )
update trabajador
set pago_acumulado = pago_acumulado - 8 * tarifa_hr *
        ( select sum(num_dias) from deleted
        where deleted.id_trabajador = trabajador.id_trabajador)
```

Estas dos instrucciones causan que el sistema pase a través de la relación trabajador dos veces. En la primera instrucción de actualización se trabaja con las tuplas que se han añadido a la relación asignación. Si se han añadido algunas, como habrá ocurrido si se han insertado o actualizado datos, entonces el atributo número de días (núm_días) en estas tuplas se usa para actualizar la tupla correspondiente en trabajador. Similarmente, la segunda instrucción mira las tuplas borradas de la relación asignación. Si ha habido borrados, como ocurriría si se han actualizado o borrado datos de la relación asignación, entonces se usa el número de días de estas tuplas para efectuar la resta correspondiente en la tupla apropiada de trabajador. Si la tabla *inserted* o la tabla *deleted* está vacía, entonces la instrucción de actualización que la usa no tendrá efecto en la relación trabajador. De esta manera el disparador trabajará de la forma deseada.

Usando disparadores en SQL Server y en Oracle. Los disparadores abren fuego siempre que el tipo de actualización que se especifique (insertar, borrar, actualizar) ocurran sobre la tabla. No importa qué usuario o qué programa hace la actualización. Si está definido un disparador para tal actualización, éste se disparará. Consecuentemente es importante reservar el uso de los disparadores para aquellas operaciones que *siempre* se deben realizar a continuación de una actualización específica. En los ejemplos que se han dado

previamente se pueden ver situaciones en las cuales los disparadores se han utilizado apropiadamente. Pero hay también muchas otras.

Un uso importante de los disparadores es para forzar negociaciones y otras reglas de integridad. Por ejemplo, SQL Server usa los disparadores para garantizar la integridad referencial y la unicidad de la clave primaria. Por otro lado, en Oracle, estos aspectos están construidos dentro del lenguaje de definición de datos. Esto es, en Oracle simplemente se declaran que ciertos atributos son claves primarias o externas y automáticamente se garantiza la unicidad y la integridad referencial. Pero en SQL Server se deben definir los disparadores si se quiere forzar el cumplimiento de estas restricciones. Por lo tanto, los disparadores son herramientas esenciales para la integridad de base de datos en SQL Server.

Sin embargo, de todos modos, Oracle requiere de disparadores por razones similares. Aunque lo relativo a las claves primarias y externas se garantiza automáticamente (cualquier regla de negociación que requiera referirse a otras tablas de la base de datos sólo podría garantizarse su cumplimiento a través de un disparador). Recuérdese que, como parte de la definición de esquema, Oracle permite la definición de restricciones CHECK para forzar el cumplimiento de las reglas sobre la base de datos. Sin embargo, recuérdese también que las restricciones CHECK no pueden incluir consultas que hagan referencia a otras tablas u otras tuplas en la misma tabla. Es decir, las restricciones CHECK sólo pueden ver una tupla a la vez. Por tanto, una regla como

"No puede planificarse a ningún trabajador por más de cien días para un trabajo de cada vez"

requiere de una consulta en la tabla asignación para calcular el número total de días (núm_días) que un trabajador dado tiene en asignación. No es posible declarar una restricción CHECK para forzar el cumplimiento de esta regla. Sin embargo, un disparador lo puede lograr muy atractivamente.

▼ Desarrollando aplicaciones cliente

La sección precedente describe la base de datos desde el lado del servidor de las aplicaciones cliente/servidor. En esta sección se verá un entorno de desarrollo de aplicaciones, PowerBuilder, que se utiliza para construir las partes cliente de las aplicaciones que harán de interfaces sobre las redes con los sistemas de bases de datos. PowerBuilder es un entorno gráfico de desarrollo de aplicaciones que opera sobre Windows. Puede usarse para crear programas de aplicación que hagan de interfaz con un gran número de diferentes sistemas de gestión de bases de datos (SGBD) relacionales comerciales. Para auxiliar en la creación de programas de aplicación PowerBuilder proporciona una variedad de *painters* (pintores)[3], que son subprogramas interactivos que realizan tipos específicos de funciones necesarias para el desarrollo completo de la aplicación. Tanto PowerBuilder como las aplicaciones desarrolladas con él funcionarán de forma autónoma (*stand-alone*), o en una plataforma cliente/servidor.

painter. Un subprograma interactivo que lleva a cabo un tipo específico de funcionalidad necesaria para el desarrollo completo de una aplicación.

El enfoque de PowerBuilder

Desde el punto de vista lógico, el desarrollo de aplicaciones en PowerBuilder es como sigue:

1. Primero se debe definir una base de datos. Esto implica dar el nombre de la base de datos, definir las tablas con sus columnas, los tipos de datos, las claves, las

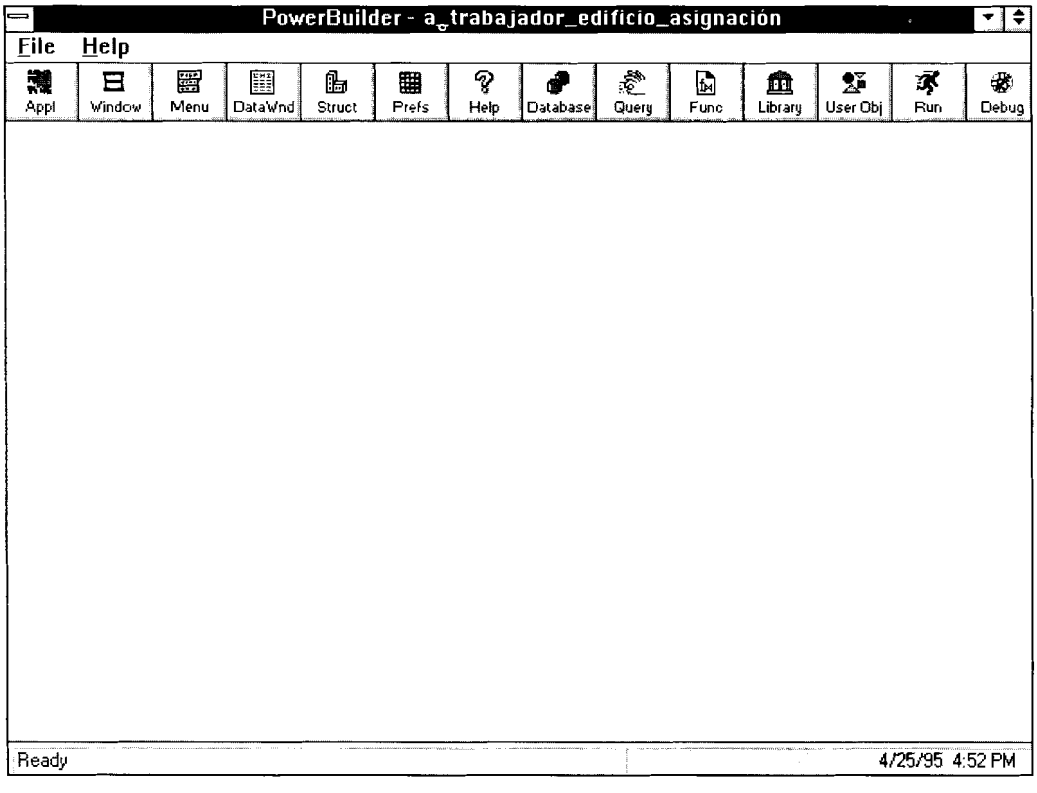
[3] Este y otros términos como *DataWindow*, *script*, etc., que son propios de PowerBuilder, se mantendrán en inglés (N. del T.).

claves externas, etc. Este paso ha sido tratado en la primera parte de este capítulo y se supone que ya ha sido entendido.

2. Después que se haya definido la base de datos, se puede comenzar con el desarrollo de las aplicaciones. Esto se hace con los *painters* de PowerBuilder. Normalmente, para cada aplicación se deben realizar tres pasos principales:
   a. Definir la aplicación. Esto se realiza con el *Application painter*.
   b. Definir las ventanas de aplicación. Esto se hace con el *Window painter* (pintor de ventanas). Windows tiene los medios por los cuales los usuarios pueden interactuar con la aplicación. Para darle a los usuarios las capacidades necesarias se definen ventanas con botones, cajas de listas (*list boxes*), cajas de datos de entrada (*data input boxes*) y menús. Varios *painters* se usan para realizar estas funciones.
   c. La aplicación actuará sobre los datos en la base de datos. Por lo tanto, debemos identificar los datos que se necesita introducir, los que se cambiarán, y los datos que deberán darse como salida. Esta identificación se hace a través de *DataWindows*, que se definen en el *DataWindows painter* antes de que se ponga en una ventana. Esto es, este paso debe hacerse antes de completar el paso anterior.

Usando PowerBuilder

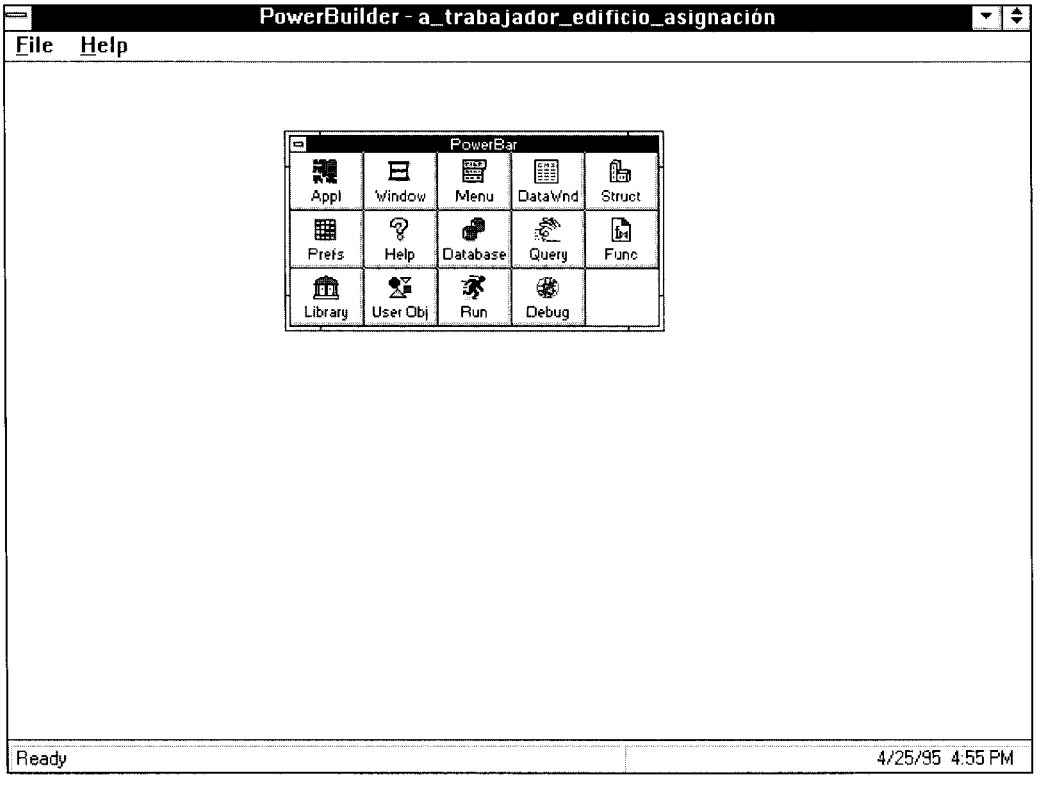
En la Figura 9.8 se muestra la pantalla principal del entorno de desarrollo de aplicaciones de PowerBuilder. Esta pantalla contiene una barra de título en la parte superior, con una barra de menú en la línea siguiente y la "PowerBar" en la tercera línea. La Power-



Bar (barra de Power) muestra los iconos de la mayoría de los variados *painters* que constituyen las capacidades funcionales de PowerBuilder. En la Figura 9.9 se muestran los iconos de los 14 *painters*.

Leyéndolos de izquierda a derecha, los botones de PowerBar representan:

1. **Appl.** Este es el *painter Application* (aplicación). Se usa para definir los aspectos generales de una aplicación, tales como el nombre de la aplicación, los programas que abren y cierran la aplicación (o "*scripts*"), que indican el procesamiento que debe tener lugar cuando se inicia o se termina la aplicación.
2. **Window.** Este *painter* se usa para construir ventanas para las aplicaciones.
3. **Menu.** Este *painter* se usa para construir menús para las ventanas.
4. **DataWnd.** Este *painter* construye objetos *DataWindow*, que se utilizan para acceder a la base de datos y poner los resultados en las ventanas. El *painter* Window (elemento 2) se utilizará para indicar qué DataWindow se usará en cada ventana.
5. **Struct.** Este *painter* se usa para definir las estructuras de datos que se usarán en los *scripts* (programas PowerBuilder). Las estructuras pueden concebirse como registros o grupos de campos construidos de modo que puedan referirse como un grupo.
6. **Prefs.** Según el *Help* de PowerBuilder: "En *Preferences* (preferencias) se pueden dar valores a todas las variables de PowerBuilder" o a un número de *painters*.
7. **Help.** El botón Help (ayuda) se usa para obtener información explicativa sobre las características de PowerBuilder.
8. **Database.** Este *painter* se usa para definir bases de datos.
9. **Query.** Este *painter* se usa para formular consultas SQL.



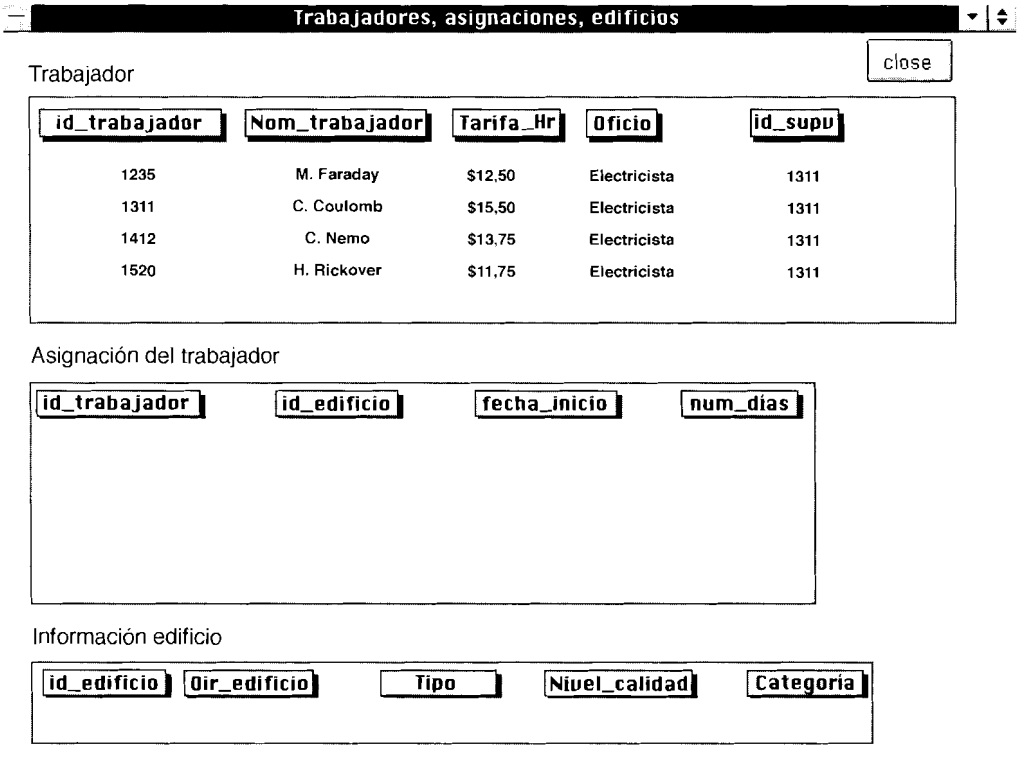
10. **Func.** Este *painter* se usa para desarrollar funciones definidas por el usuario, las cuales se pueden usar en los *scripts* de PowerBuilder.
11. **Library.** La bibiloteca controla la gestión de las aplicaciones, las ventanas y otros objetos creados usando PowerBuilder.
12. **User Obj.** Este *painter* se usa para crear objetos de usuario, "controles" (botones, listas, etc.) a la medida, los cuales se pueden poner en las ventanas.
13. **Run.** Presionar este botón provoca la ejecución de la aplicación en curso.
14. **Debug.** La facilidad de depuración (*debug*) brinda medios para la depuración de las aplicaciones.

El que desarrolla los sistemas introduce al *painter* que desea pulsando el botón adecuado. En tanto el sistema introduce el *painter*, se muestra otra barra de botones, el "PainterBar". A medida que avancemos se irán viendo ejemplos del PainterBar.

Como puede verse, PowerBuilder tiene un gran número de funciones y proporciona al que desarrolla la aplicación de una capacidad considerable. Nuestra discusión se limitará sólo a unos pocos de éstos principalmente, Applications, Windows y DataWindows (los elementos 1, 2 y 4).

Construir una aplicación

Habiéndose mostrado en la primera parte del capítulo cómo definir una base de datos en un servidor de SGBD, tal como Oracle o SQL Server, ya estamos preparados para construir una aplicación. La aplicación que se construirá se muestra en la Figura 9.10 y presentará las tuplas de la base de datos que se muestran en la Figura 9.11. Se construirán



TRABAJADOR

ID_TRABAJADOR	NOMB_TRABAJADOR	TARIFA_HR	OFICIO	ID_SUPV
1235	M Faraday	12,50	Electricista	1311
1412	C Nemo	13,75	Fontanero	1520
2920	R Garret	10,00	Albañil	2920
3231	P Mason	17,40	Carpintero	3231
1520	H Rickover	11,75	Fontanero	1520
1311	C Coulomb	15,50	Electricista	1311
3001	J Barrister	8,20	Carpintero	3231

ASIGNACIÓN

ID_TRABAJADOR	ID_EDIFICIO	FECHA_INICIO	NUM_DIAS
1235	312	10/10	5
1412	312	01/10	10
1235	515	17/10	22
2920	460	05/10	18
1412	460	08/12	18
2920	435	28/10	10
2920	210	10/11	15
3231	111	10/10	8
1412	435	15/10	15
1412	515	05/11	8
3231	312	24/10	20
1520	515	09/10	14
1311	435	08/10	12
1412	210	15/11	12
1412	111	01/12	4
3001	111	08/10	14
1311	460	23/10	24
1520	312	30/10	17
3001	210	27/10	14

EDIFICIO

ID_EDIFICIO	DIR_EDIFICIO	TIPO	NIVEL_CALIDAD	CATEGORIA
312	123 Elm	Oficina	2	2
435	456 Maple	Comercio	1	1
515	789 Oak	Residencia	3	1
210	1011 Birch	Oficina	3	1
111	1213 Aspen	Oficina	4	1
460	1415 Beech	Almacén	3	3

tres DataWindows, que se ubicarán en la ventana de la aplicación, tal como se muestra en la Figura 9.10. Una **DataWindow** es realmente una consulta SQL, que se puede poner en una ventana. Por tanto, antes de que podamos crear la ventana de la aplicación, debemos crear los DataWindows.

Después que hayamos completado nuestra aplicación, ésta trabajará como sigue. Cuando la aplicación comienza consultará la tabla trabajador e indicará todos los trabajadores en la ventana de datos (DataWindow) del tope. Luego, cuando el usuario seleccione pulsando la línea de algún trabajador, se rellenará la segunda ventana que contiene todos los edificios asignados a dicho trabajador. Por último, pulsar en cualquier tupla de asignación de la segunda ventana provocará que se muestre una tercera ventana que contiene la información completa sobre el edificio.

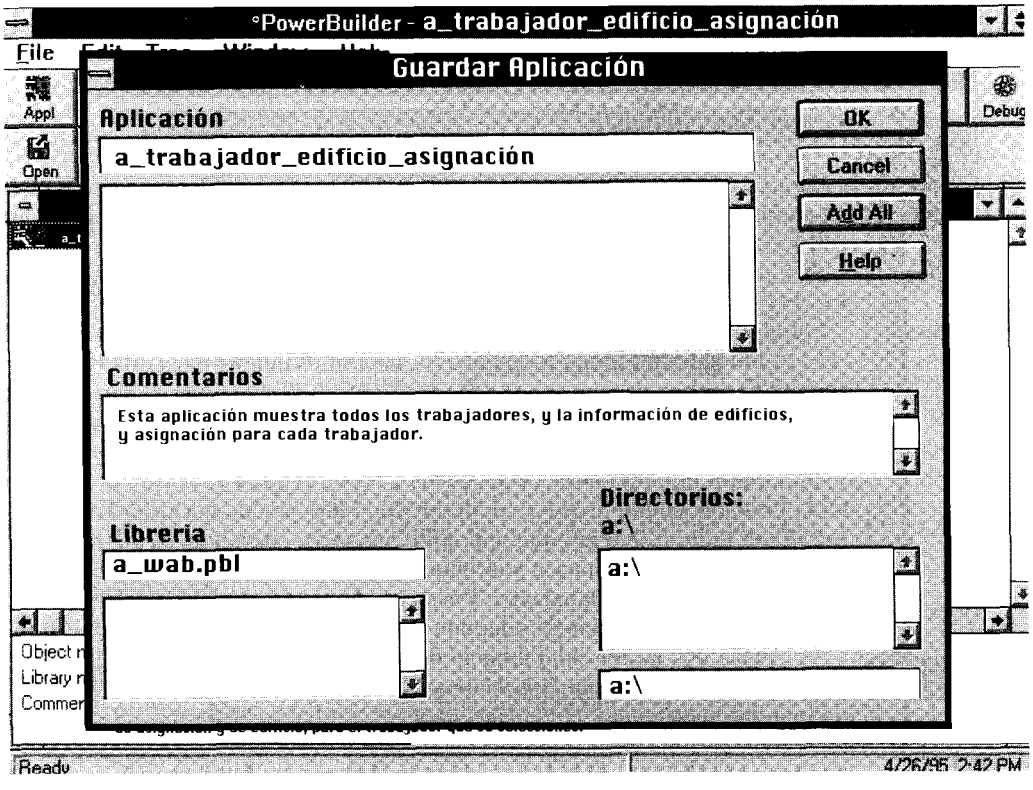
Para crear la información completa se debe proceder de la forma siguiente:

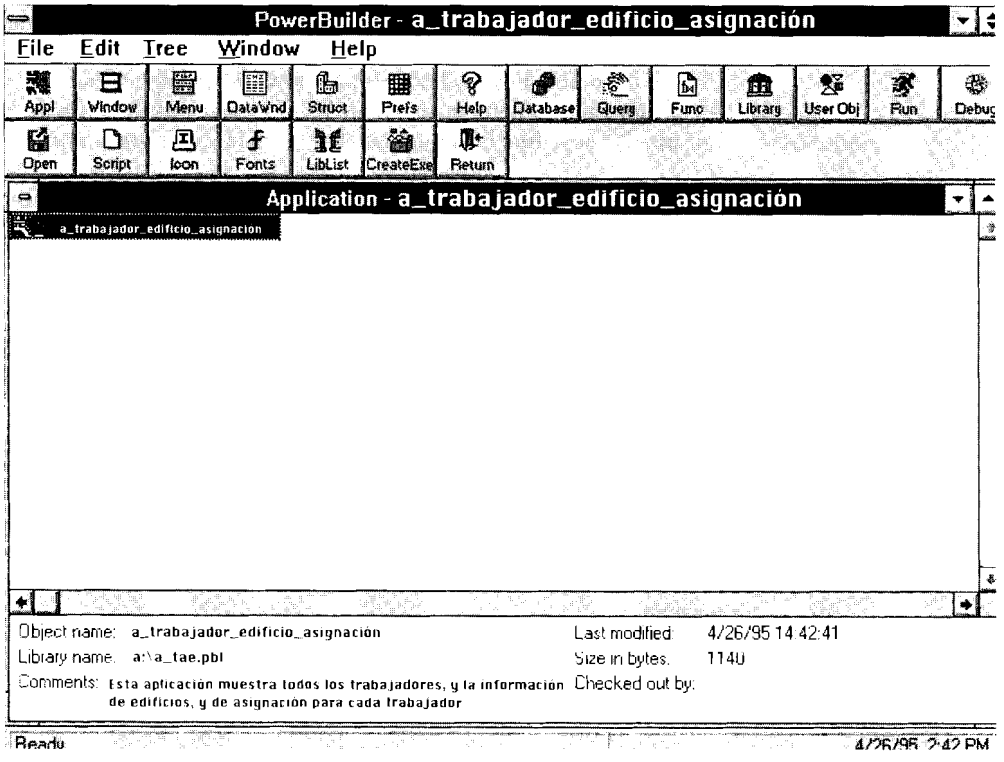
1. Primero se define la aplicación usando el *painter* Application.
2. A continuación se definen tres DataWindow mediante el *painter* DataWindow.
3. Por último, se define la ventana de la aplicación, usando el *painter* Window.

application painter. Un painter que se usa para definir los aspectos generales de una aplicación, tales como el nombre y la biblioteca de la aplicación.

De modo que comencemos por el paso 1. Si estamos usando el PowerBuilder, llamaríamos al **Application painter,** pulsando en el botón **Appl** de PowerBar. Después se obtiene la caja de guardar la aplicación (Save Application box) de la Figura 9.12. Le damos nombre a la aplicación tecleando "a_trabajador_edificio_asignación" en la primera caja. Esta aplicación desplegará los datos de las tablas trabajador, asignación y edificio, lo que se indica mediante estos nombres de tablas. Se les ha puesto el prefijo "a_" como convenio para indicar que es una aplicación para estas tablas. En la caja **Comments** (comentarios) introducimos "Esta aplicación despliega todos los trabajadores, y la información de asignación y de edificio, para el trabajador que se seleccione". En la caja **Libraries:** se pone el nombre del archivo a_tae.pbl para indicar que éste es el lugar donde queremos que se pongan todos los objetos que tienen que ver con esta aplicación. El sufijo ".pbl" viene de "PowerBuilder library" (Biblioteca de PowerBuilder) y es un sufijo estándar en PowerBuilder para estos fines. PowerBuilder automáticamente controla esta biblioteca y pondrá y recuperará de ella todos los objetos que hayamos creado o actualizado para usar en nuestra aplicación.

Ya se han definido los aspectos generales de la aplicación. Una descripción resumen de qué es lo que se ha hecho en la aplicación se muestra en la Figura 9.13. El próximo paso será crear las DataWindows que se usarán en la aplicación.





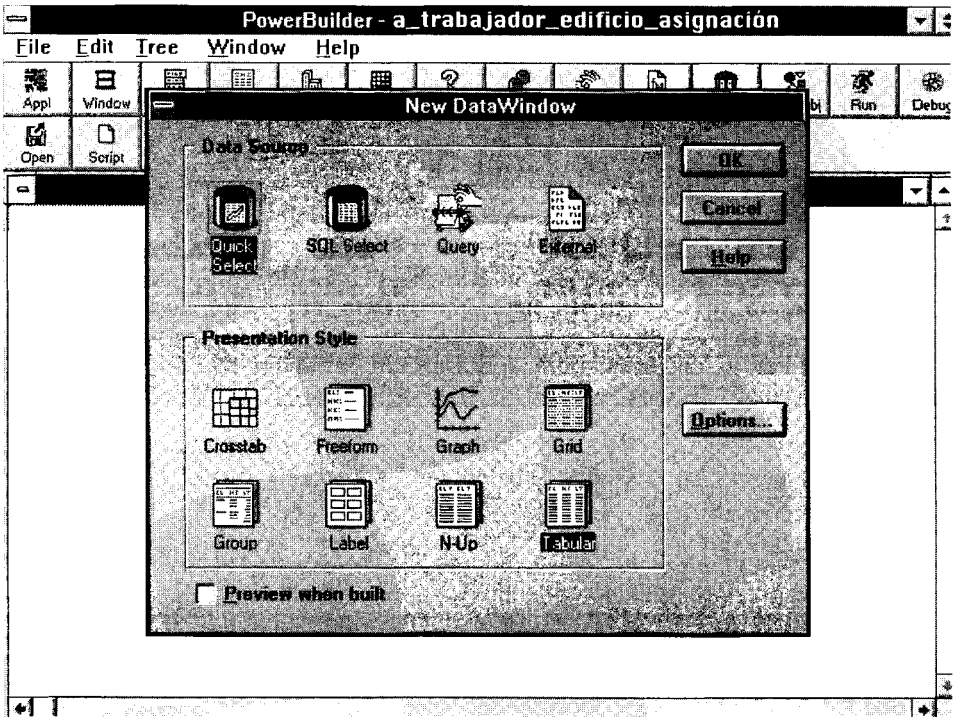
Crear DataWindows

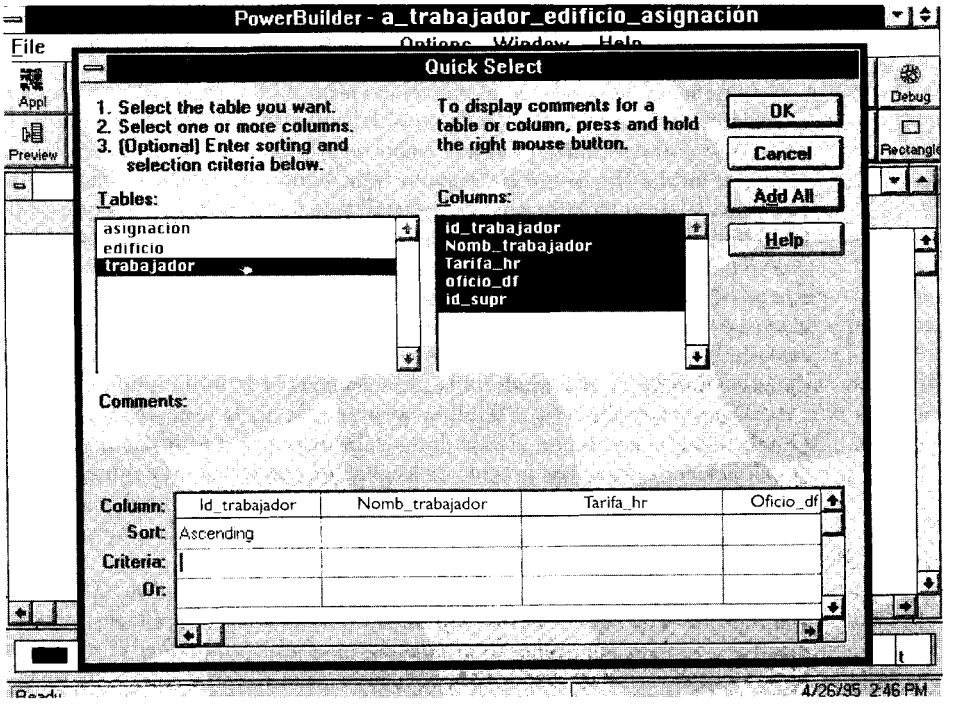
DataWindow painter. Un *painter* que construye objetos DataWindow que acceden a la base de datos y ponen los resultados en ventanas.

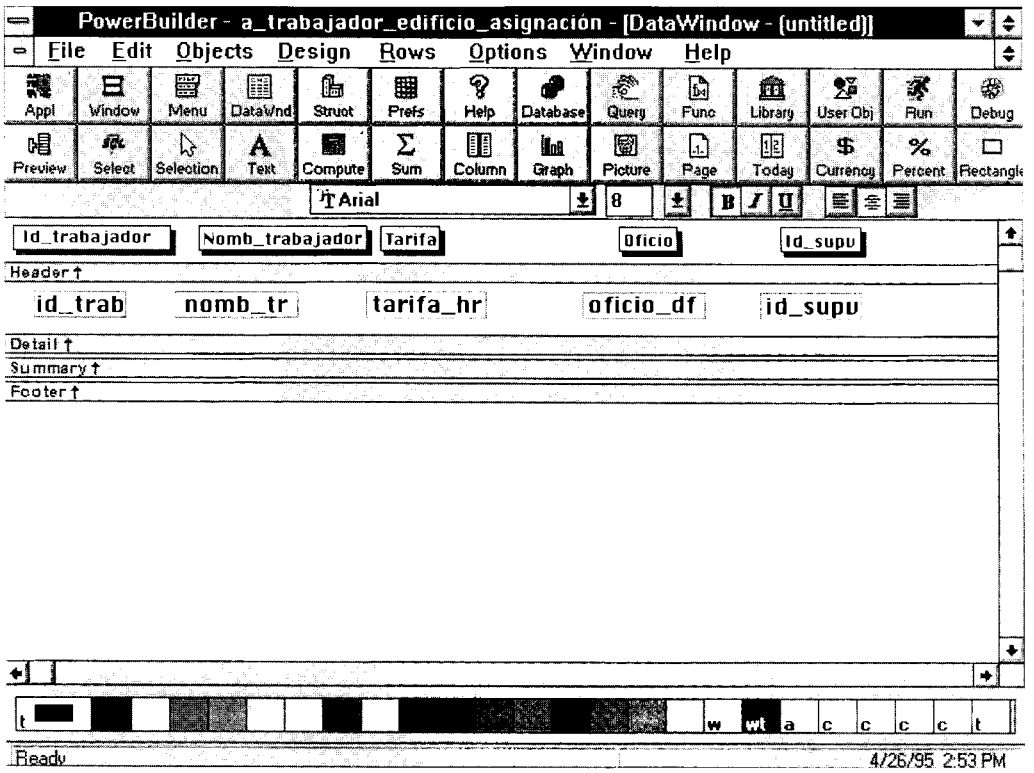
El próximo paso es introducir el **DataWindow** *painter* mediante el botón **DataWnd** de la PowerBar (ver Figura 9.9). Se crearán tres DataWindows: una para trabajador, otra para asignación y otra para edificio. Usaremos la caja New DataWindow (Figura 9.14). Como se indica en la caja, debemos designar el **Data Source** (fuente de datos) y el **Presentation Style** (estilo de presentación).

Las cuatro opciones de Data Source nos dan una potencia considerable para la creación de DataWindows. Sin embargo, las DataWindows para este ejemplo serán todas simples consultas de una sola tabla, por tanto podemos seleccionar la fuente de datos **Quick Select** (selección rápida) en cada caso. En la próxima caja se seleccionará el estilo de presentación **Tabular**. Esto significa que los datos se presentarán como tablas, en filas y columnas. Pasamos ahora a la caja Quick Select (Figura 9.15), donde se deben indicar la tabla y las columnas que queremos desplegar. Seleccionamos la tabla Trabajador y **Add All** (añadirlos todos), de modo que se desplieguen todas las columnas de trabajador. Luego se indica la orden de ordenación y el criterio de selección. Se indica Ascending (ascendiente) en la fila **Sort** bajo **Id Trabajador**. Puesto que queremos todas las tuplas de trabajador, no se indicará ningún criterio de selección. Con esto se completa la caja Quick Select.

Ahora nos movemos al espacio de trabajo del *painter* DataWindow (Figura 9.16), que contiene en la sección Detail (detalle) la información que hemos definido. Obsérvese que el espacio de trabajo tiene cuatro secciones: Header (cabecera), Detail (detalle), Summary (resumen) y Footer (pie). La sección Header contiene información textual que se añadirá a los encabezamientos de las columnas. La sección Detail contiene una caja para cada campo de datos con el nombre del campo dentro de la caja. Esto indica en qué lugar se desplegarán realmente los datos. Las secciones Summary y Footer están en este caso vacías. Se


PowerBuilder - a_trabajador_edificio_asignación
File Edit Tree Window Help
Appl
Window
Run
Debug
Open
Script
New DataWindow
Data Source
Quick Select
SQL Select
Query
External
OK
Cancel
Help
Presentation Style
Crosstab
Freeform
Graph
Grid
Group
Label
N-Up
Tabular
Options...
Preview when built


PowerBuilder - a_trabajador_edificio_asignación
File
Appl
Preview
Debug
Rectangle
Quick Select
1. Select the table you want.
2. Select one or more columns.
3. (Optional) Enter sorting and selection criteria below.
To display comments for a table or column, press and hold the right mouse button.
OK
Cancel
Add All
Help
Tables:
asignacion
edificio
trabajador
Columns:
id_trabajador
Nomb_trabajador
Tarifa_hr
oficio_df
id_supr
Comments:
Column:
Sort:
Criteria:
Or:
Id_trabajador
Nomb_trabajador
Tarifa_hr
Oficio_df
Ascending
4/26/95 2:46 PM

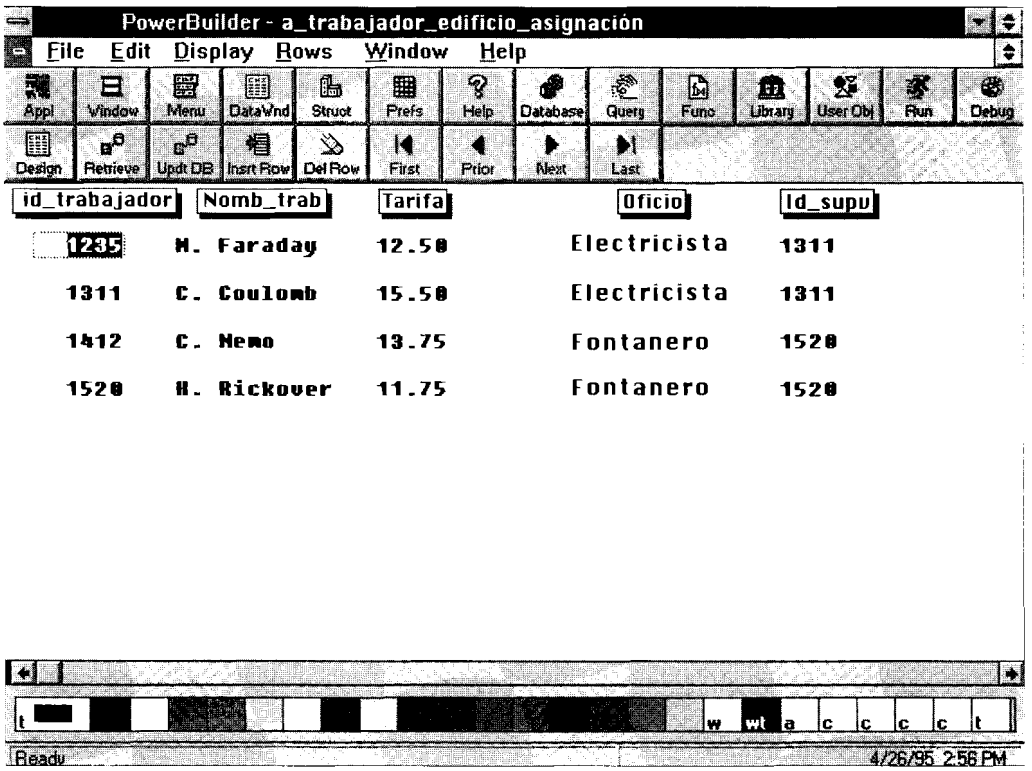


preview (*vista preliminar*). Un recurso que permite al que desarrolla la aplicación ver cómo aparecerá la DataWindow cuando se ponga en una ventana.

podría poner el total y otra información en la sección Summary y el número de página y otras informaciones textuales en la sección Footer. Podemos usar el recurso **Preview** (visión previa) de PowerBuilder para ver cómo será DataWindow (Figura 9.17). Observe que la información del Header se muestra en DataWindow tal y como vimos en la pantalla anterior. Sin embargo, la información en Detail lista los datos concretos que están contenidos en la tabla trabajador, a una línea para cada tupla. Podría usarse la barra de desplazamiento (*scroll bar*) que está a la derecha para desplegar todas las tuplas de la tabla. Si alguno de los campos de datos no es lo suficientemente grande o no se ha alineado adecuadamente, podemos regresar al espacio de trabajo de DataWindow para cambiarlo. Cuando se termine con esto, habremos completado la definición de DataWindow para la tabla trabajador. Le damos el nombre d_trabajador y la almacenamos en la biblioteca para esta aplicación.

Nuestra segunda DataWindow será para la tabla asignación . Se indicará de nuevo **Quick Select** como fuente de los datos y **Tabular** como estilo de presentación. En la caja Quick Select indicaremos la tabla de asignación con todas sus columnas y marcaremos a **Id Trabajador** e **Id Edificio** como **Ascending** (ascendente) forma de ordenación. Sin embargo, antes de finalizar con este Quick Select debemos hacer algo diferente de lo que hicimos con la primera DataWindow.

Recordemos cómo es que queremos usar esta DataWindow. A diferencia de la DataWindow anterior, que muestra una lista de *todos* los trabajadores, ahora sólo queremos mostrar cada vez las tuplas de asignación para un trabajador. El usuario verá la lista de los trabajadores en la primera DataWindow y podrá pulsar en la fila de un trabajador en particular. Queremos que entonces en la segunda DataWindow se desplieguen todas las tuplas de asignaciones para dicho trabajador. Por tanto, queremos que esta ventana recupere sólo las tuplas que se aplican al trabajador seleccionado por el usuario. Por lo tanto

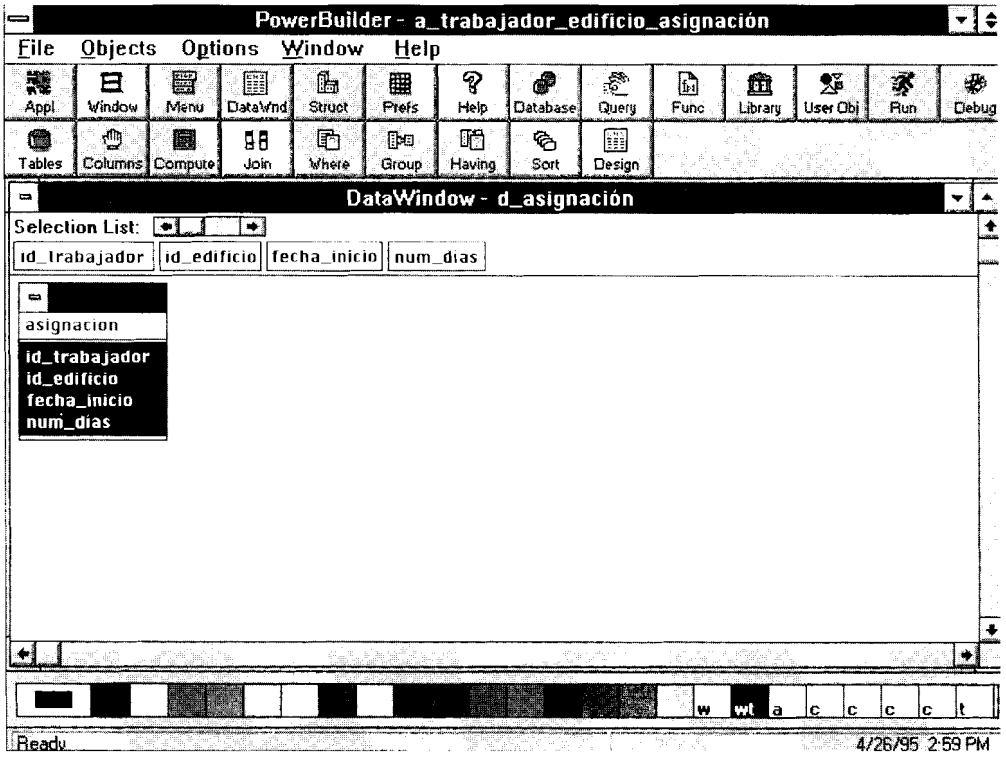


debemos modificar la consulta para la DataWindow de la tabla asignación poniéndole una cláusula Where que provoque que se muestren solamente las tuplas y la asignación del trabajador seleccionado. Esto se logra poniendo en la cláusula Where una variable que contendrá el Id_Trabajador como argumento. Este valor se pondrá en la variable en tiempo de ejecución cuando el usuario pulse sobre la tupla del trabajador. Para hacer la modificación necesaria en el DataWindow de la tabla asignación debemos acceder a una nueva ventana que muestre la tabla asignación con todas sus columnas y un nuevo conjunto de botones que nos permitan construir consultas más complejas (Figura 9.18).

Primero definimos la variable argumento de recuperación que se indicó anteriormente. Esta variable, nombrada ar_id_trabajador contendrá el Id_Trabajador para aquellos trabajadores de los que queremos recuperar sus tuplas en la tabla asignación. Se usan ahora las funciones de esta pantalla para construir una cláusula Where de la forma

```
WHERE id_trabajador = :ar_id_trabajador
```

(Los dos puntos en :ar_id_trabajador indican que ar_id_trabajador es una variable y no una columna de la base de datos.) Después de definir la cláusula Where retornamos al espacio de trabajo del *painter* DataWindow. Se definen ahora la fuente de datos y el estilo de presentación como antes y se escoge para desplegar a la tabla edificio con todas sus columnas. Una vez más se define un argumento de recuperación porque estamos interesados solamente en ver la información para el edificio en el que el usuario pulsó en la tupla de asignación. Esta vez definimos el argumento de recuperación como ar_id_edificio y construimos la cláusula Where siguiente:




```
WHERE id_edificio = :ar_id_edificio
```

Nombremos d_edificio a esta última DataWindow y la almacenamos en la biblioteca. Hemos ya terminado las tres definiciones de DataWindow y estamos listos para definir la ventana que las desplegará.

Crear Windows

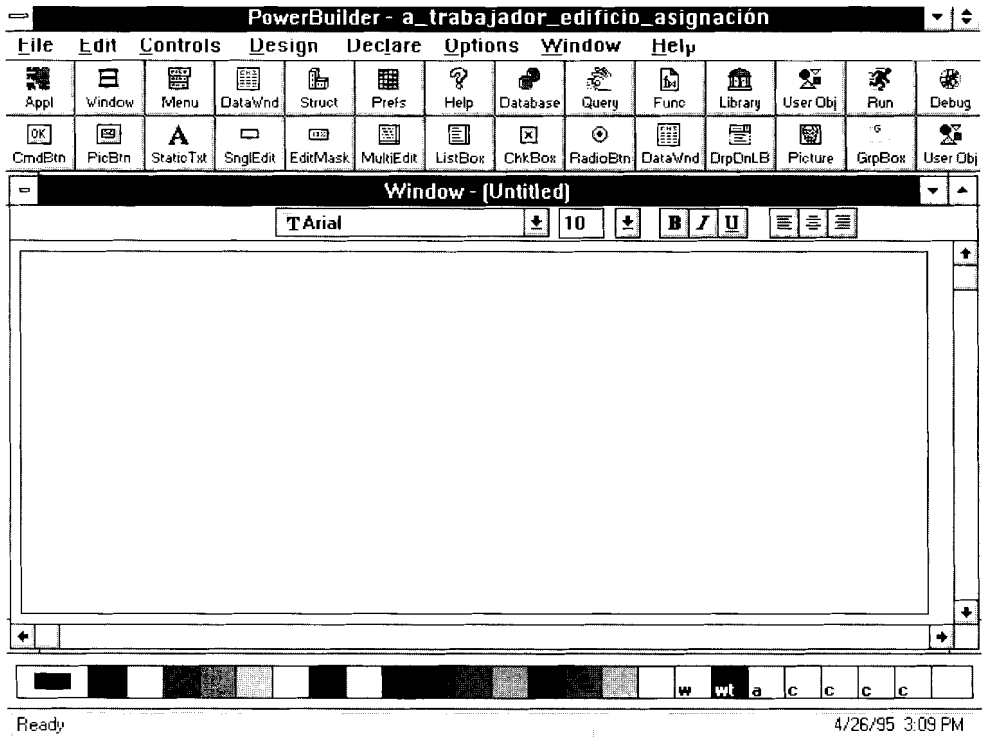
Hemos definido y nombrado a la aplicación (a_trabajador_asignación_edificio) y hemos definido tres objetos DataWindow (d_trabajador, d_asignación y d_edificio). Hemos usado como convenio para dar los prefijos "a_" para el nombre de una aplicación y "d_" para el nombre de un objeto DataWindow. A continuación definiremos una ventana (que es diferente a "DataWindow") para la aplicación. Esta ventana nombrada "v_tae" contiene los controles (definidos más tarde), incluidos los tres controles para los tres objetos DataWindow. Empecemos entonces.

window painter. Un *painter* que se usa para construir ventanas de una aplicación.

Se introduce el espacio de trabajo de **Window painter** que tiene la barra de título "Window - [Untitled]" (Figura 9.19). Observe la barra PainterBar que se muestra debajo de la PowerBar. Esta barra de botones se usa sólo en el Window painter y sus botones indican funciones del *painter*.

control. Un dispositivo gráfico por el cual el usuario interactúa con el sistema.

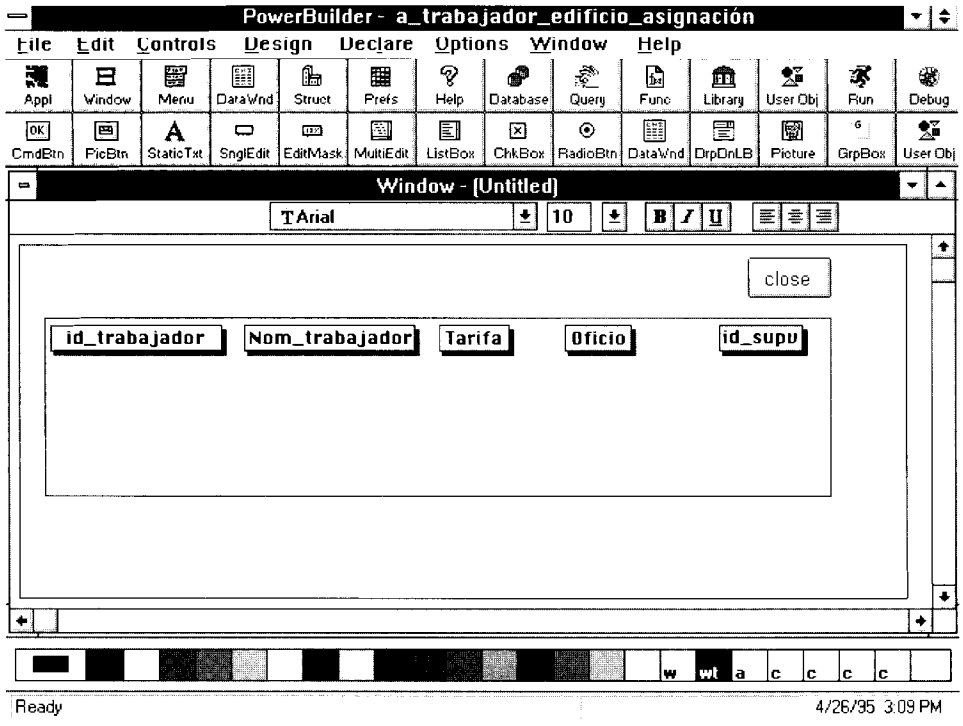
En la construcción de una ventana quisiéramos usar una variedad de estructuras llamadas controles. Un **control** es un dispositivo gráfico por el cual el usuario interactúa con el sistema. Hay una variedad de tipos de controles, incluidos los botones, cajas de listas, DataWindows, etc. En esta sección se ilustrará el proceso de poner controles en las ventanas usando un botón y tres DataWindows.

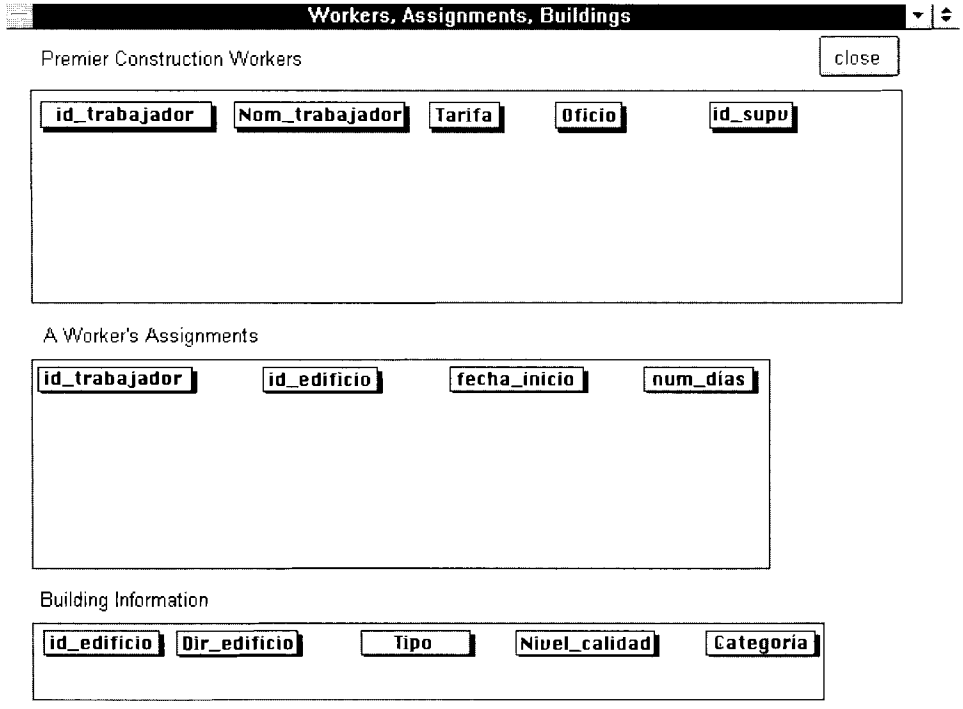


Añadamos primero el botón. La PainterBar (barra de *painters*) que se muestra en la Figura 9.19 muestra los botones de los diferentes tipos de control que se pueden poner en la ventana. Si se pulsa en el botón **CmdBtn** de la PainterBar, luego movemos el puntero hacia la esquina superior derecha de la caja y volvemos a pulsar, esto pondrá un botón con el texto "none" (nada) en la esquina superior derecha. Luego podemos teclear "cerrar" en el botón para sustituir a "none". (Ver Figura 9.20). El propósito de este botón es brindar un medio simple para que el usuario pueda abandonar la aplicación. Más tarde podemos escribir un *script* (guión) para este botón que cause que la aplicación termine cuando el usuario pulse en este botón.

Ahora añadiremos los controles DataWindow. Para definir el primer control DataWindow debemos pulsar en el botón **DataWnd** de PainterBar y luego pulsar en la esquina superior izquierda del espacio de trabajo. Usamos el ratón para darle tamaño a la ventana y luego indicar cuál de las ya definidas DataWindow debe colocarse aquí. Seleccionamos la d_trabajador e indicamos que también necesitaremos aquí una barra de desplazamiento vertical. Con esto habremos añadido el primer control DataWindow (Figura 9.20).

Los otros dos DataWindow se añaden de la misma manera. Hacemos esto poniendo el control d_asignación debajo del control d_trabajador y el control d_edificio debajo del control d_asignación. Dejamos espacio entre los controles de modo de poder añadir los textos de los encabezamientos. Añadimos la información textual encima de cada control DataWindow, pulsando en el botón **StaticTxt** de la PainterBar y luego moviendo el puntero hacia el lugar deseado y pulsando de nuevo. Tecleamos el texto deseado en la caja texto. Esto se hace tres veces, una por cada control DataWindow. El resultado final debe parecer algo así como lo que se muestra en la Figura 9.21. Finalmente guardamos la definición de esta ventana con el nombre v_tae.


PowerBuilder - a_trabajador_edificio_asignación
File Edit Controls Design Declare Options Window Help
Appl Window Menu DataWnd Struct Prefs Help Database Query Func Library User Obj Run Debug
CmdBtn PicBtn StaticTxt SnglEdit EditMask MultiEdit ListBox ChkBox RadioBtn DataWnd DrpDnLB Picture GrpBox User Obj
Window - (Untitled)
TArial
10
close
id_trabajador
Nom_trabajador
Tarifa
Oficio
id_supv
Ready
4/26/95 3:09 PM


Workers, Assignments, Buildings
Premier Construction Workers
close
id_trabajador
Nom_trabajador
Tarifa
Oficio
id_supv
A Worker's Assignments
id_trabajador
id_edificio
fecha_inicio
num_dias
Building Information
id_edificio
Dir_edificio
Tipo
Nivel_calidad
Categoria

Revisemos lo que hemos hecho hasta aquí. Se ha definido la base de datos de construcciones con tres tablas: trabajador, asignación y edificio. Estamos ahora en el proceso de creación de una aplicación que accederá a la base de datos, lista las tuplas de los trabajadores que hay en ella, y lista las tuplas de las asignaciones para el trabajador que se seleccione, así como lista también la tupla del edificio para una asignación. Para crear esta aplicación se ha utilizado el Application painter para definir ésta y darle nombre, luego usamos el DataWindow painter para definir los tres objetos DataWindow (d_trabajador, d_asignación y d_edificio) y finalmente usamos el Window painter para crear una única ventana que contenga los controles de los tres DataWindow y un botón de orden para cerrar la aplicación. Para las apariencias externas hemos hecho todo lo necesario por la aplicación.

Pero se ha olvidado algo. Tenemos todas las piezas necesarias: la base de datos, la aplicación, los DataWindows, y la ventana con los controles, pero no las hemos conectado del todo entre sí. Para hacer esto debemos escribir *scripts* (guiones) para los controles, así como también para la ventana y la aplicación como tales. Estos *scripts* contienen órdenes con los parámetros apropiados para estar seguros de que las cosas van a ocurrir correctamente y en el momento preciso.

Desgraciadamente, el lenguaje de *script* de PowerBuilder es bastante amplio y complicado y por lo tanto se sale del alcance de este libro. Sin embargo, antes de terminar con esta sección precisemos las tareas para las cuales se deben escribir *scripts*, de modo que la aplicación funcione adecuadamente. También describiremos la lógica que está detrás de la estructura de la aplicación y sus *scripts*. Aunque no se cubrirá la sintaxis específica del lenguaje de *script*, se podrá ver con más claridad la filosofía que éste encierra.

dirigida-por-eventos. Una característica de la aplicación que significa que se escriben *scripts* para responder a los eventos que tienen lugar.

Las aplicaciones de PowerBuilder son **dirigidas-por-eventos** (*event-driven*). Esto significa que los *scripts* se escriben para dar respuesta a los eventos que tendrán lugar. Ejemplos de eventos incluyen el comienzo de una aplicación, la apertura de una ventana, pulsar sobre un botón o sobre una línea en un DataWindow. Cada vez que ocurre un evento se ejecuta el *script* de tal evento. Por tanto, para escribir el *script* correcto para cada evento necesitamos conocer cuáles son los eventos que se suponen ocurrirán en la aplicación. Para hacer esto es necesario entender la estructura general de la aplicación y cómo encaja cada pieza en esta estructura.

Las aplicaciones están estructuradas jerárquicamente. El primer evento en una aplicación es la ejecución de la aplicación en sí misma. Cuando el usuario inicia la aplicación, se ejecuta el *script* para el evento "abrir" (*open*). Una de las principales responsabilidades de este *script* será abrir la ventana de la aplicación. Esto dispara un nuevo evento: el evento abrir (*open*) para la ventana inicial. Esto origina a su vez que se ejecute un nnevo script. Después que se despliega la ventana se presentan al usuario una serie de opciones (botones que puede pulsar), DataWindows en las que puede desplazarse o líneas en las que puede pulsar, etc. Estos son casos adicionales para los cuales se pueden escribir *scripts*. En conjunto, los *scripts* determinarán qué procesamientos tendrán lugar en respuesta a los eventos.

Veamos ahora el procesamiento y los eventos que lógicamente tienen lugar en nuestra aplicación y con esto iremos identificando qné *scripts* necesitamos escribir.

Evento 1: **Evento abrir aplicación.** Este evento activará un área de comunicación de transacciones, la cual es necesaria para identificar el SGBD con el que la aplicación va a comnnicarse y abrirá la ventana inicial (v_tae). Abrir la ventana inicial es en sí mismo un evento para el que hay que escribir nn *script*.

Evento 2: **Evento abrir ventana.** Cuando se abre la ventana inicial (v_tae) queremos que se despliegue la ventana y que la lista de trabajadores de la tabla trabajadores aparezca en DataWindow de la parte superior. Por lo tanto hay qne escribir un *script* para estar seguros de que esto ocnrrirá.

Evento 3: **Evento de pulsar sobre la línea de un trabajador.** El usuario, que está viendo la lista de trabajadores de la DataWindow superior, puede desplazarse a lo largo de esta lista para ver a todos los trabajadores. Si desea ver la lista de asignaciones de edificios

de un trabajador, lo único que tiene que hacer es pulsar sobre la línea de datos del trabajador. Esto dispara un evento para el cual debe escribirse su correspondiente *script*. Este *script* causará que las tuplas de las asignaciones de dicho trabajador sean desplegadas en la segunda DataWindow.

Evento 4: **Evento de pulsar sobre la línea de una asignación.** El usuario puede obtener información sobre el edificio de una determinada asignación si pulsa en la línea correspondiente a dicha asignación en la DataWindow. Esto dispara un evento para el que un *script* deberá recuperar la tupla del edificio indicado y mostrarla en la tercera DataWindow.

Evento 5: **Pulsar sobre el botón de la orden cerrar (*close*).** El usuario puede finalizar la aplicación simplemente pulsando en este botón.

Estos son todos los eventos de nuestra aplicación. El próximo paso sería desarrollar *scripts* para cada uno de ellos. Después que se haya hecho esto, la aplicación se ha terminado y podemos comprobarla ejecutándola. Inicialmente ésta mostrará toda la lista de trabajadores en la DataWindow superior con una barra vertical de desplazamiento a la derecha. Si se pulsa en la tupla de un trabajador, se indicarán todas las tuplas de las asignaciones correspondientes a dicho trabajador. Si pulsamos en la tupla de una asignación, se mostrará la información del edificio.

▼ Algunos comentarios finales

Se han cubierto con cierto detalle sistemas para el desarrollo de aplicaciones cliente (PowerBuilder) y de SGBD servidor (SQL Server y Oracle). Habrá observado que ambos tipos de sistemas incluyen lenguajes de programación que permiten la manipulación extensiva de los datos para cálculos, actualizaciones y presentaciones. Una de las ventajas del concepto de sistemas abiertos que se mencionó en el Capítulo 1 es que podemos tener "lo mejor de ambos mundos", brindando una buena interfaz entre los dos mundos existentes.

La experiencia de este ejemplo muestra que hay ciertas operaciones que mejor se llevan a cabo al nivel de la base de datos y hay ciertas que mejor se hacen a nivel del cliente. En particular, si encontramos que uno de los sistemas (digamos el sistema cliente) adolece de cierta capacidad, podemos pasar al sistema de gestión de base de datos y desarrollar esa capacidad allí. Por ejemplo, puede resultar difícil manipular conjuntos de tuplas al nivel del cliente, pero el SGBD está optimizado para brindar precisamente tales manipulaciones. De este modo, el desarrollador del sistema tiene múltiples herramientas disponibles para la solución de problemas. Si un problema no puede manejarse fácilmente por el lenguaje del SGBD se puede hacer por el lenguaje orientado al cliente, y viceversa. Teniendo múltiples herramientas disponibles tenemos más medios a nuestro alcance para elaborar poderosos sistemas.

Una ventaja adicional del enfoque cliente/servidor y del software para el desarrollo de sistemas que se encuentra a nivel del cliente o del servidor es que la separación del hardware fuerza que se cumpla el principio de modularización. Por tanto, no tenemos otra opción que dividir algunas de las funciones en piezas separadas física y lógicamente. Las interfaces entre estas piezas deben estar claramente definidas e implementadas con precisión, de modo que todas las ventajas de la modularización que han sido tradicionalmente señaladas deben ser aplicadas por el desarrollador del sistema.

Esperamos que este capítulo le haya transmitido al lector el sentido del significado potencial de los sistemas cliente/servidor para expandir el poder de los sistemas de aplicación en los negocios. Una revisión de la literatura actual nos deja muy poca duda sobre que los sistemas cliente/servidor han llegado para quedarse y que tendrán una importancia creciente en toda nuestra vida profesional durante un buen tiempo.

En este capítulo nos hemos extendido en las ideas asociadas con los sistemas de bases de datos cliente/servidor que fueron introducidas en los primeros capítulos. Se explicó que una plataforma cliente/servidor es una red de computadores, algunos de las cuales actúan como servidores brindando servicios de varios tipos a otros computadores clientes. Separando estas funciones se pueden especializar los sistemas de hardware, optimizando las máquinas clientes y las máquinas servidor, de modo que cada una pueda hacer mejor la tarea que tiene asignada. Vimos sistemas de software, tanto para los computadores servidores como para las clientes. Se examinaron los SGBDs SQL Server y Oracle en algún detalle, se vieron sus recursos para la definición y la manipulación de datos. Después se revisó PowerBuilder, un entorno para el desarrollo de aplicaciones en los computadores clientes.

Los lenguajes de definición de datos de SQL Server y Oracle permiten definir tablas de bases de datos mediante la creación de tipos de datos definidos por el usuario, tablas, columnas y restricciones sobre todos éstos. Los tipos de datos definidos por el usuario se desarrollan a partir de tipos básicos suministrados por el sistema y añadiendo ciertas restricciones sobre éstos. Estos tipos de datos pueden luego usarse en la definición de columnas específicas dentro de las tablas. Al definir las tablas se definen sus columnas con sus tipos de datos y sus restricciones. Las restricciones pueden ser reglas que restringen los valores que pueden estar en las columnas, éstas pueden indicar si se permiten valores nulos y pueden también garantizar la unicidad de los valores de una columna dentro de la tabla. Ellas también definen claves primarias o externas de una columna o múltiple-columna. Por último, las restricciones pueden ser más generales usando el concepto de restricción CHECK, que indica una regla que debe cumplir cada tupla.

Los lenguajes de manipulación de datos de estos dos SGBDs son muy poderosos, permiten bloques de instrucciones, instrucciones IF, lazos WHILE, variables locales y procedimientos almacenados con parámetros de entrada y de salida. La condición que se usa en las instrucciones IF o WHILE, a diferencia de las típicas condiciones en los lenguajes de programación, pueden usar valores computados a partir de relaciones completas en lugar de simples tuplas de una relación. Estos sistemas también soportan disparadores, que son programas escritos en el lenguaje de manipulación de datos y que se ejecutan cada vez que ocurre un tipo específico de cambio (inserción, borrado o actualización) en una relación específica. Éstos permiten el cumplimiento centralizado de un amplio rango de tipos de reglas de negocios y el cálculo de importantes valores derivados.

El desarrollo de las aplicaciones de los sistemas clientes pueden tener lugar en un entorno que brinde capacidades bien establecidas en las interfaces gráficas de usuario. Usando PowerBuilder como ejemplo estudiamos la definición de aplicaciones, DataWindows que definen consultas y la presentación de los resultados de las consultas. Se estudiaron las ventanas en las que se ponen las DataWindows y otros controles orientados al usuario. También se describieron los tipos de eventos que puede provocar el usuario en una ventana y para los cuales queremos escribir programas *scripts* que respondan a dichos eventos.

Por último, se ha señalado que los entornos de desarrollo de aplicaciones cliente/servidor nos brindan la oportunidad de usar múltiples herramientas para la creación de sistemas modulares de considerable poder y flexibilidad.

1. Defina con sus propias palabras cada uno de los términos siguientes:
   a. Interfaz Gráfica de Usuario (IGU, en inglés GUI)
   b. regla
   c. valor por defecto
   d. restricción de columna

e. restricción CHECK
f. lenguaje de flujo de control
g. procedimiento almacenado
h. bloque de instrucciones
i. BREAK
j. variable local
k. valor por defecto de un parámetro
l. tablas de comprobación de disparadores
m. *DataWindow*
n. *Application painter*
o. *Window painter*
p. control

2. Describa cómo se pueden usar los tipos de datos definidos por el usuario en la definición de las tablas de una base de datos. ¿Por qué son valiosos?

3. Describa las interrelaciones entre una restricción de valor nulo y un valor por defecto. Si se define un valor por defecto, ¿podría definirse una restricción de valor nulo? ¿Por qué?

4. ¿Cuál es la diferencia entre una regla y una restricción CHECK?

5. ¿Cuál es la diferencia entre una restricción CHECK y un disparador?

6. ¿Cuál es la diferencia entre restricciones de columna y restricciones de tabla?

7. ¿Qué pasa cuando está activado un ON DELETE RESTRICT? ¿Qué pasa cuando está un ON DELETE CASCADE?

8. ¿Cuál es la diferencia entre una restricción UNIQUE, la restricción PRIMARY KEY y la restricción NOT NULL?

9. Indique las diferentes construcciones del lenguaje de flujo de control.

10. ¿Qué es un bloque de instrucciones y cómo se especifica?

11. ¿En qué forma difieren fundamentalmente las instrucciones IF y WHILE del lenguaje de flujo de control de las instrucciones respectivas en un lenguaje convencional?

12. ¿Cómo se usan el BREAK y el CONTINUE?

13. ¿Mediante qué instrucción se definen las variables locales en el lenguaje de flujo de control?

14. ¿Qué aspectos de los procedimientos almacenados hacen posible que éstos puedan usarse en la creación de módulos de programas que se puedan comunicar entre sí?

15. Analice las ventajas de los procedimientos almacenados.

16. ¿Qué pueden hacer los disparadores que no pueden hacer las restricciones?

17. ¿Qué funciones llevan a cabo los *painters* en PowerBuilder?

18. Indique los *painters* de PowerBuilder.

19. ¿Cuál es la diferencia entre una ventana y un DataWindow?

20. Analice el procedimiento mediante el cual se definen las consultas en un DataWindow.

21. Analice el procedimiento mediante el cual se ponen los controles en las ventanas.

22. ¿Qué ejemplos de eventos que pueden ocurrir en una ventana han sido analizados en este capítulo?

Para todos los problemas, a continuación desarrolle las respuestas usando las facilidades de SQL Server, Oracle o PowerBuilder que se discutieron en este capítulo.

1. Haga corresponder cada término con su definición.

—*plataforma cliente/servidor*	a. Lleva a cabo el tipo de funcionalidad que se necesita para el desarrollo completo de la aplicación.
—*tipo de datos definido por el usuario*	b. Lenguaje de manipulación de datos de Oracle.
—*restricción*	c. Garantiza que dos tuplas no tengan idénticos valores en una columna.
—*restricción de tabla*	d. Significa que se escriban *scripts* para responder a las ocurrencias.
—*restricción UNIQUE*	e. Red local de computadoras que dan y reciben servicios.
—*PL/SQL*	f. Causa que el control de la ejecución retorne a la primera instrucción en un lazo WHILE.
—*instrucción IF*	g. Subtipo a la medida del usuario de un tipo de datos suministrado por el sistema.
—*instrucción WHILE*	h. Restricción sobre los posibles valores de tuplas y atributos.
—*CONTINUE*	i. Construye objetos que acceden a la base de datos y pone los resultados en ventanas.
—*parámetro*	j. Se usan para pasar datos hacia y desde procedimientos almacenados.
—*disparador*	k. Instrucción cuya ejecución depende del valor verdadero de una condición.
—*painter*	l. Se aplica a múltiples columnas simultáneamente.
—*DataWindow painter*	m. Instrucción que usa una condición para controlar una ejecución iterativa.
—*Preview (prevista)*	n. Se ejecuta cuando se intenta hacer una actualización específica sobre una tabla específica.
_*dirigida por evento*	o. Permite al desarrollador ver cómo aparecerá una DataWindow cuando se ponga en una ventana.

2. Identifique los valores por defecto apropiados en la base de datos de una escuela para las situaciones siguientes:
   a. Un nuevo año escolar.
   b. Un nuevo rango de profesor.
   c. Un nuevo año esperado de graduación.

3. Defina un valor por defecto 3231 para usar con la columna id_supv en la tabla trabajador.

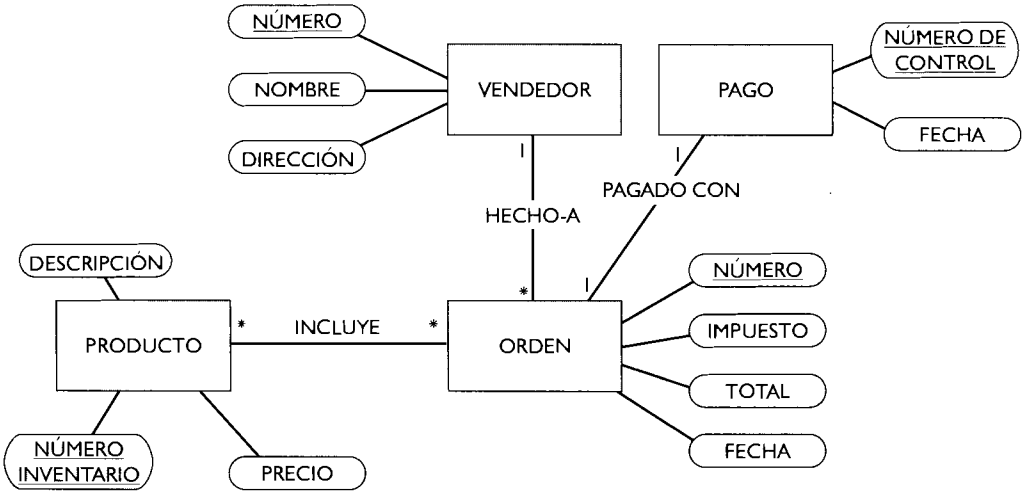
4. Especifique un nombre, un tipo, una longitud, un valor por defecto y una regla según sea necesario en los siguientes tipos de datos definidos por el usuario:
   a. Un tipo de datos para definir edad, entre 0 y 80, y que sea 18 si no se introduce ningún valor.
   b. Un tipo de datos para definir salario por hora, entre $5.00 y $20.00, que sea $5.00 si no se introduce ningún salario.
   c. Un tipo de datos para definir el género, que debe ser masculino o femenino.
   d. Un tipo de datos para definir el estado marital, que debe ser soltero o casado, y que sea soltero cuando no se introduzca ningún valor.

5. Defina las siguientes tablas en las que la información básica de las columnas es:

```
edificio
 id_edificio       tipo_id
 dir_edificio      character 12     nulo
 tipo              characetr 9      nulo, no por defecto, valor debe ser
                                    oficina, comercio, residencia o
                                    almacén.
 nivel_calidad     integer          nulo, defecto = 1, el valor debe
                                    estar entre 1 y 4 (inclusive)
 categoría         integer          nulo, defecto = 1, el valor debe
                                    estar entre 1 y 3 (inclusive)

asignación
 id_trabajador     tipo_id
 id_edificio       tipo_id
 fecha_inicio      datetime         nulo
 núm_días          integer          nulo, el valor debe ser mayor que 0
```

6. Defina un esquema de base de datos, incluyendo los nombres de tablas, columnas y claves primarias y externas para el esquema de la Figura 9.1E.



7. Defina las restricciones CHECK para cada uno de los casos siguientes:
   a. En la relacional edificio, si el tipo es Oficina, entonces el nivel de calidad debe estar por encima de 2.

b. En la relación trabajador, si el tipo es fontanero, la tarifa por hora debe estar por encima de 10,00.
c. En la relación trabajador, si el id_supv no es nulo, la tarifa por hora debe estar por debajo de 20,00.

8. Crear las instrucciones de flujo de control para cada uno de los casos siguientes:
   a. Si la categoría media de los edificios está sobre 1, incrementar en 1.00/hr la tarifa salarial por hora de cada uno de los trabajadores que sean carpinteros.
   b. Si al supervisor 3335 se le asigna a trabajar por un total de más de treinta días en varios edificios, decrezca repetidamente en 1 el número de días asignados a cada edificio hasta que esté trabajando por debajo de veinticinco días en total.

9. Crear un procedimiento almacenado que tome el tipo de oficio como parámetro y calcule y devuelva el número medio de días en las tuplas de asignación de todos los trabajadores con dicho oficio.

10. Suponga que hay una columna en la relación edificio, núm_tot_días, que contiene el número total de días que son asignados los trabajadores a dicho edificio. Cree un disparador que actualice esta columna cada vez que haya una actualización en la tabla asignación.

11. Describa las ventanas, DataWindows y otros controles que se usarían para crear las siguientes aplicaciones en PowerBuilder:
    a. Una aplicación que permita al usuario ver y actualizar las tuplas de la relación trabajador por su oficio. El usuario escogería un oficio y el sistema le mostraría todos los trabajadores que tienen dicho oficio. El usuario podría entonces cambiar los valores de los datos de la tuplas de los trabajadores seleccionados e indicar que se debe actualizar la base de datos.
    b. Esta aplicación es la misma que la anterior, con la excepción que se aplica a edificios. El usuario selecciona los edificios según su tipo.

1. Obtenga copias de trabajo de varios SGBDs servidores (tales como SQL Server y Oracle) y examine sus características de definición y de manipulación de datos. Específicamente, identifique cuáles SGBDs permiten definición de tipos de datos, reglas, valores por defecto, claves primarias y claves externas, con cumplimiento automático de estas dos últimas. Identifique cuáles SGBDs soportan instrucciones IF y WHILE, así como otros recursos de lenguajes de control de flujo discutidos en este capítulo, con condiciones que se apliquen a las relaciones completas de cada vez en lugar de a tuplas simples. Escriba un informe comparando estos SGBDs.

2. Obtenga copias de trabajo de diferentes paquetes de entornos de desarrollo de aplicaciones cliente (tales como PowerBuilder o SQLWindows) y examine sus capacidades con respecto a las ventanas (*window*), DataWindow y otros controles, así como también a sus lenguajes de manipulación de datos. Escriba un informe comparando estas facilidades.

CAPÍTULO

10

ORGANIZACIÓN FÍSICA DE LOS SISTEMAS DE BASE DE DATOS

Introducción
Acceso físico a la base de datos
Formas de almacenamiento físico
- Almacenamiento secundario
- Bloques de almacenamiento físico

Factores de rendimiento del disco
- Tiempo de posicionamiento
- Tiempo de activación de la cabeza
- Retraso de rotación
- Velocidad de transferencia de datos
- Tiempo de transferencia de datos
  - Ejemplo de un registro accedido aleatoriamente
  - Ejemplo de un registro accedido secuencialmente

Formatos de almacenamiento de datos en disco
- Formatos de pistas
- Formatos de registros
  - Registros de longitud fija
  - Registros de longitud variable
- Gestión de entrada/salida

Organización de archivos y métodos de direccionamiento
- Organización secuencial de un archivo
- Organización secuencial-indexada de un archivo
- Organización directa de un archivo
- Funciones *Hash* estáticas
- Funciones *Hash* dinámicas

Implementación de interrelaciones lógicas
- Listas enlazadas
- Listas invertidas
- Árbol equilibrado indexado (B+ -Árbol)

Implementación cliente-servidor de la indexación de un árbol balanceado
Correspondencia entre estructuras de datos lógicas y estructuras de datos físicas
- Correspondencia en las bases de datos relacionales
- Correspondencia en las bases de datos en red
- Correspondencia en las bases de datos jerárquicas

Acceso a la clave secundaria
Optimización de las consultas
- Combinar las operaciones de selección y reunión
- Combinar las operaciones de proyectar, selección y reunión

Resumen
Preguntas de comprobación
Ejercicios y problemas
Proyectos y cuestiones profesionales

Billie Tollefson, la actual administradora de los Servicios de Computación en la Corporación Zeus, encontraba inquietante la actitud de uno de sus colegas. Archie Krepsbach, que está a cargo de la Planificación de Requisitos de Materiales (PRM) en la División de Manufactura; se había quejado al Vicepresidente de Manufactura sobre la labor cuestionable de los Servicios de Computación para satisfacer las necesidades de información de PRM. "Aquí tenemos un nuevo sistema de base de datos, que se dice que nos lleva a la vanguardia en la gestión de la información, pero no siempre puedo obtener la información que necesito cuando la necesito. Me siento frustrado con los tiempos de respuesta que obtengo de nuestro SGBD."

Después de oír esto de su jefe, Billie respondió: "Puedo entender la frustración de Archie con nuestro tiempo de respuesta, pero lo que parerece no entender es que los tipos de informes que necesita son particularmente difíciles de producir rápidamente. Si conociera mejor las complejidades de la organización física de las bases de datos, encontraría más fácil ser paciente con nuestro tiempo de respuesta. Comprendería la necesidad de identificar informes que requieren respuestas rápidas y aquellos que pueden esperar un poco más. Entonces podríamos perfeccionar la base de datos para dar una mejor respuesta cuando se trate de informaciones de alta prioridad." Después de considerar el asunto por uno o dos días, Billie decidió recomendarle a su jefe la oferta de seminarios internos para familiarizar a los usuarios con los rudimentos del diseño físico y operaciones con las bases de datos.

Este capítulo trata acerca de las estructuras físicas que se usan para implementar bases de datos. Después de leer este capítulo será capaz de:

- Describir las estructuras para el acceso físico de la base de datos.
- Explicar las características de almacenamiento, recuperación y ejecución en disco.
- Entender los tipos básicos de organización de un archivo y cómo funcionan.
- Describir el uso de los punteros para crear listas enlazadas, listas invertidas y B^+-árbol.
- Entender cómo los modelos de datos lógicos se corresponden con estructuras de datos físicas.
- Definir y entender claves secundarias.
- Entender los principios básicos de la optimización de consultas.

▼ Introducción

La organización física de una base de datos es un tópico extenso. Sus detalles principalmente son del interés de los especialistas técnicos involucrados en el diseño de hardware y sistemas de software. Sin embargo, el rendimiento general de un sistema de base de datos se determina en gran medida por las estructuras de datos físicas usadas y por la eficiencia con la cual el sistema trabaja sobre las mismas. Aunque los usuarios no deben tener conocimiento de los detalles del diseño físico de la base de datos, éstos afectan al rendimiento, un factor de gran importancia en la satisfacción del usuario con el sistema de base de datos. ¿Podría el usuario obtener la información deseada en el formato apropiado y en un tiempo conveniente? Esta última frase, "tiempo conveniente", puede expresarse generalmente como tiempo de respuesta aceptable. La "información deseada" y el "formato apropiado" no se afectan mucho por la organización física de la base de datos, pero el tiempo de respuesta sí. El tiempo de respuesta es el tiempo transcurrido entre la inicialización de una operación sobre la base de datos y la disponibilidad del resultado. Un tiempo de respuesta lento es la queja más frecuente que expresan los usuarios de los sistemas de bases de datos, posiblemente debido a que es lo qne se observa más fácilmente.

Un buen diseño físico de la base de datos almacenaría datos de forma que puedan recuperarse, actualizarse y manipularse en el mínimo tiempo posible. En este capítulo nos interesaremos en aspectos de la organización física de la base de datos que soportan la eficiencia de las operaciones en las mismas. Aunque como analista de sistemas de negocios, no esté envuelto directamente en los detalles del diseño físico y de la implementación, le es importante entender los problemas que se plantean y sus soluciones típicas, ya que podrían afectar sus sistemas a nivel del usuario.

▼ Acceso físico a la base de datos

En la Figura 10.1 se muestra el sistema para el acceso físico a la base de datos. Se puede ver la interacción del usuario con el sistema de base de datos al iniciar una consulta. El **selector de estrategia** traduce la orden del usuario a su forma más eficiente para su ejecución. La orden traducida activa entonces al **administrador de buffer,** que controla el movimiento de datos entre la memoria principal y el almacenamiento en disco. El **administrador de archivos** da soporte al administrador de buffer administrando la reserva de localizaciones de almacenamiento en disco y las estructuras de datos asociadas. Además de los datos del usuario, el disco contiene el **diccionario de datos**, que define la estructura de los datos del usuario y cómo éstos pueden usarse.

selector de estrategia. Software que transforma una consulta del usuario en una forma efectiva para la ejecución.

gestor de buffer's. Software que controla el movimiento de datos entre la memoria principal y el almacenamiento en disco.

administrador de archivos. Software que administra la reservación de localizaciones de almacenamiento y estructuras de datos.

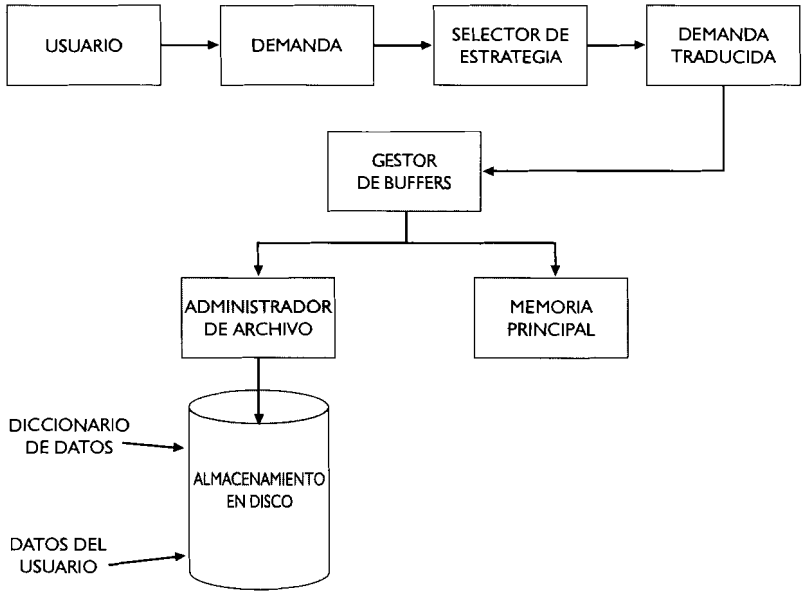
diccionario de datos. Parte del SGDB que define la estructura de los datos del usuario y cómo éstos son usados.

Los datos del usuario se almacenan como una base de datos física o colección de registros. Por ejemplo, una fila en una relación puede almacenarse como un registro físico, donde cada valor de los atributos de la fila se almacena en su propio campo. Análogamente, un registro lógico de un modelo en red o jerárquico puede almacenarse como un registro físico donde los elementos de datos lógicos se convierten en elementos de datos físicos del registro físico almacenado.

▼ Formas de almacenamiento físico

La **memoria principal** es el almacenamiento intermedio usado por los datos que están disponibles para las operaciones del usuario. Aquí es donde reside la ejecución del programa.

memoria principal. Almacén localizado en la unidad de procesamiento central; usado para hacer disponible los datos para las operaciones de los usuarios.



Como los datos se necesitan por el programa para ejecutar sus funciones, se transmiten desde el almacenamiento secundario a la memoria principal. Aunque la memoria principal puede ser capaz de almacenar varios megabytes de datos, es normalmente muy pequeña para almacenar la base de datos completa, por lo que es necesario el almacenamiento secundario.

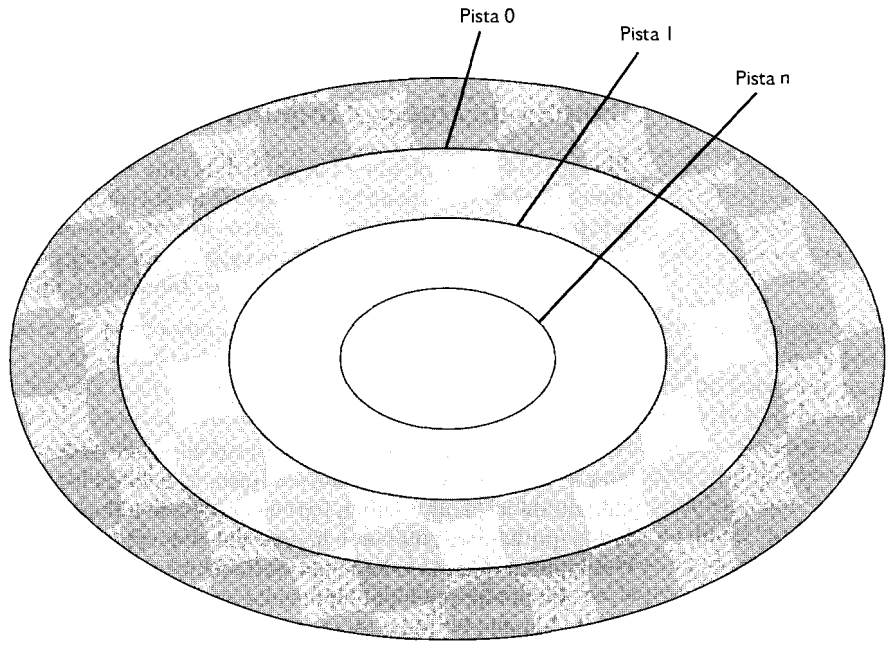
Almacenamiento secundario

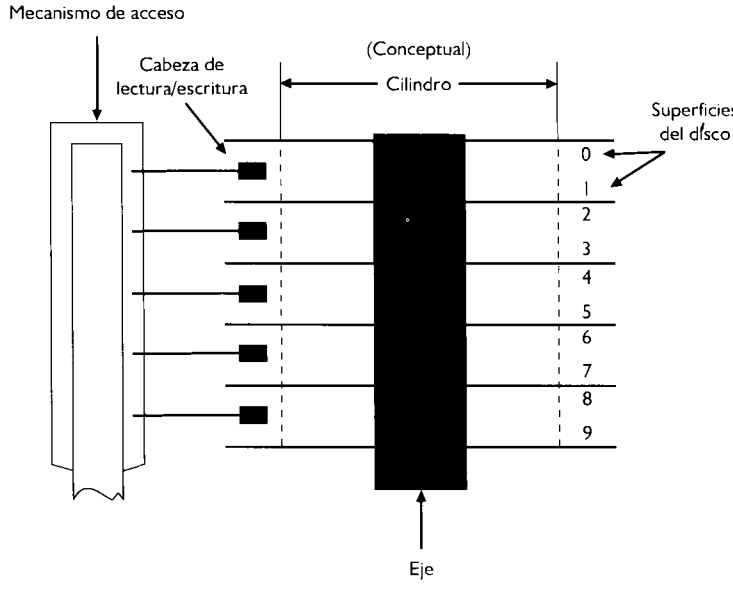
El almacenamiento secundario para los sistemas de base de datos está compuesto generalmente por el almacenamiento en disco y el almacenamiento en cinta magnética. Casi siempre, la base de datos completa se almacena en disco y porciones de ésta se transfieren desde el disco a la memoria primaria, a medida que se necesita. El almacenamiento en disco es la forma principal de almacenamiento con acceso directo, por lo que los registros individuales se pueden acceder directamente. Aunque el almacenamiento en cinta magnética es menos costoso que el almacenamiento en disco, los registros pueden ser solamente accedidos secuencialmente (y más lentamente que en disco). Su función en el sistema de base de datos está básicamente limitada a archivar datos.

controlador de disco. Unidad física que contiene la unidad de almacenamiento en disco.

La unidad física en la que está contenido el medio de grabación del disco se llama **controlador de disco** (*disk driver*). El controlador de disco contiene un paquete de disco o volumen. Las Figuras 10.2 y 10.3 muestran las componentes principales de un paquete de disco y el mecanismo de lectura/escritura para la transmisión de datos. El paquete de disco está formado por un conjunto de superficies grabables (discos) montados sobre un eje. En operación, el eje y los discos rotan a una alta velocidad. Los datos se graban sobre las pistas, que son coronas circulares grabables encontradas sobre cada superficie (Figura 10.2). Pueden existir varios cientos de pistas sobre una superficie simple. Una metáfora común para el paquete de disco es una pila de discos musicales sobre un eje, excepto que aquí las pistas son concéntricas y no en forma de espiral interna hacia el centro.

Tal como se muestra en la Figura 10.3, un conjunto de cabezas de lectura/escritura ubicadas al final de un brazo, simbólicamente como los dientes de un peine, se mueven





cilindro. La misma pista extendida a través de todas las superficies de la unidad de almacenamiento en disco.

como un grupo, de tal forma que éstos pueden ser posicionados sobre todas las pistas del mismo radio. El conjunto de dichas pistas se denomina **cilindro**. Es decir, conceptualmente, un conjunto de pistas del mismo diámetro rotando a alta velocidad forman un cilindro. Esta definición es muy útil, ya que cualquier posicionamiento de un conjunto de cabezas de lectura/escritura puede ser descrito por la localización del cilindro (por ejemplo, cilindro 199). Por lo que todas las pistas del cilindro especificado se pueden escribir o leer, sin un movimiento adicional de las cabezas de lectura/escritura. La dirección de un registro del disco normalmente necesita información sobre el número del cilindro, de la superficie y del bloque.

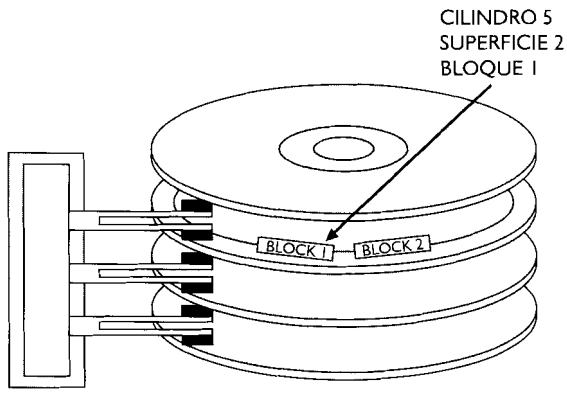
Bloques de almacenamiento físico

El registro físico o bloque es la unidad de dato más pequeña en un disco que es físicamente direccionable. Cada pista en una superficie está compuesta de un número de bloques. Un bloque puede contener uno o más registros lógicos. Supongamos que tenemos un factor de compactación de 3, esto significa que en cada bloque se almacenan tres registros lógicos. Supongamos que deseamos recuperar el registro John Jones, almacenado en la siguiente dirección:

Número de cilindro: 5
Número de superficie: 2
Número de bloque: 1

Ver la figura 10.4. Para recuperar el registro John Jones, las cabezas de lectura/escritura se mueven sobre el cilindro 5 (pista 5 en todas las superficies). Entonces se activan las cabezas de lectura/escritura para la superficie número 2 y se leen los números de bloques a la vez que la pista gira sobre las cabezas. Cuando se detecta el bloque 1, el bloque entero de tres registros lógicos se lee en memoria principal, donde se selecciona el registro John Jones.

En nuestro ejemplo suponemos la estructura más general de un disco, donde las cabezas de lectura/escritura están sujetas a un brazo movible. No todas las unidades de disco están configuradas de esta forma. En algunas las cabezas de lectura/escritura son fijas para cada cilindro. Típicamente estas unidades son más costosas, pero mas rápidas, debido a que no hay retraso en mover las cabezas de lectura/escritura sobre un nuevo cilindro.



Generalmente, el tiempo necesario para ejecutar cálculos en un bloque es mucho menor que el necesario para transferir los datos entre el almacenamiento secundario y el primario. Sin embargo, una buena estrategia de diseño es identificar, donde sea posible, los registros lógicos que probablemente se usan en las mismas operaciones y agruparlos en bloques. Por ejemplo, supongamos que una firma almacena tres tipos de alambres, A, B, y C, y que se entregan en el mismo cargamento. Si cada bloque contiene tres registros y los registros A, B y C se almacenan en bloques separados, se necesitarían tres operaciones de entrada/salida (E/S) para actualizarlos. Sin embargo, si se agrupan en el mismo bloque, entonces sólo es necesaria una operación de E/S. Debido a que generalmente el acceso a disco es un cuello de botella en las operaciones de acceso a una base de datos, una asignación cuidadosa de los registros en los bloques puede mejorar significativamente el tiempo de respuesta.

▼ Factores de rendimiento del disco

En general, hay cuatro factores que directamente afectan a la velocidad, con la que los datos se transfieren a y desde el almacenamiento en disco: tiempo de posicionamiento (*access motion time*), tiempo de activación de la cabeza, retraso de rotación y razón de transferencia.

Tiempo de posicionamiento

tiempo de posicionamiento. El tiempo necesario para posicionar las cabezas de lectura/escritura sobre el cilindro deseado.

Tiempo de posicionamiento (P), en ocasiones se conoce como *tiempo de búsqueda* (*seek time*), es el tiempo necesario para mover las cabezas de lectura/escritura desde su posición actual a una nueva dirección de cilindro. Obviamente, un movimiento hacia una posición adyacente no toma la misma cantidad de tiempo que moverse a través de toda la superficie del disco (desde la pista más interna hasta la más externa, y viceversa). Como una aproximación en los cálculos puede usarse el tiempo medio de posicionamiento:aproximadamente el tiempo necesario para moverse a través de la mitad de los cilindros, por lo que debe usarse un método más sofisticado. Un acuerdo estándar consiste en que la probabilidad de acceso para todos los registros sea igual, brindando una distribución de probabilidad uniforme. El promedio para una distribución uniforme se encuentra en el medio entre los valores extremos. Para el tiempo de posicionamiento, el valor extremo pudiera ser (1) mantenerse posicionado sobre el cilindro actual, o (2) moverse desde el cilindro más interno hacia el más externo (o viceversa). Dado que asumimos la distribución uniforme, la media sería el tiempo para moverse a través de la mitad de los cilindros. De doce a veinte

milisegundos es el tiempo medio típico de posicionamiento, de acuerdo con el modelo y la composición del controlador de disco [1].

Tiempo de activación de la cabeza

tiempo de activación de la cabeza. El tiempo necesario para activar una cabeza de lectura/escritura.

Tiempo de activación de la cabeza es el tiempo necesario para activar electrónicamente la cabeza que se encuentra sobre la superficie cuando ocurre la transferencia de datos. En comparación con otros factores de rendimiento, generalmente este tiempo es despreciable. Por esta razón rara vez se usa en los cálculos de rendimiento.

Retraso de rotación

retraso de rotación. El tiempo que necesita el disco para rotar el registro solicitado bajo la cabeza de lectura/escritura

Retraso de rotación o latencia es el tercer factor de tiempo. Representa la cantidad de tiempo que necesita el bloque seleccionado para rotar la cabeza, de forma tal que la transferencia de datos pueda comenzar. El tiempo de rotación depende de dos factores: a qué rapidez rota el disco y la localización del bloque buscado en relación con el tiempo de activación de la cabeza de lectura/escritura. Físicamente este tiempo puede variar desde cero hasta el tiempo necesario para completar una revolución del disco (R). Analógicamente, supongamos que se desea montar sobre el caballo púrpura en el carrusel (asumiendo que existe sólo un caballo de este tipo). Si compra un ticket y corre a montarse en el carrusel, la probabilidad de que el caballo púrpura puede estar justo donde se detuvo sería la misma que para cualquiera de los otros caballos. Si fuera un fanático y lo intentara varias veces, puede que alguna vez se detenga frente al caballo púrpura. También puede encontrarse con la situación en que lo pierda y tenga que esperar por una vuelta completa del carrusel. Como media, espera media vuelta para montarse en el caballo púrpura. La moraleja de esta historia es que los cálculos de rendimiento generalmente asumen un retraso de rotación media de $R/2$.

Velocidad de transferencia de datos

razón de transferencia de datos. La razón en la cual los datos se leen del disco hacia la memoria principal, o equivalentemente, la razón en la cual los datos se escriben desde la memoria principal al disco.

Razón de transferencia de datos (D) se refiere a la cantidad de tiempo necesario para transferir los datos desde el disco hacia (o desde) la memoria principal. Depende de la velocidad de rotación y de la cantidad de datos almacenados. El tiempo de transferencia de datos normalmente se expresa en cientos de bytes por segundo.

Tiempo de transferencia de datos

El tiempo previsto (T) para acceder a una dirección en disco y transferir un bloque de datos se estima como

$$T = P + \frac{R}{2} + \frac{L}{D}$$

donde P es el tiempo de posicionamiento, R es el retraso de rotación, L es el tamaño del bloque en bytes y D es la velocidad de transferencia de datos.

Ejemplo de un registro accedido aleatoriamente. Snpongamos que los registros de reclamación de una compañía de seguros se almacenan en bloques de tres registros en el disco (un factor de compactación de tres) y que cada registro de reclamación ocupa 200 bytes. La velocidad de transferencia de datos es de 806.000 bytes por segundo. El tiempo de posicionamiento medio es de 30 milisegundos. La unidad de disco rota a una velocidad

[1] En el momento de realizar esta traducción ya es normal encontrar tiempos de acceso menores (N. del T.).

de 3.600 vueltas por minuto. Supongamos que un usuario llama para averiguar el estado de una reclamación. ¿Cuál es el tiempo de transferencia de datos para buscar el bloque de datos? Para responder esta pregunta se asignan valores apropiados a las variables anteriores en la forma siguiente.

$$A = 0{,}030 \text{ segundos}$$

$$\text{Revoluciones por segundos} = \frac{7.200}{60} = 120$$

$$R = \frac{1}{120} \text{ segundos} = 0{,}0083 \text{ segundos}$$

$$\frac{R}{2} = 0{,}0083 \times \frac{1}{2} \text{ (la media de espera es de media revolución)} = 0{,}00415$$

$$\frac{L}{D} = 600/806000 = 0.00074 ,$$

Por lo que,

$$T = 0{,}030 + 0{,}00415 + 0{,}00074 = 0{,}03489 \text{ segundos}$$

Ejemplo de un registro accedido secuencialmente. Ahora analizaremos el cálculo del tiempo medio de acceso a un registro en un archivo accedido secuencialmente. Supongamos que en vez de responder al acceso aleatorio de un bloque de datos, como en el ejemplo anterior, estamos actualizando el archivo de un usuario de la compañía de seguros con los pagos recibidos a principios de mes. Tiene sentido que estos archivos estén organizados secuencialmente por el número de la póliza y que estén localizados en bloques secuenciales por cilindros. Esto significa que primero se rellena el cilindro N con bloques secuenciales, después el N +1 y así sucesivamente. De esta forma se minimiza el tiempo de movimiento de la cabeza. En particular, si las cabezas de lectura/escritura se encuentran en el primer cilindro, entonces todos los registros en este cilindro se transfieren sin un tiempo de posicionamiento adicional. Por lo que, en el cálculo del tiempo de acceso medio para cada registro de un archivo procesado secuencialmente, el tiempo de posicionamiento es despreciable y se ignora.

Pudiera existir un pequeño retraso cada vez que la función de lectura/escritura cambia desde una pista de un cilindro hacia otra. Esto es necesario con el objetivo de disminuir pequeñas diferencias en la alineación de las pistas sobre diferentes superficies. Para nuestros propósitos, este retraso puede ser aproximadamente el tiempo necesario para dar una media vuelta del paquete de disco. Una vez se haya encontrado el registro inicial de la nueva pista, se pueden transmitir el resto de los bloques sobre la pista. Por tanto, si el archivo del usuario ocupa ocho pistas sobre el cilindro, el número de retraso de medias vueltas sería 8.

Supongamos ahora que cada pista contiene 1.000 bloques. Tenemos un total de 8.000 bloques; si los mismos tienen un factor de compactación de tres, tenemos 24.000 registros de pólizas. Asumamos, como antes, que cada registro contiene 200 bytes, entonces nuestros bloques ocupan 600 bytes. Si procesamos secuencialmente un archivo completo, el tiempo medio para acceder a un registro se calcula como sigue

$$\text{Tiempo total para leer todos los bloques} =$$
$$= 0{,}00415\ (8) + 0{,}00074\ (8000) = 0{,}0664 + 5{,}92 = 5{,}9232$$

$$T = \frac{5{,}9532}{8000} = 0{,}00074415 \text{ segundos}$$

T representa el tiempo de transferencia medio para un registro accedido secuencialmente en el archivo de pólizas.

▼ Formatos de almacenamiento de datos en disco

En esta sección examinaremos los aspectos físicos de la manipulación de datos sobre el disco. Veremos los formatos sobre las pistas y en los registros físicos, y también la manipulación de la entrada/salida.

Formatos de pistas

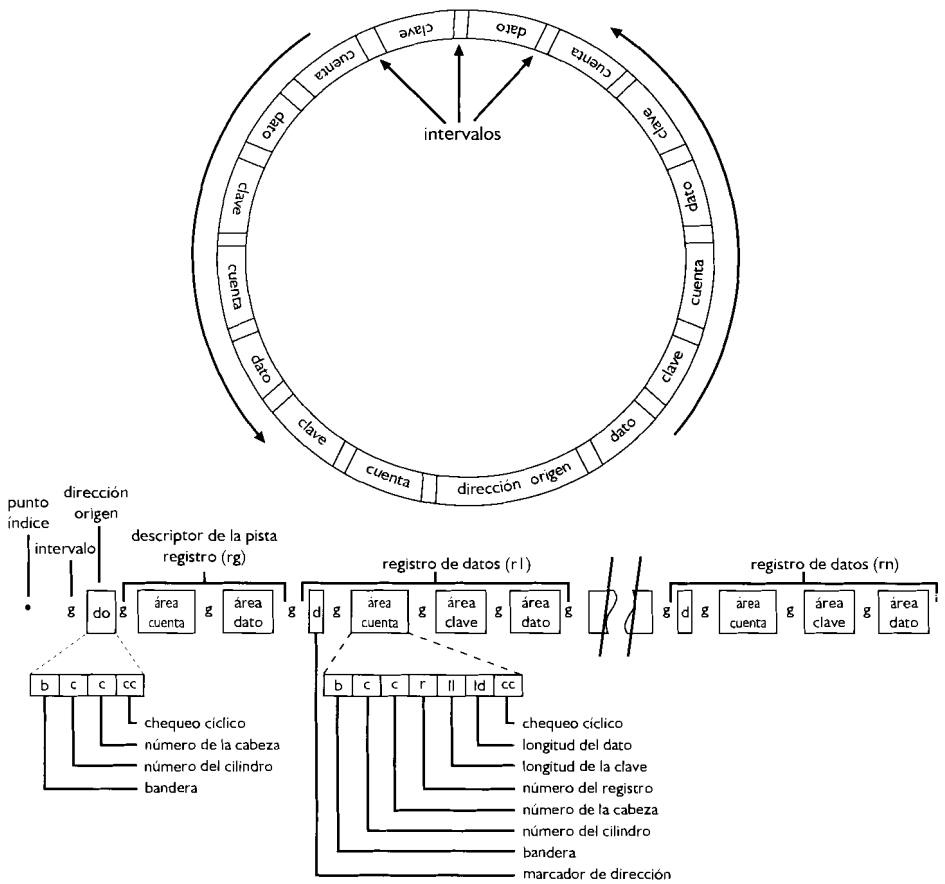
formato cuenta-clave. Un formato de datos para pistas que usan claves externas.

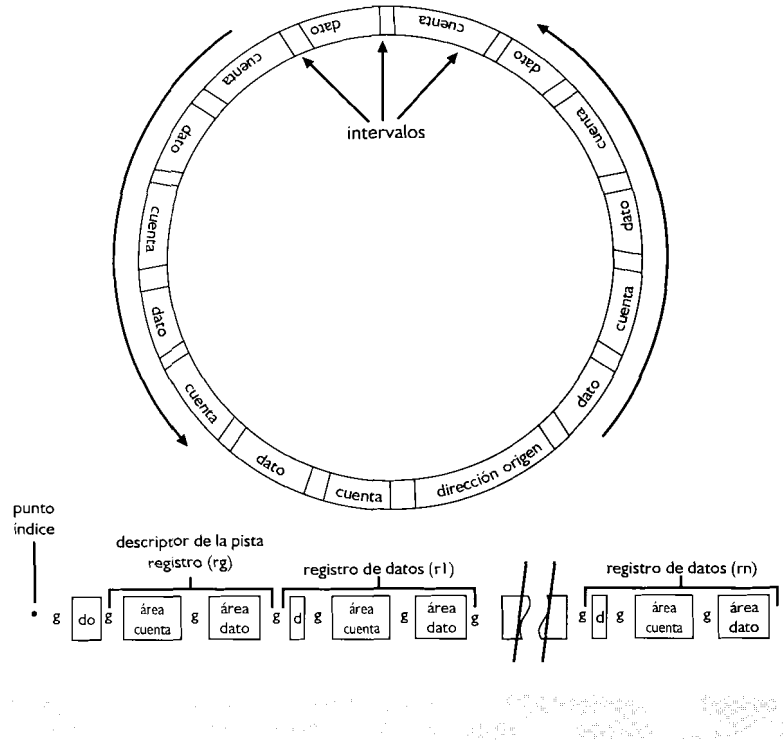
Los registros pueden guardarse en el disco en uno u otro formato, un **formato cuenta-clave** y otro **formato cuenta-dato**, tal como se muestra en las Figuras 10.5 y 10.6. La diferencia fundamental es que el formato cuenta-clave incluye una clave que es externa al registro de dato como tal. Esta clave se utiliza por el sistema operativo para acceder a un registro en particular. Usaremos el término *registro* en el sentido más amplio de un registro físico, el cual pudiera ser un bloque. Ambos formatos, el de cuenta-dato y cuenta-clave, serán descritos por las definiciones siguientes.

formato cuenta-dato. Un formato de datos para pistas que no usan claves externas.

Cada pista tiene un punto de entrada, que constituye una marca especial para identificar el comienzo de cada una. Como la pista es circular, éste también identifica el final de la pista.

La dirección origen (DO) caracteriza al cilindro y el número de cabezas de lectura/escritura que contiene la pista, así como el estado de la pista (bandera), si está defec-





tuosa o no. Si la pista está defectuosa se usará otra pista. Se incluyen dos-bytes de chequeo cíclico con el objetivo de detectar errores en las operaciones de entrada/salida.

Los intervalos (G) separan las diferentes áreas sobre la pista. La longitud del intervalo varía de acuerdo con el dispositivo, su localización y la longitud del área precedente. El intervalo que sigue al punto índice es diferente con respecto al que sigue a la dirección origen y la longitud de los intervalos que siguen a un registro dependen de la longitud de este último. Esto se debe a la necesidad de brindar un tiempo adecuado para las funciones requeridas por el equipamiento cuando la cabeza de lectura/escritura rota sobre el intervalo. Estas funciones varían de acuerdo con el tipo de área que le precede.

El marcador de dirección (D) es un segmento de dos-bytes suministrado por la unidad de control (hardware que controla el controlador de disco) cuando se escribe en el registro. Esto permite a la unidad de control localizar el comienzo de un registro en un tiempo mínimo.

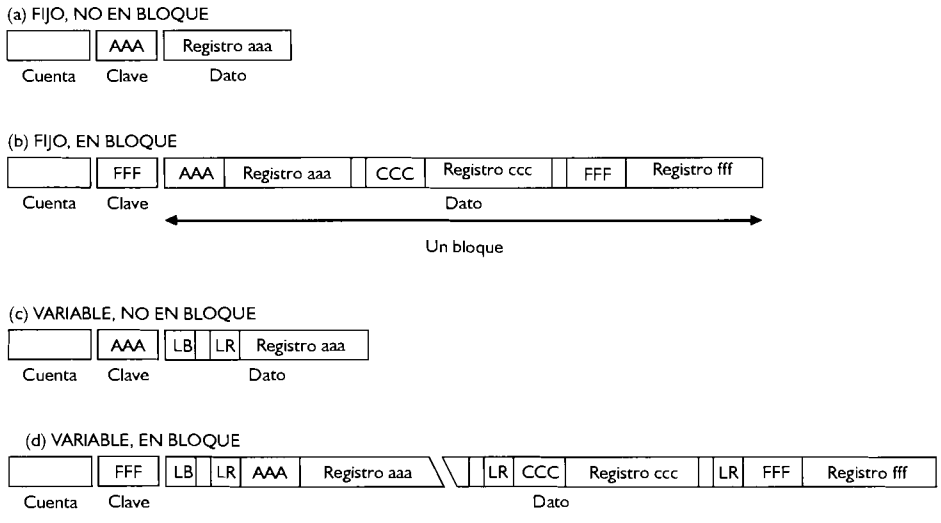
En la Figura 10.5 se detalla el área de cuenta. El campo bandera duplica la información acerca de la condición de la pista y añade información usada por la unidad de control. El número del cilindro, el cabezal y los campos del registro lo identifican unívocamente. El campo longitud-clave es un campo de un-byte. Este campo siempre tiene un 0 si el registro se encuentra en el formato cuenta-dato. El campo longitud-dato contiene dos bytes, que especifican el número de bytes en el área de datos del registro, excluyendo el chequeo cíclico. El chequeo cíclico contiene dos bytes para la detección de errores.

Formatos de registros

registro físico. Un bloque físico de datos.

Los **registros físicos** o bloques se almacenan sobre las pistas en cualquiera de los cuatro formatos que se muestran en la Figura 10.7.

Registros de longitud fija. En este caso todos los registros son de la misma longitud. Si los registros físicos no están compactados en un bloque, existirá un registro lógico (por



ejemplo, un registro nómina) por cada registro físico (los datos que están almacenados en el área registro de la pista). Si los registros forman un bloque, entonces cada registro físico podrá contener más de un registro lógico. Por ejemplo, si existen tres registros nómina que constituyen cada registro físico, tenemos entonces registros de bloque con un factor de bloque tres. En este caso, al área clave generalmente se le asigna la clave del registro *más alto* en el bloque. Esto facilita la localización de registros de interés. Supongamos que tenemos dos bloques sucesivos que contienen los registros 10, 12, 14 y 15, 19, 24, respectivamente. Si el sistema operativo busca el registro lógico 15, la clave del primer bloque es 14. por lo tanto el registro 15 no está en el primer bloque. La clave del segundo bloque es 24. Como 24 es mayor que 15, el registro 15 debiera estar en este bloque. Entonces el bloque completo se lee en la memoria principal donde se realiza la búsqueda del registro 15.

Registros de longitud variable. El formato longitud-variable, como su nombre indica, permite que la longitud de los registros varíe. Si el usuario de la base de datos necesita un registro lógico del cliente que almacena datos acerca de las facturas pendientes, este formato sería apropiado, ya que el número de facturas varía de acuerdo al cliente. Como la longitud del registro no es uniforme, se necesita un método que indique dónde termina éste. Esta información se encuentra en las áreas LB (longitud-bloque) y LR (longitud-registro).Como en el formato de longitud-fija, si el registro no es un bloque indica que cada bloque contiene exactamente un registro lógico. El formato de bloque permite que puedan almacenarse en un bloque varios registros lógicos.

Gestión de entrada/salida

Basados en los conceptos de formatos de datos de esta sección, consideraremos brevemente las operaciones de entrada/salida del SGDB. Supongamos que se ejecuta una instrucción de E/S recibida desde un usuario o desde un programa de aplicación. El SGDB primero chequea si el subesquema asociado con la instrucción de E/S se encuentra en su diccionario de datos, así como que si al usuario o al programa, que es la fuente de una orden tal, le está permitido acceder al subesquema. Suponiendo que todo esté bien, el SGDB manda las órdenes pertinentes de E/S al sistema operativo para acceder a los registros físicos específicos. Entonces el sistema operativo busca en los dispositivos de almacenamiento secundario y accede a los registros físicos apropiados. El sistema operativo transfiere estos regis-

tros a la memoria principal, donde el SGDB extrae de los registros físicos aquellos registros lógicos solicitados y los pasa al usuario o a la aplicación para su posterior utilización.

▼ Organización de archivos y métodos de direccionamiento

Hasta ahora se ha aprendido algo acerca de los dispositivos que se usan para almacenar datos y las operaciones de E/S usadas para transmitir datos hacia y desde dichos dispositivos. Ahora consideraremos los métodos de organización y direccionamiento de estos datos sobre dichos dispositivos para facilitar el almacenamiento y las operaciones de E/S.

Existen tres formas básicas de organización física de archivos sobre los dispositivos de almacenamiento: organización secuencial, organización secuencial-indexada y organización directa. Éstas no son un conjunto completo de todas las opciones de organización posibles, pero aquellas que se omiten no son más que modificaciones de estos tipos básicos. Por lo tanto, no es necesario ser exhaustivos, con vistas a cubrir los conceptos esenciales.

En la presentación de este tema, los términos *organización* y *acceso* se usan a menudo libremente, pero no son intercambiables. La razón es que la forma en que los datos estén almacenados está estrechamente relacionada con el método de acceso. En las siguientes secciones se intenta aclarar esto.

Organización secuencial de un archivo

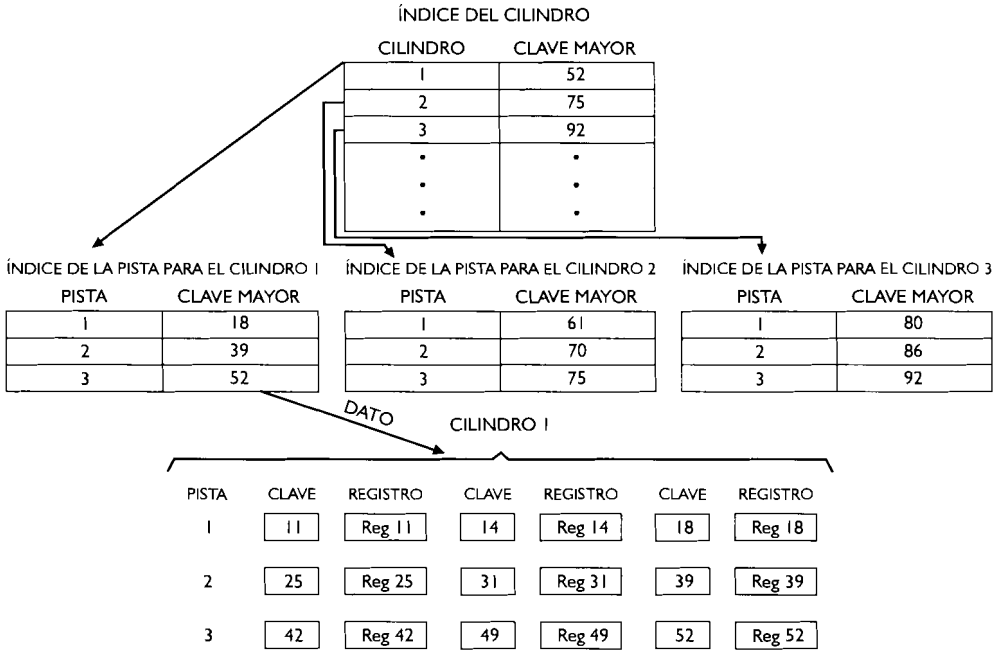
La organización secuencial de un archivo significa que los registros se almacenan adyacentes unos a otros, de acuerdo con la clave, como son el número del empleado, el número de cuenta, entre otros. Una implementación convencional organiza los registros en orden ascendente de los valores de la clave. Este es un método eficiente de organización de registros cuando una aplicación, como el programa de nómina, actualiza un número insignificante de los registros ya almacenados.

Si un archivo secuencial se encuentra sobre una cinta magnética, sus registros se acceden sólo de forma secuencial. Esto significa que, si deseamos acceder al décimo registro secuencial, generalmente se deberán leer los anteriores nueve registros. El acceso directo a un registro en particular no es posible. Como consecuencia, las cintas magnéticas no son propias para las operaciones de bases de datos y normalmente se usan para guardar los archivos de diario y almacenar información de archivos.

Organización secuencial-indexada de un archivo

Consiste en que los archivos están organizados secuencialmente; sin embargo, es posible acceder directamente a los registros. La organización secuencial-indexada de un archivo brinda facilidades para acceder a los registros de ambas formas, secuencial y directamente. Los registros se almacenan en la secuencia física usual por la clave primaria. Además, se almacena en el disco el índice de la localización del registro. Esto permite el acceso secuencial a los registros para aquellas aplicaciones que realicen una gran cantidad de actualizaciones y, por otra parte, el acceso directo de acuerdo con las solicitudes de los usuarios.

En la Figura 10.8 se muestra una versión simplificada de cómo opera el acceso secuencial-indexado. Los índices y registros se almacenan en disco. Limitamos grandemente el número de cilindros y pistas para los propósitos de nuestro ejemplo. En primer lugar debe observarse que los registros se encuentran organizados secuencialmente en las tres pistas del cilindro 1. El procesamiento secuencial se realiza comenzando por el primer registro del archivo, procediendo entonces a través del archivo desde el primer registro hasta el último. También se facilita el acceso directo a los registros. Supongamos que deseamos recuperar el registro 31. Al buscar en el índice del cilindro encontramos que dicho registro se encuentra en el cilindro 1. Ya que la clave del registro de numeración más alta



en el cilindro 1 es el 52, y como los registros se ordenaron secuencialmente por la clave, el registro 31 debe estar en el cilindro 1. La búsqueda del índice de la pista para el cilindro 1 muestra que el registro 31 está en la pista 2. En esta situación, la cabeza de lectura/escritura se mueve hacia el cilindro 1 y se activa la cabeza de lectura/escritura de la pista 2. Sobre la pista 2 se realiza una búsqueda secuencial para localizar el registro 31. Aunque la búsqueda secuencial no ha sido completamente eliminada, su alcance se redujo suficientemente como para justificar el término *acceso directo*.

Nuestro ejemplo utiliza registros que no son bloques, pero puede extenderse fácilmente a registros que sean bloques. Pensemos en el área de datos como un contenedor de, digamos, tres registros y consideremos el área clave como la clave del registro más alto en el bloque. El procedimiento de búsqueda procedería de modo similar.

Organización directa de un archivo

Hasta aquí se han discutido dos formas de organización de archivos: secuencial y secuencial-indexada. Se han destacado simultáneamente los dos métodos de acceso a un archivo: acceso secuencial y acceso directo. Los registros en una simple organización secuencial de un archivo se acceden solamente de forma secuencial. Los registros en una organización secuencial-indexada de un archivo se acceden directa y secuencialmente. Se pasará ahora a la discusión de un tercer tipo de organización de archivos llamada directa o *hash*. Solamente los métodos de acceso directo son aplicables a este tipo de organización de archivos.

Funciones Hash estáticas

Una de las desventajas del esquema indexado es que se debe acceder y leer el índice para localizar los registros. El uso de técnicas *hashing* como método de direccionamiento elimina la necesidad del mantenimiento y búsqueda de índices. La eliminación del índice evita la necesidad de hacer dos accesos al almacenamiento secundario para acceder a un registro: uno para leer el índice y otro para acceder al archivo.

Existen muchas aplicaciones cuya necesidad predominante es al acceso directo a los registros, y cuyos servicios podrían estar sobrecargados por el uso de métodos de indexación. Estos ejemplos abarcan los sistemas de reserva de líneas aéreas, hoteles y alquiler de coches, así como la transferencia de fondos públicos.

La idea básica consiste en la negociación del tiempo y el esfuerzo que se emplea en el almacenamiento, mantenimiento y búsqueda de un índice por el tiempo que necesita la unidad central de proceso (CPU) para ejecutar un algoritmo hash, que genera la dirección del registro. El algoritmo hash es un procedimiento que calcula la dirección de un registro desde uno de sus campos, generalmente la clave. Continuaremos con un ejemplo que nos ayude en la ilustración del método.

Supongamos que se almacenan 500 registros de nómina, de 100 bytes cada uno, en un disco magnético que tiene una capacidad de 2.000-bytes por bloque. Se necesitarían exactamente 25 bloques si direccionamos uno y sólo uno de los registros por cada localización posible en cada uno de los 25 bloques. Ya que no existe hasta la fecha ningún algoritmo hash, h(ki) (donde ki es la clave del registro), que pueda lograr esto, normalmente se añade espacio de almacenamiento adicional, por ejemplo, un 20 por 100, para reducir el número de instancias cuando el algoritmo calcula la misma dirección para más de un registro (una colisión). La razón entre el espacio que realmente necesitan los registros en el archivo y el espacio real reservado para el mismo se llama *factor de carga*. En nuestro ejemplo, los registros necesitan 25 bloques de espacio de almacenamiento, pero reservamos (25) × (1.2) = 30 bloques para reducir el número de colisiones. El factor de carga es (25/30 =) de un 83 por 100.

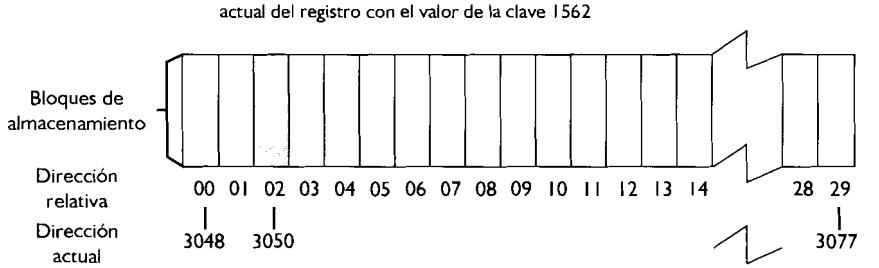
Estrategias efectivas para el almacenamiento directo a registros tienen tiempos de búsqueda lentos y tienen algunas colisiones. Esto se completaría mejor con la selección de un factor de carga bajo, usando un factor de compactación alto y un algoritmo que distribuya uniformemente los registros sobre el área de almacenamiento. El mejor método para lograr este último objetivo es mediante el uso del método de la división con resto, que ilustraremos a continuación.

Después de reservar los 30 bloques para almacenar nuestro archivo, una distribución uniforme de los registros hacia los bloques significa que aproximadamente cuatro de las cinco posibles direcciones de almacenamiento se corresponden con un registro. Supongamos que el primer registro que se almacena tiene una clave (sobre la que trabaja el algoritmo) con valor 1562. Haremos lo siguiente (Figura 10.9):

Paso 1: Dividir la clave por el tamaño del archivo reservado en bloques

$$
\begin{array}{r}
52 \\
30\overline{)1562} \\
-150 \\
\hline
62 \\
-60 \\
\hline
2 \text{ resto}
\end{array}
$$

Paso 2: Añadir el resultado del Paso 1 a la dirección inicial para obtener la dirección actual del registro con el valor de la clave 1562



1. Dividir la clave por el número de bloques de almacenamiento reservados. El resto nos da la dirección relativa de almacenamiento.
   Resto de (1562/30) = 2
   Bloque relativo de almacenamiento = 2
   Conservar el resto es en realidad una garantía de que una dirección siempre se calcula en el rango de 0 al 29, incluyendo este último.
2. Sumar el resultado obtenido en (1) con la dirección del primer bloque para obtener la dirección en la cual se almacena el registro.

Supongamos que nuestro conjunto de 30 bloques comienza en la dirección 3048 y se extiende hasta la dirección 3077. Entonces el registro 1562 se encuentra en el bloque (2 + 3048 =) 3050. Si existe únicamente un registro almacenado en cada bloque, entonces ningún otro registro se almacenará en la dirección 3050. Si en cada bloque se almacena más de un registro, éstos se seguirán almacenando en dicha dirección hasta que se llene el bloque.

Supongamos que cada bloque contiene uno y sólo un registro, y que durante el proceso de hash encontramos un segundo registro cuya clave es 1592. Si volvemos a calcular con la dirección relativa 2, obtenemos una colisión. Uno de los métodos utilizados para el tratamiento de colisiones es el uso de una función de división hash con un refinamiento, que se conoce como *el método del cociente cuadrático*. Cuando no hay colisiones, el direccionamiento se realiza exactamente como vimos antes, pero si ocurren colisiones, tenemos que continuar con otro método.

Con el método del cociente cuadrático, cuando la clave se divide por el tamaño de la tabla, ambos el cociente, C, y el resto, R, se guardan. Usando C y R, podemos generar una secuencia de direcciones relativas a registros mediante la fórmula

$$(R + C \times i^2 + i) \text{ (módulo el número de bloques)}$$

donde i se incrementa a través de los enteros, comenzando por el 0 hasta que no existan colisiones.

Continuando con nuestro ejemplo, para el registro 1562 tenemos,

$$(2 + 52 \times 0^2 + 0) \text{ módulo } 30 = 2$$

igual que antes. (Módulo 30 es el resto de la división por 30.) Pero para el nuevo registro 1592 obtenemos:

$$(2 + 53 \times 0^2 + 0) \text{ módulo } 30 = 2$$

El registro 1562 ya se encuentra en esta dirección relativa, por lo que iteraremos sobre el valor de i hasta encontrar una dirección desocupada. Para este ejemplo, volvemos a calcular

$$(2 + 53 \times 1^2 + 1) \text{ módulo } 30 = 26$$

y obtenemos una dirección desocupada donde almacenar el registro.

Cuando busquemos el registro 1592, el algoritmo se ejecuta de la misma forma; primero nos conduce a la ubicación del registro 1562. Como éste no es el registro buscado, repetimos el algoritmo hasta encontrar el registro deseado.

Otro método para el tratamiento de las colisiones consiste simplemente en almacenar un puntero en la posición calculada por el algoritmo. Este puntero indicaría la dirección donde el próximo registro se encuentra y cuya clave también coincidió con esta ubicación al aplicar la función hash. Si existen muchos registros de este tipo, se pudiera mantener un

puntero por cada localización de almacenamiento. De esta forma existiría una cadena de punteros que se pueden recorrer hasta encontrar el registro deseado.

Funciones Hash dinámicas

Como hemos visto, la función estática hash es sumamente simple. Sin embargo, a medida que la base de datos crece, esta función pierde su sentido. Aparecen un número creciente de colisiones, que traen como resultado una sobrecarga en el acceso a los registros. Una de las estrategias para el tratamiento de este problema es reservar desde el comienzo un espacio estimado para requisitos futuros; pero esto desperdicia espacio de almacenamiento. Otra solución es reservar un espacio adicional y reorganizar el archivo a medida que crezca. Ambas soluciones implican un sobrecosto a la hora de recalcular la nueva función hash para cada registro en el archivo y generar las nuevas asignaciones de bloques.

La *función hash dinámica* nos brinda una mejor solución. En ocasiones, el método que mostraremos se llama *hash extensible*. El hash extensible divide y combina los bloques a medida que la base de datos crece o se reduce. Esto nos garantiza una utilización eficiente del espacio. Por otra parte, si cada vez que se reorganiza se implica a un solo bloque, el costo asociado es mínimo.

El hash extensible usa una función hash, h, que posee características importantes, como son la aleatoriedad y la uniformidad. También (generalmente) utiliza una cadena binaria de 32-bits con el objetivo de crear e identificar los índices de los bloques. (Ya que se ha discutido uno de los algoritmos de hash, el lector debe entender las ideas básicas que lo caracterizan. Por lo tanto, para evitar confusiones, no definiremos específicamente la función hash, h.)

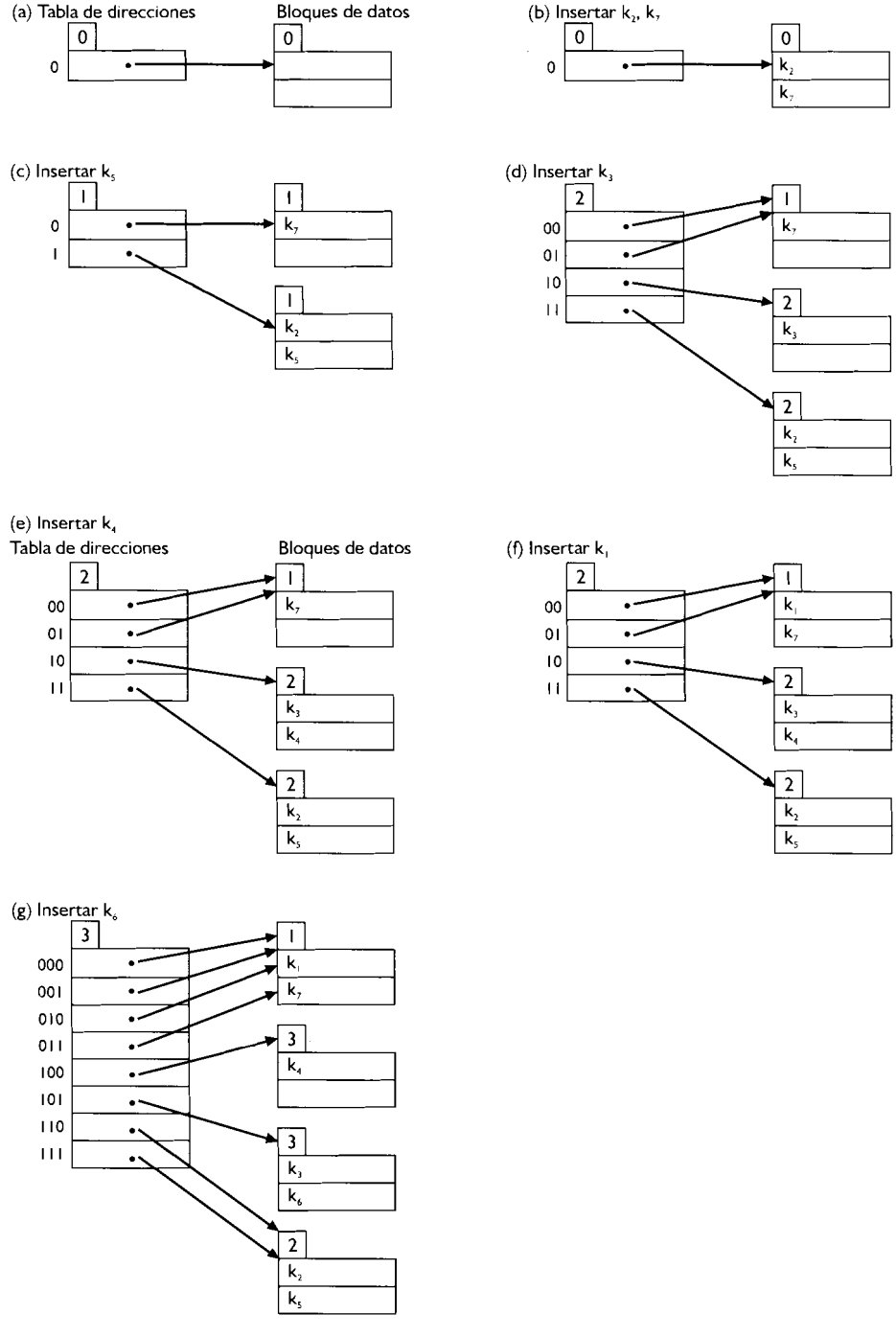
La mayoría de los algoritmos usan i bits en cualquier momento, donde $0 \pounds i \pounds 32$. Para empezar, fijemos $i = 0$. El valor de i crecerá y se reducirá cuando el tamaño de la base de datos crezca o se reduzca. En la Figura 10.10 se muestra un esquema del método hash extensible. Para determinar el bloque que contiene la clave, k_i, haremos lo siguiente:

1. Calcular $h(k_i) = x$ (representada como uua cadena binaria).
2. Tomar los primeros i bits de mayor orden de x.
3. Usar los i bits como un desplazamiento dentro de la tabla de direcciones a bloques, y seguir al puntero hasta el bloque correspondiente.

Para ilustrarlo, consideremos una colección de registros de inventario que tienen claves k_1, k_2, k_3, k_4, k_5, k_6, k_7. Supongamos que los bits de mayor orden de $h(k_i)$, donde k_i es una clave, son los que se muestran en la Figura 10.10. Inicialmente, tenemos el archivo vacío mostrado en la Figura 10.11(a). El 0 que aparece encima de la tabla que contiene la direcciones de los bloques indica que se requieren 0 bits de h(ki) para calcular la dirección del bloque para la clave k_i.

Primeramente insertemos los registros con claves, k_2 y k_7. (Para seguir el ejemplo con claridad, supongamos que cada bloque de memoria contiene un máximo de dos registros.)

$h(k_1) = 0010...$
$h(k_2) = 1101...$
$h(k_3) = 1011...$
$h(k_4) = 1000...$
$h(k_5) = 1110...$
$h(k_6) = 1010...$
$h(k_7) = 0100...$



La tabla de direcciones apunta a un bloque de memoria que está vacío, en el cual se insertan los registros, tal como se muestra en la Figura 10,11(b). Si después intentamos insertar el registro k_5, encontramos que el bloque al cual apunta la tabla de direcciones está lleno. Entonces necesitamos incrementar la cantidad de bits que representan el valor

del hash. Es decir, previamente usamos 0 bits cuando teníamos un solo bloque de memoria. Ahora debemos usar 1 bit, para poder utilizar $2^1 = 2$ bloques de memoria. Esto requiere que aumentemos la tabla de direcciones para permitir dos entradas, una para cada bloque de datos.

Por consiguiente, dividimos el bloque de memoria, colocando aquellos registros que tienen al "0" como el bit de mayor-orden del valor del hash en un bloque, y aquellos que tienen al "1" como el bit de mayor-orden en otro bloque. Para nuestro ejemplo, el registro k_7 se coloca en el bloque-0 porque su bit más izquierdo en $h(k_7)$ es 0, y los registros k_2 y k_5 se colocan en el bloque-1 porque el bit más izquierdo de ambos valores $h(k_2)$ y $h(k_5)$ es 1 (Figura 10.11(c)).

Ahora insertaremos el registro k_3, que tiene a "1" como su valor de mayor-orden - pero el bloque-1 está lleno. Nuevamente, tenemos que incrementar el número de bits que se usan en los valores del hash. Usaremos 2 bits, permitiendo $2^2 = 4$ bloques, tal como se muestra en la figura 10.11(d). El bloque-1 inicial se divide en dos bloques, mostrando "2" en el valor de *i*. Sólo el registro k_7 tiene como valor de mayor-orden al "0", por lo que ambos punteros desde las posiciones "00" y "01" apuntan al bloque-1.

Como se observa, los registros k_2, k_3 y k_5 contienen "1" en la posición de mayor-orden, por lo que su almacenamiento se basa en el siguiente valor de mayor-orden.

Suponga que se inserta el registro k_4. El registro $h(k_4)$ tiene a "10" en sus dos posiciones de mayor-orden y se inserta en el bloque que contiene el registro k_3. Ver Figura 10.11(e).

Insertemos ahora el registro k_1. Sus dos bits de mayor-orden son "00". El puntero desde la fila "00" de la tabla índice apunta a un bloque que tiene una posición vacante, por lo que el registro 1 se inserta junto con el 7, ver Figura 10.11(f).

Por último, cuando consideramos al registro k_6, todos los bloques disponibles están llenos. Ahora ajustamos la tabla índice para usar los tres bits de mayor-orden, permitiendo el uso de $2^3 = 8$ bloques. Los dos bits de mayor orden del registro k_6 son "10". Los cuales coinciden con el de los registros k_3 y k_4 que se encuentran en el segundo bloque. Por lo tanto, dividimos este bloque. Ambos registros k_3 y k_6 tienen "1" en el tercer bit de mayor-orden, por lo que se almacenan juntos, y el registro k_4 se inserta en el otro bloque (Figura 10.11(g)). Observe que si los tres registros tuvieran el mismo valor en el tercer bit de mayor-orden, sería necesario analizar el valor del cuarto bit, y así sucesivamente.

Si el archivo se reduce, el proceso se invierte.

Observe que la sobrecarga se reduce porque la memoria se reserva, cuando es necesaria para ambos la tabla de direcciones y los bloques de memoria. El espacio desperdiciado es mínimo. Además, se simplifica la reorganización porque se involucra sólo a un bloque en cada paso.

▼ Implementación de interrelaciones lógicas

Ahora delinearemos los fundamentos de las estructuras de datos físicas, que son la estructura molecular básica del universo de las bases de datos para desempeñar sus funciones. Estas estructuras de datos físicas se caracterizan de dos formas, la primera consiste en cuál de los registros de la base de datos se enlaza con otro, y la segunda en la forma que se usan estos enlaces para soportar las operaciones de la base de datos.

Listas enlazadas

puntero. Un elemento de dato que contiene una dirección física.

Un concepto fundamental en el enlace de un registro físico con otro es el uso de los punteros. Un puntero es un campo asociado con un registro de dato que se usa para encontrar un registro de dato relativo. ¿Qué significa esto?

Supongamos que la Corporación Zeus almacena información acerca de su personal en forma de registros lógicos, tal como se muestra en la Figura 10.12. Un campo de interés es OFICIO. Supongamos que se abre una nueva posición para un ingeniero. Para una

LISTA CABECERA

Contador = 1	Seattle = 1
Dibujante = 2	Los Angeles = 2
Ingeniero = 3	Portland = 6

NÚMERO DEL REGISTRO	NÚM. EMPLEADO	NOMBRE	UBICACIÓN	OFICIO	PUNTERO OFICIO	PUNTERO UBICACIÓN
1	0123	James	Seattle	Contador	4	3
2	0211	Poirot	Los Angeles	Dibujante	6	4
3	0223	Smith	Seattle	Ingeniero	5	7
4	0245	Cubic	Los Angeles	Contador	7	5
5	0301	Black	Los Angeles	Ingeniero	9	0
6	0401	Iwerks	Portland	Dibujante	10	8
7	0601	Ivans	Seattle	Contador	8	9
8	0711	Nell	Portland	Contador	0	10
9	0908	Steel	Seattle	Ingeniero	0	0
10	1067	Schwartz	Portland	Dibujante	0	0

determinada evaluación puede desearse recuperar todos los registros del personal que es ingeniero. Obsérvese que para cada registro del personal, el elemento de dato PUNTERO OFICIO contiene la dirección del próximo registro con el mismo valor que tiene el registro actual en el campo OFICIO. Una **lista cabecera**, que se encuentra separada en el disco, apunta al primer registro de la base de datos que contiene el valor "ingeniero" en el campo OFICIO. En nuestro ejemplo sería el registro Smith's. Este registro apunta hacia Black, que a su vez apunta hacia Steel. Estas cadenas de punteros nos suministrau un conjunto de registros enlazados que contienen valores comunes para atributos de gran interés. Una lista de registros enlazados por dicha cadena de punteros se llama una **lista enlazada**. En nuestro ejemplo, la misma facilita la rápida recuperación de un conjuuto de registros que son ingenieros. Obviamente las listas enlazadas necesitan algún tipo de organización con acceso-directo sobre el archivo con el objetivo de desempeñar su función.

lista cabecera. Una lista de punteros, cada uno de los cuales apunta hacia el primer registro del archivo.

lista enlazada. Un conjunto de registros físicos enlazados por punteros que se encuentran dentro de los mismos registros.

Podríamos llevar a cabo el mismo objetivo si los registros del personal estuvieran organizados físicamente sobre el disco, de tal forma que todos los ingenieros estuvieran en secuencia comenzando con el primer registro. No existiría la necesidad de los punteros para recuperar todos los registros para ingenieros. Se recuperaría el primer registro, después el segundo, y así sucesivamente hasta que aparezca el primer registro con otro oficio. Hasta ahora, todo bien. Sin embargo, si un tiempo después necesitáramos recuperar todos los empleados asignados a la Oficina de Seattle, este esquema no serviría. Tendríamos que mantener otra copia adicional del archivo sobre el disco que estuviera físicamente organizada por la ubicación. Obviamente esto es una ineficiencia de almacenamiento y hemos considerado solamente una consulta alternativa. Imagine las dificultades que traería como consecuencia tener en cuenta todas las posibles consultas durante la construcción y creación de los archivos físicos que respondan a cada una.

Lo que tenemos aquí es un conflicto entre el ordenamiento físico de los registros y el ordenamiento lógico que necesita el usuario. El uso de los punteros resuelve este dilema con la representación de un número finito de listas lógicas, sin necesitar una reorganización de la secuencia física del archivo.

En el ejemplo de la Figura 10.12 observe que el valor en el campo PUNTERO OFICIO para el registro Steel es "0". Esta es la notación del puntero nulo, o el fin de la lista. El mismo le comunica al sistema operativo que no existen más registros en la lista eulazada para ingenieros. Por otra parte, el último registro en la lista enlazada pudiera contener

un puntero al primer registro en la lista, formando una estructura de anillo. Las estructuras de anillos son útiles en los SGBD en red, donde una entrada puede estar en una ubicación por encima del tope de la lista enlazada.

Listas invertidas

A pesar de que las listas enlazadas constituyen una vía efectiva para implementar interrelaciones lógicas, tienen sus limitaciones. Si la lista es extremadamente grande, el recorrido por la misma puede consumir tiempo. El mantenimiento de la lista puede ser engorroso, particularmente si con frecuencia existen inserciones y eliminaciones. Las listas largas también están sujetas al problema del fraccionamiento debido a un mal funcionamiento del sistema operativo.

lista invertida. Un directorio en donde cada entrada contiene un puntero a todos los registros físicos con un valor específico.

Un método alternativo para acceder a los registros según un orden lógico es mediante el uso de listas invertidas. Una **lista invertida** es un archivo independiente, o índice, que generalmente contiene solamente dos datos: un valor de interés, como por ejemplo "ingeniero", y todas las direcciones de los registros que tienen dicho valor. La Figura 10.13 muestra dos listas invertidas para los datos de la Figura 10.12. El uso de la lista invertida elimina la necesidad de utilizar los campos PUNTERO OFICIO y el PUNTERO UBICACIÓN dentro de los registros. En la práctica, las referencias a la lista invertida y al archivo se realizan sobre archivos de acceso-directo.

Recordemos que las listas enlazadas necesitan que los punteros se almacenen en los registros de datos; por el contrario, la lista invertida elimina esta necesidad, almacenando los punteros separados de los datos. Ambos métodos se usan en muchos SGBD.

Árbol-equilibrado indexado (B^+-árbol)

Un refinamiento de la estrategia de lista-invertida es el B^+-árbol. Los B^+-árbol se desarrollaron con el objetivo de proporcionar un método eficiente para el mantenimiento de una jerarquía de índices. Una pregunta que le surge a muchos cuando ven por primera vez un B^+-árbol es cómo se compara con una organización de archivos secuencial-indexada. La respuesta consiste en que el B^+-árbol es más eficiente. El rendimiento de un método secuencial-indexado se degrada a medida que el archivo crece. Por otra parte, el rendimiento puede mejorarse con la reorganización periódica del archivo, aunque dichas reorganizaciones pueden consumir mucho tiempo y ser costosas. Un B^+-árbol conserva su eficiencia a medida que el archivo crece o se reduce.

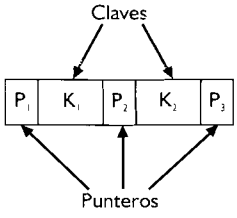
El B^+-árbol posee un indexamiento multinivel que resulta eficaz para el procesamiento de registros de datos tanto secuencial como directo. Un B^+-árbol consiste en una

OFICIO	DIRECCIONES				
Contador	1	4	7	8	
Dibujante	2	6	10		
Ingeniero	3	5	9		

(a) Lista invertida de oficios

UBICACIÓN	DIRECCIONES			
Los Angeles	2	4	5	
Portland	6	8	10	
Seattle	1	3	7	9

(b) Lista invertida de ubicaciones



jerarquía de registros índices junto con un archivo de registros de datos. Los registros índices contienen claves y punteros que se usan para localizar los registros de datos. En la Figura 10.14 se muestra un registro índice común (RI) en el B⁺-árbol. En este ejemplo tenemos tres punteros y dos claves. En general, en un IR podemos tener un máximo de n punteros y n-1 claves, donde n lo determina el diseñador del B⁺-árbol.

Con el objetivo de definir el B⁺-árbol, necesitamos definir ciertos términos. Un **árbol enraizado** es una jerarquía de RIs que tiene un único RI en el nivel más alto. Este registro se denomina **raíz**. Un RI es una hoja si el mismo se encuentra en el nivel más bajo en el árbol enraizado. (Estos árboles son al revés, la raíz arriba, las hojas abajo.) Los punteros en todos los RIs que no son hojas apuntan a otros RIs. Sin embargo, los punteros en los RIs que son hojas apuntan a los registros de datos y a otros RIs que son hojas.

Un B⁺-árbol está equilibrado si todos los RIs que son hojas se encuentran a igual distancia del RI raíz. La distancia se mide por el número de RIs que se examinan para alcanzar el RI hoja. La secuencia de RIs que se examinan se llama **camino**. Este concepto de balance garantiza que el acceso a todos los registros de la base de datos tenga el mismo grado de eficiencia. En el siguiente tema estableceremos las propiedades de un B⁺-árbol y se aclarará su uso con algunos ejemplos.

árbol enraizado. Una jerarquía de registros índices que tiene un único registro índice en el nivel más alto; este registro se llama raíz.

hoja. El registro índice en el nivel más bajo de un árbol enraizado.

camino. Un conjunto de punteros que llevan desde un registro índice hacia otro.

Formalmente, el B⁺-árbol es un árbol enraizado que satisface los requisitos siguientes:

1. Todos los caminos desde el RI raíz hasta un RI hoja son de la misma longitud.
2. Cada RI que no es una hoja contiene al menos $\lceil n/2 \rceil$ punteros y a lo sumo n punteros hacia los RIs de los niveles más bajos. ($\lceil ./. \rceil$ denota la operación aritmética de redondeo hacia el próximo valor entero. Por lo que, si n es 3, entonces $\lceil n/2 \rceil$ es 2. Si n es 4, entonces $\lceil n/2 \rceil$ es 2.)
3. Un RI hoja contiene al menos $\lceil (n\text{-}1)/2 \rceil$ y a lo sumo n-1 punteros a registros en el archivo de datos.
4. Las claves en un RI se ordenan por valor, $k_1 < k_2 \ldots < k_{n-1}$.
5. Todas las claves en el subárbol al cual apunta el puntero p_1 son estrictamente menores que k_1.
6. Para $2 <= i <= n - 1$, todas las claves en el subárbol hacia el cual apunta p_i tienen valores mayores o iguales que k_{i-1} y menores que k_i.
7. Todas las claves en el subárbol hacia el cual apunta p_n son mayores o iguales que k_{n-1}. Por otro lado, para $i > 1$, k_{i-1} siempre será la clave más baja en el subárbol referido por p_i.

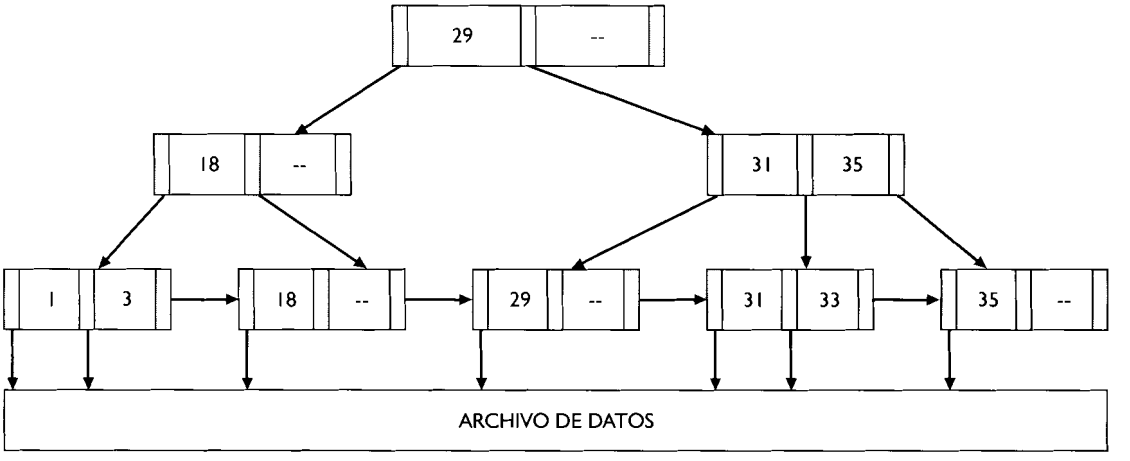
Explicaremos estas propiedades mediante un ejemplo. (Por razones de brevedad, en la siguiente discusión, nos referiremos a las anteriores siete propiedades como reglas.) El procedimiento general consiste en hallar un RI hoja donde se pueda insertar el valor de una clave, e insertarlo si hay capacidad. Si ese RI está lleno, lo dividimos en dos RIs hojas, ordenando las claves existentes más la nueva en orden ascendente, ubicamos las primeras $\lceil n/2 \rceil$ claves en el RI más a la izquierda, y el resto de las claves en el otro RI. Después de dividir un RI hoja, ajustamos las claves y los punteros en los RIs sobre el camino seguido hacia el RI que se dividió.

La Figura 10.15 muestra un ejemplo de un B⁺-árbol para $n = 3$ que indexa los registros con las siguientes claves adicionándolos en el orden:

31, 3, 18, 35, 29, 1, 33

Las líneas discontinuas en algunas posiciones indican que no existe clave. Si no existe clave, entonces tampoco hay un puntero hacia su derecha. La forma en que se utilizan los punteros en los RIs hojas se explicará brevemente.

Ahora mostraremos cómo se construyó este árbol añadiendo los registros a un árbol vacío. Refiérase a la Figura 10.16. El primer registro que se añade tiene la clave "31". Este simplemente se inserta en un RI tal como se muestra en la Figura 10.16(a). Como nuestro



árbol solamente tiene un registro, este es a su vez un RI hoja y raíz. En la Figura 10.16(b) vemos que existe espacio para indexar el registro 3 sin requerir ningún RI adicional. (Se necesita un ordenamiento dentro del RI por la regla 4.) Para mayor claridad, ilustramos el archivo de datos sólo en la Figura 10.16(a).

Las cosas cambian cuando indexamos el registro 18. No es posible insertar la clave 18 en el RI existente, ya que se encuentra lleno. Recuerde que un RI hoja puede tener a lo sumo n-1 punteros hacia registros en el archivo de datos (regla 3); en este caso, solamente se permiten dos. Por consiguiente, debemos dividir el RI existente, ordenar las claves, y por convenio ubicar las primeras $\lceil n/2 \rceil = 2$ claves ordenadas en el RI más a la izquierda. La otra clave(s) se coloca en el RI que queda. Esto se muestra en los RIs del nivel-inferior en la Figura 10.16(c).

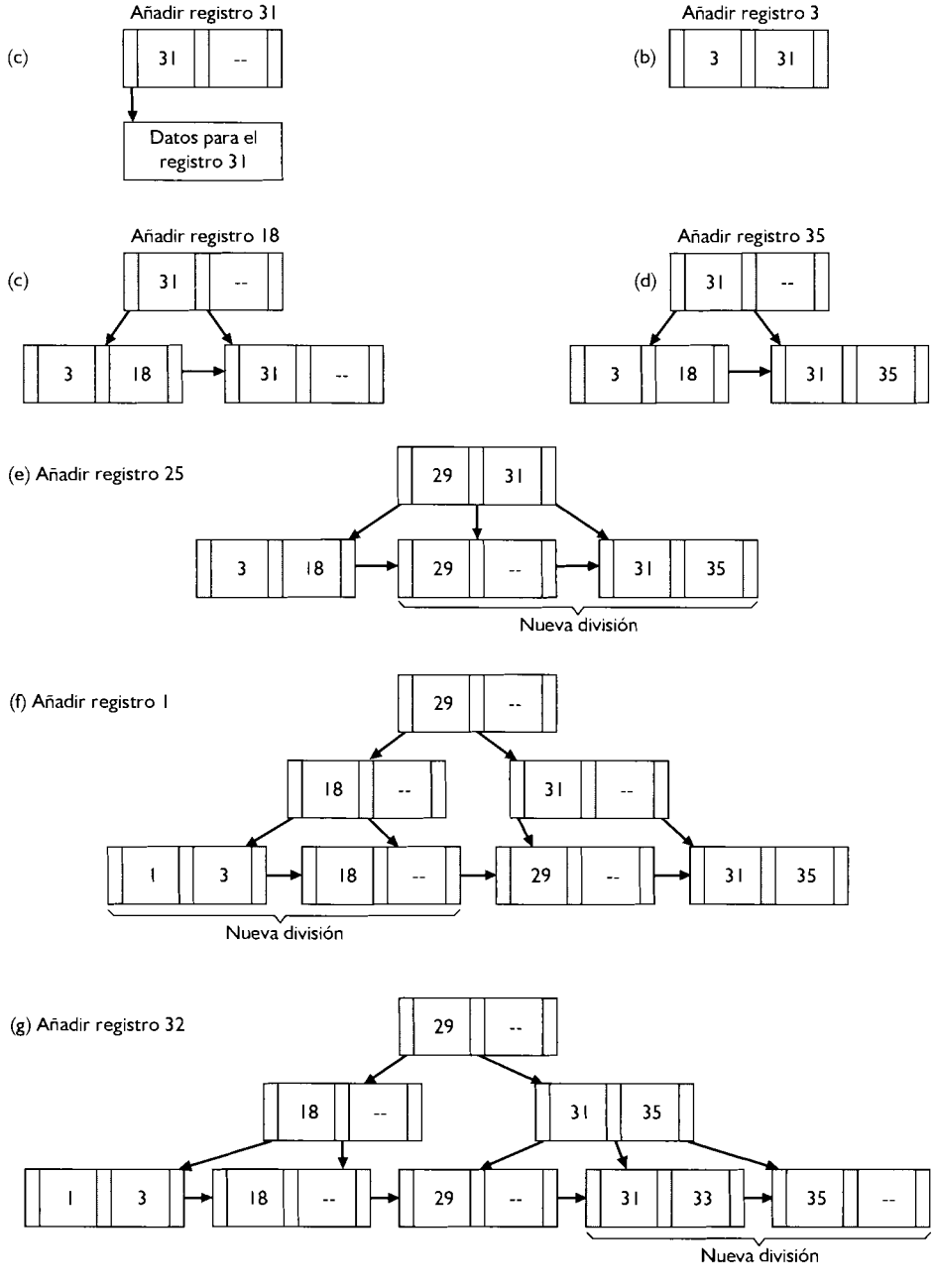
Como ahora tenemos más de un RI hoja, necesitamos añadir un RI en el nivel-más alto en el árbol para indexar los RIs hojas, tal como se muestra en el nivel superior de la Figura 10.16(c). (Por el contrario, ya no tenemos un árbol enraizado.) Este RI en el nivel-mayor es ahora la raíz.

Observe que existen $n = 3$ punteros en cada RI. En cada RI hoja, el puntero más a la izquierda (p_1) apunta a la localización donde se almacena el registro completo para la clave k_1, p_2 realiza la misma función para k_2. Sin embargo, p_3 apunta hacia el próximo nodo hoja en secuencia, por lo que el archivo puede procesarse secuencialmente cuando se necesite.

Observe también que en el RI no hoja creado recientemente, que se muestra en la Figura 10.16(c), p_1 y p_2 son los únicos punteros utilizados y ambos satisfacen las reglas (5-7). El puntero hacia la izquierda del valor de la clave 31 apunta a un subárbol cuyos valores de las claves son estrictamente menores que 31. El puntero hacia la derecha del valor de la clave 31 apunta hacia aquellas claves que son mayores o iguales que 31.

La Figura 10.16(d) muestra la adición de la clave del registro 35. Buscamos una localización en el árbol donde colocar la clave 35. En este caso seguimos el puntero derecho en el RI raíz y hallamos que la clave 35, que es mayor que 31, debe ubicarse en el nodo hoja que contiene la clave 31. Como existe un lugar vacante, simplemente entramos la clave 35, tal como se muestra. No se requieren otros cambios.

La Figura 10.16(e) muestra la adición del registro 29. Como 29 es menor que 31 seguimos el puntero izquierdo en el RI raíz. El RI hacia el cual apunta es donde se ubicaría la clave 29, pero este RI está lleno. Por lo tanto, dividimos el RI existente, ordenamos las claves correspondientes y añadimos las primeras $[n/2] = 2$ claves en el RI más a la izquierda, con el resto de la clave insertado en el otro IR, tal y como se muestra en la figura.


Añadir registro 31
(c)
31
--
Datos para el registro 31
Añadir registro 3
(b)
3
31
Añadir registro 18
(c)
31
--
3
18
31
--
Añadir registro 35
(d)
31
--
3
18
31
35
(e) Añadir registro 25
29
31
3
18
29
--
31
35
Nueva división
(f) Añadir registro 1
29
--
18
--
31
--
1
3
18
--
29
--
31
35
Nueva división
(g) Añadir registro 32
29
--
18
--
31
35
1
3
18
--
29
--
31
33
35
--
Nueva división

La Figura 10.16(g) muestra la adición de la clave 33. Se debe dividir la hoja más a la derecha. Entonces, como se muestra, colocamos las claves en el orden 31, 33, y 35. Como se observa, esto fuerza un cambio en las entradas en el RI que contiene los punteros hacia los nuevos nodos hojas (reglas 5, 6, y 7).

Veamos ahora cómo se procesa una consulta usando nuestro B^+-árbol. Supongamos que deseamos acceder al registro 33. Primero se examina el nodo raíz. Como 33 es mayor que 29, seguimos el puntero intermedio hacia el RI en el nivel próximo-inferior, donde 31 es la primera clave índice, pero menor que 33, seguimos el puntero intermedio hacia el RI hoja que contiene el puntero a los datos del registro 33. Este es el mismo proceso que seguimos cuando determinamos dónde insertar un nuevo registro.

Como observamos en la creación del B^+-árbol de la Figura 10.16(g), la inserción puede ser más complicada que un acceso a un simple registro, ya que en ocasiones es necesario dividir un RI, lo que podría ser muy largo como resultado de la inserción. ¿Qué pasa con la eliminación de registros? ¿Cómo afectaría la eliminación al B^+-árbol?

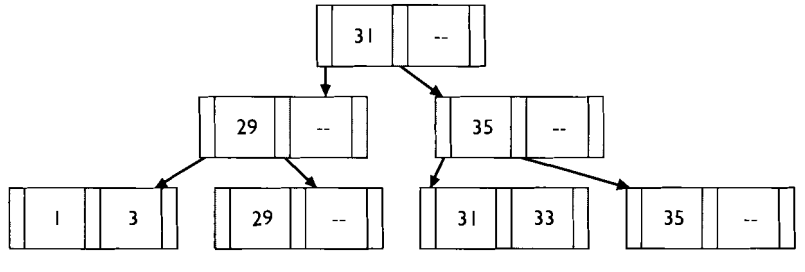
En el caso más simple, una eliminación no necesita ningún cambio en la estructura del B^+-árbol. Refiérase nuevamente a la Figura 10.16(g). Suponga que se elimina el registro 33. Usando el mismo procedimiento, como si estuviéramos accediendo al registro 33, localizamos el RI hoja donde aparece el mismo. Entonces eliminamos la clave 33 y su registro de datos correspondiente. Ahora el RI hoja donde estaba la clave 33 sólo contiene la clave 31. Recalculando los nodos hojas que pueden tener $[(n\text{-}1)/2]$ valores (= 1 en este caso). El nodo hoja se puede quedar como es. Por otro lado, el valor 33 no aparece en ningún otro lugar en el B^+-árbol, por lo que no se necesitan otros cambios.

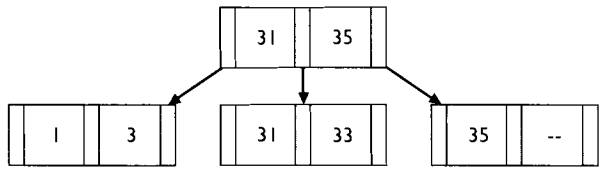
Sin embargo, al igual que las inserciones, las eliminaciones pueden causar una reestructuración del índice. El método es básicamente el inverso del que ocurre con los cambios de inserción. Examinemos un ejemplo de una eliminación que necesita una modificación en la estructura del índice.

Refiérase a la Figura 10.16(g). Suponga que deseamos eliminar el registro 18 de este árbol. Cuando se elimina el registro 18 en la hoja donde se encuentra, este RI contiene sólo valores nulos y como tal debe ser eliminado, junto con el puntero desde su RI padre (el RI en el nivel próximo-superior). Ahora el RI padre tiene un solo puntero, violando la regla 2; sin embargo, este RI contiene información útil, que no se debe eliminar. Por lo tanto, buscamos en sus RI hermanos (mismo nivel). Si algún RI hermano tiene una vacante, podríamos eliminar el RI que contiene el 18, insertar la clave 18 en el mismo y ordenar los punteros.

Como éste no es el caso, la única solución es reservar dos punteros a cada uno de los RIs y introducir las claves apropiadas para satisfacer las reglas 4-6. Esto se muestra en la Figura 10.17.

Ahora, supongamos que eliminamos el registro 29 desde la Figura 10.17. Tenemos prácticamente la misma situación anterior. Cuando se elimina la clave 29 del RI hoja, se elimina el RI, dejando al RI padre con sólo un puntero. Sin embargo, en este caso, el RI tiene un hermano vacante, y podemos integrar estos dos RIs en un RI, tal como se muestra en la Figura 10.18. Observe que al integrar estos RIs en el segundo nivel, el mismo se convierte en un nodo simple. Por lo que el nodo raíz inicial es redundante y puede eliminarse.





Consideremos un ejemplo más acerca de la construcción de un B⁺-árbol, sólo por esta vez, nsaremos $n = 4$ y el conjunto de claves de registros siguiente:

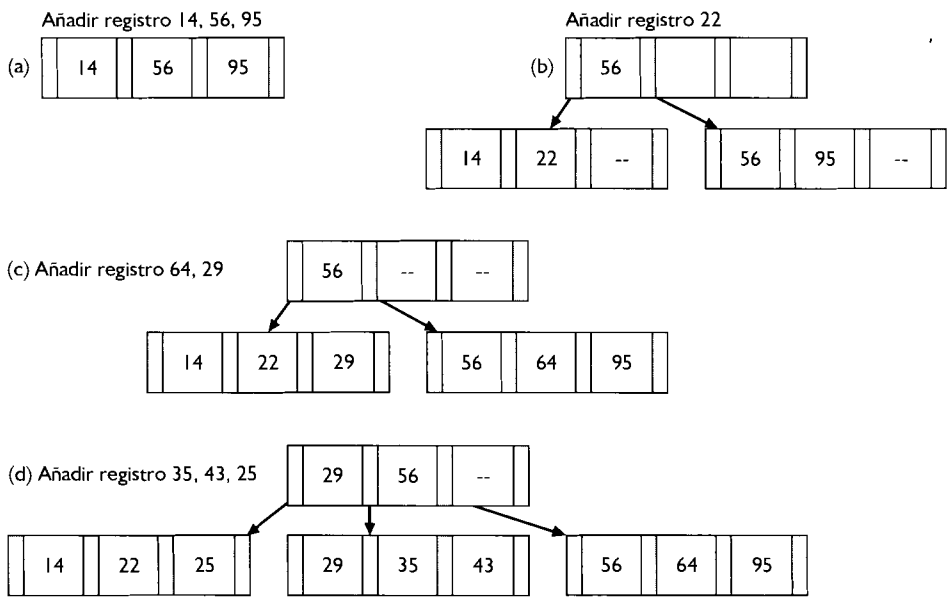
14, 95, 56, 22, 64, 29, 35, 43, 25

Como se muestra en la Figura 10.19(a), llenamos el RI inicial con las claves de los registros 14, 95, 56. Cuando añadimos el registro 22, debemos dividir este nodo, ordenar el conjunto de claves, añadir las primeras $[n/2]$ (=2) claves en el primer RI y el resto de las claves (dos en este caso) en el segundo RI. Debido a esto se crea un nodo raíz con la clave 56 (Figura 10.19(b)).

Ahora se insertan los registros 64 y 29 sin cambiar la configuración del árbol (Figura 10.19(c)). Si adicionamos el registro 35 se divide el nodo más a la izquierda, con la consiguiente ordenación e inserción de las cuatros claves de registros correspondientes de acuerdo con las reglas previas. Entonces se pueden insertar los registros 43 y 25 sin cambios en la configuración del árbol. Este resultado se muestra en la Figura 10.19(d).

El lector no tiene duda de que acercándonos a valores de n grandes se necesitan menos divisiones para un determinado conjunto de registros, pero como media aparece más espacio vacante en el B⁺-árbol.

El B⁺-árbol es la estrategia más ampliamente usada para mantener la eficiencia con respecto a las inserciones y eliminaciones. Existen conceptos similares para un método indexado llamado un *B-árbol* (sin el "+"). Un B-árbol permite la ocurrencia de solamente



una clave en el árbol, esto reduce los requisitos de espacio, pero complica el mantenimiento de los RI. Consecuentemente, el B-árbol no considera la utilidad del concepto nivel del B^+-árbol.

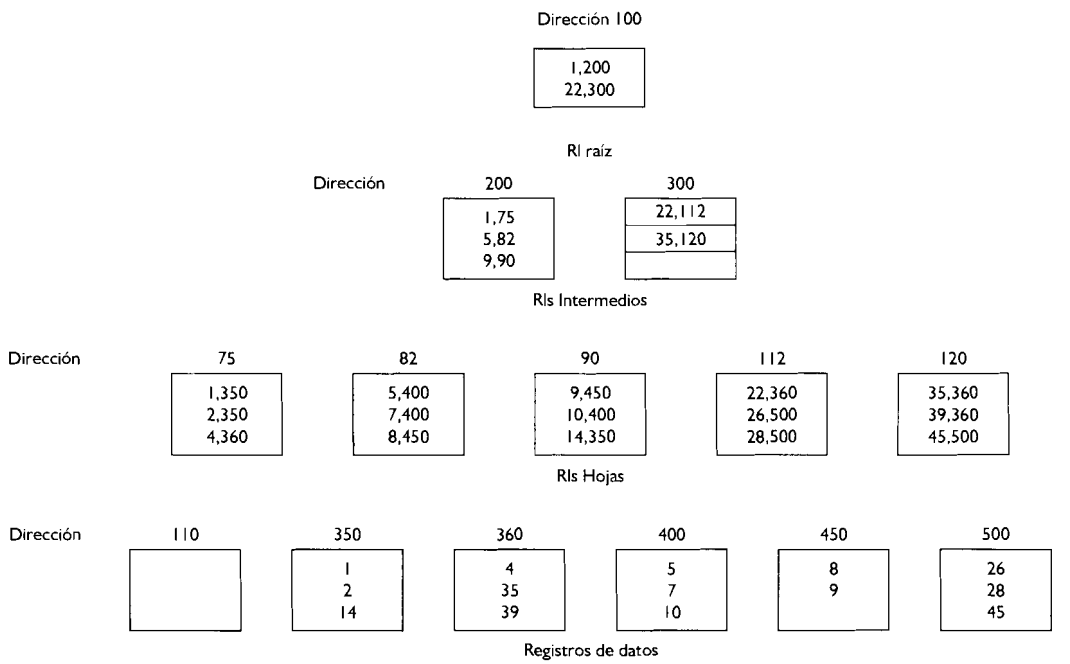
▼ Implementación cliente/servidor de la indexación árbol-balanceado (árbol-equilibrado)

El uso de los índices de un árbol-balanceado contiene altos niveles de rendimiento en un sistema de base de datos cliente/servidor. Aunque en algunos casos los mecanismos varían un poco, los objetivos y métodos básicos permanecen consistentes con nuestra discusión anterior.

Consideremos el ejemplo que se muestra en la Figura 10.20. La arquitectura es muy similar a la analizada en la sección previa, con algunos ajustes necesarios que analizaremos más adelante. La modificación más importante son las *páginas*, que son bloques de registros. Observe en la Figura 10.20 que la página raíz está en la dirección 100 y contiene dos registros índices. El registro índice superior indica que la página que *comienza con* el registro de datos 1, se encuentra por el acceso siguiente al IR de la dirección 200. Similarmente, la página que comienza con el registro 22 se encuentra siguiendo el próximo acceso al RI en la dirección 300.

Supongamos que queremos acceder al registro 10. Una búsqueda en la página raíz indica que el registro 10 se encuentra en el próximo acceso a la página índice localizada en la dirección 200 (ya que 10 es menor que 22). Al buscar en esta página el registro 10 se encuentra en la página índice localizada en la dirección 90 (ya que 10 es mayor que 9). Al buscar en esta página (una hoja) encontramos que el registro de datos 10 se encuentra en la página localizada en la dirección 400.

Un mecanismo similar se aplica en la inserción y eliminación de registros.



▼ Correspondencia entre estructuras de datos lógicas y estructuras de datos físicas

Ahora buscaremos la aplicación de estas ideas en un sistema de base de datos. En esta sección examinaremos algunos de los aspectos físicos de la implementación de las estructuras de datos utilizadas en las bases de datos relacionales, en red y jerárquicas.

Correspondencia en las bases de datos relacionales

Algunos sistemas de bases de datos almacenan cada relación como un archivo con un registro para cada tupla (fila). Normalmente esto es correcto para bases de datos pequeñas, pero a medida que la base de datos crece en tamaño, sería más eficiente almacenar varias relaciones en un solo archivo.

Considere las relaciones que se muestran en la Figura 10.21(a) y una consulta que para cada venta necesite el nombre del cliente, el vendedor, la cantidad y el número del producto. Para cada tupla de la relación VENTA se buscan las tuplas en la relación CLIENTE que coincidan con el valor del número del cliente. Si el CLIENTE y las VENTAS se almacenan en archivos diferentes, entonces se necesita una operación de lectura sobre el archivo por cada registro en la consulta. En la Figura 10.21(b) se sugiere un método más eficaz. Esta estructura de archivo mezcla las tuplas de ambas relaciones con el objetivo de facilitar una reunión (*join*) de ambas. Las tuplas de VENTAS para cada número de cliente se almacenan junto con las tuplas de CLIENTE con el mismo valor del número del cliente. Cuando se lee

CLIENTE #	NOMBRE	LÍMITE DE CRÉDITO	BALANCE DE DEUDAS
251	H. Barlow	$ 5.000	$1.250
095	J. Krupke	$10.000	$5.000
312	O. Reed	$ 5.000	$ 800
419	M. Little	$15.000	$7.500

VENTAS

INV. #	CLIENTE #	AMT	VENDEDOR	PROD #
12	095	120	V. Blab	W851
13	312	592	J. Wells	0912
15	312	750	V. Blab	W851
18	251	157	J. Wells	R950

251	H. Barlow	5000	1250	
12	0.95	120	V. Blab	W851
13	312	592	J. Wells	0912
095	J. Krupke	10000	5000	
312	O. Reed	5000	800	
12	312	750	V. Blab	W851
18	251	157	J. Wells	R950

una tupla de la relación CLIENTE, el bloque completo que contiene la tupla se copia en la memoria primaria. Como las correspondientes tuplas de VENTAS se almacenan juntas, el bloque que contiene las tuplas de VENTAS necesarias para completar la consulta puede estar en el mismo bloque, que se encuentra en la memoria primaria. Aún si se necesitaran otras tuplas, se almacenarían en bloques que están cercanos en el disco.

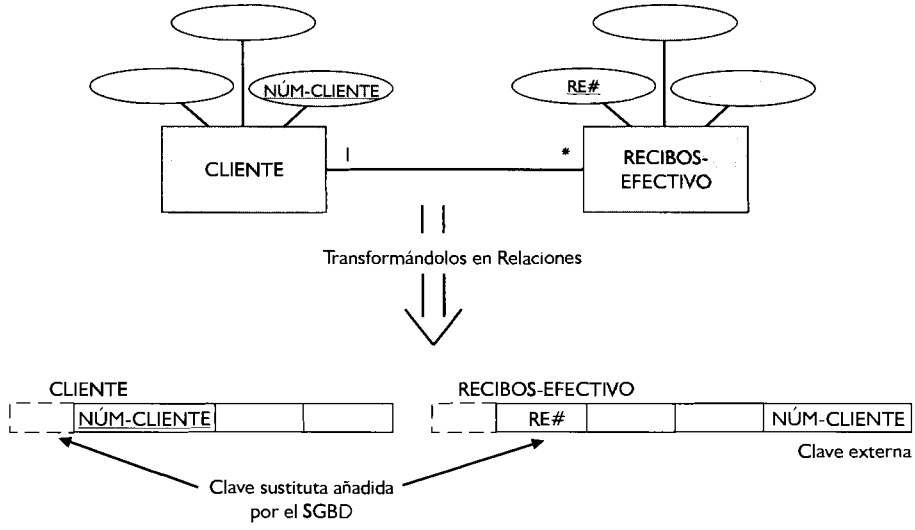
agrupamiento (*clustering*). Ubicación en un mismo bloque de relaciones que se reúnen con frecuencia.

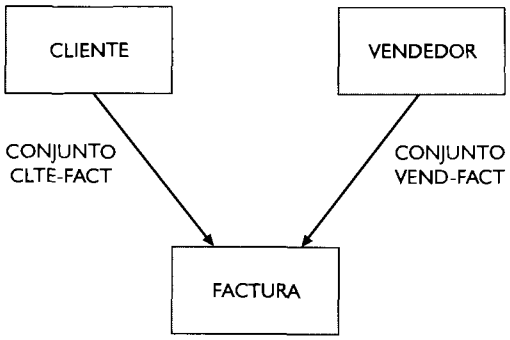
Esta estrategia de **agrupamiento** puede acelerar la reunión (*join*) de las relaciones CLIENTE y VENTAS, pero pudiera volverse más lento el proceso que involucra otras relaciones. La determinación de cuándo el agrupamiento resulta útil está en función de la frecuencia con la cual se realice una consulta.

Otro factor también se vuelve importante en la correspondencia de las relaciones con un almacenamiento físico. Ver Figura 10.22. Aquí la relación CLIENTE se encuentra relacionada con la relación RECIBOS-EFECTIVO con una cardinalidad de uno-muchos. La clave primaria para CLIENTE es NÚM-CLIENTE, y pudieran existir otros atributos adicionales. Como se muestra, NÚM-CLIENTE aparece como una clave externa en la relación RECIBOS-EFECTIVO. Si NÚM-CLIENTE se usa como la clave actual para la implementación física, podrían surgir dificultades si, por ejemplo, el NÚM-CLIENTE es erróneo o cambia. En este caso, tienen que cambiar todas las localizaciones físicas donde aparece el NÚM-CLIENTE. Esto no sólo implica tiempo y esfuerzo, sino que también es una oportunidad para que ocurran errores. Para eliminar este problema, en ocasiones los SGDBs crean una clave sustituta, solamente conocida por el propio sistema. De esta forma, si la clave primaria real, NÚM-CLIENTE, debe cambiar, la clave sustituta no se afecta y así no existe la posibilidad de que se alteren las representaciones físicas.

Correspondencia en las bases de datos en red

Cualquier interrelación en red compleja (muchos-muchos) se puede reformular como una red simple mediante el uso de registros de intersección. Por lo que nuestra discusión sobre la correspondencia necesita centrarse sólo en la red simple. Considere el ejemplo que se muestra en la Figura 10.23. Es una red simple, ya que las interrelaciones entre los registros de tipo dueño y miembro son uno-muchos. El registro de tipo factura es miembro de dos conjuntos: CLTE-FACT y VENTAS-FACT. Es decir que, cada registro FACTURA tiene un registro dueño CLIENTE y un registro dueño VENDEDOR.

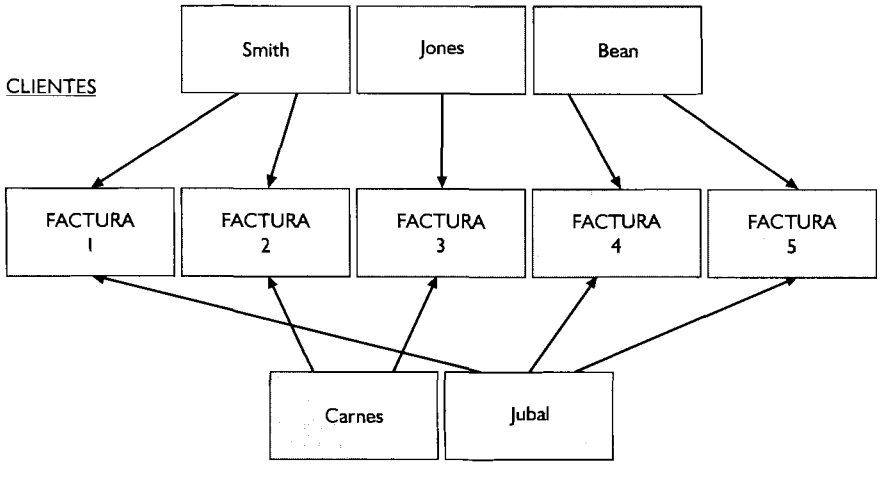




Esta simple red se puede representar mediante listas enlazadas a través de la creación de punteros por cada conjunto. Se necesita un grupo de punteros para conectar los registros de CLIENTE con sus registros en FACTURA y otro grupo de punteros para conectar los registros de VENDEDOR con sus registros en FACTURA. En la Figura 10.24 se muestra un ejemplo de la relación lógica. Sin embargo, como método físico de instrumentación es torpe porque el número de punteros requeridos es variable: Si hay tres registros de factura de un usuario, se necesitan tres punteros; si hay cinco registros de factura de un usuario, entonces se necesitan cinco punteros. De forma práctica, es difícil mantener en un registro un número variable de campos punteros. Sin embargo, la lista enlazada proporciona una forma de evitar esa dificultad.

En la Figura 10.25 se ilustra la forma en que esto se hace. Cada registro cliente contiene un puntero al número de registro de la primera factura que posee. Ese registro de factura contendrá un puntero a la próxima factura que posea ese mismo cliente. Este proceso continúa hasta que se enlaza la última factura de este cliente. Por ejemplo, el registro del CLIENTE Smith apunta a Factura #1 (número de registro 6), que apunta a Factura #2 (número de registro 7), que es la última de las facturas de Smith. El final de la lista de facturas se indica por un número de registro 0 en el campo puntero.

En la Figura 10.26 se muestra un método más práctico de implementación. Comparando la Figura 10.26 con la Figura 10.25, se puede ver que el único cambio es que el últi-



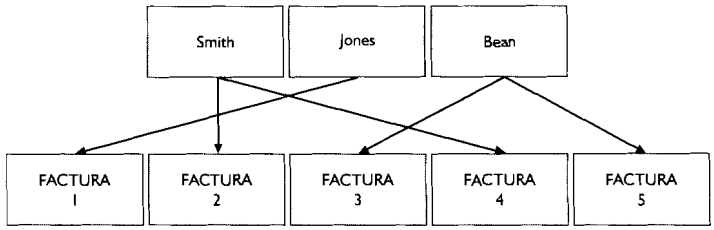
	NÚMERO RELATIVO DEL REGISTRO	REGISTRO DE DATO	PUNTERO CLIENTE FACTURA	PUNTERO VENDEDOR FACTURA
REGISTROS CLIENTE	1	Smith	6	
	2	Jones	8	
	3	Beans	9	
REGISTROS VENDEDORES	4	Carnes		7
	5	Jubal		6
REGISTROS FACTURAS	6	Fact #1	7	9
	7	Fact #2	0	8
	8	Fact #3	0	0
	9	Fact #4	10	10
	10	Fact #5	0	0

	DIRECCIÓN RELATIVA	REGISTRO DE DATO	PUNTERO CLIENTE FACTURA	PUNTERO VENDEDOR FACTURA
REGISTROS CLIENTE	1	Smith	6	
	2	Jones	8	
	3	Beans	9	
REGISTROS VENDEDORES	4	Carnes		7
	5	Jubal		6
REGISTROS FACTURAS	6	Fact #1	7	9
	7	Fact #2	1	8
	8	Fact #3	2	4
	9	Fact #4	10	10
	10	Fact #5	3	5

mo registro en cada lista contiene un puntero al registro dueño de la lista. Esta estructura circular facilita la ejecución de las consultas FIND OWNER (ENCUENTRA DUEÑO), FIND FIRST (ENCUENTRA PRIMERO) y FIND NEXT (ENCUENTRA PROXIMO) de los lenguajes comunes de manipulación de datos en forma de red. Siguiendo el círculo de punteros simplifica la localización del registro deseado. (Los lenguajes de manipulación de datos en red se estudian en el Capítulo 15.)

Correspondencia en las bases de datos jerárquicas

También, como en las estructuras de datos en red, se pueden usar múltiples punteros en los registros padres en las relaciones de una jerarquía o árbol, tal como se sugiere en la Figura 10.27. Mientras el método de múltiples punteros dentro de un registro facilita las conexiones entre un registro padre y uno hijo, éste no se usa frecuentemente por el número variable



de punteros que pueden ser necesarios en cada registro padre. Los registros de longitud variable son particularmente complicados cuando se deben realizar inserciones y eliminaciones.

El uso de punteros hijos y hermanos reduce la complejidad del mantenimiento de punteros múltiples y la restricción de la correspondencia hijo-padre. Examine la Figura 10.28, aquí ve que el primer puntero de cada registro contiene la dirección de un registro hijo, y el segundo puntero indica la locación de un registro hermano. De esta manera, cada registro contiene exactamente dos campos punteros, que es mucho más fácil de mantener. Como con cualquier lista enlazada, los punteros pueden eliminarse del mismo registro y colocarse en un directorio lista invertida. Para nuestro ejemplo, esto se muestra en la Figura 10.29.

▼ Acceso por la clave secundaria

Hasta este punto, hemos estado usando la palabra *clave*, más bien libremente, para nombrar a un elemento de dato cuyo valor identifica únicamente a un registro. Técnicamente, esta definición es para la **clave primaria**. Existen también claves secundarias que pueden ser únicas, pero no obligatoriamente. Una **clave secundaria** es la que identifica un *conjunto* de registros que tienen el mismo valor para la clave secundaria. Las claves secundarias juegan un papel importante para dar soporte a los requisitos de información de los usuarios de los SGBS.

clave primaria. Un elemento de dato cuyo valor identifica únicamente a un registro.

clave secundaria. Un elemento de dato cuyo valor identifica a un conjunto de registros.

Ilustramos los conceptos fundamentales mediante un ejemplo. La Unión de Crédito Cosmos tiene varios tipos de usuarios de su sistema de base de datos. Los cajeros necesitan acceder a los registros de las cuentas de los clientes para responder las preguntas de ellos y actualizar los registros cuando hay transacciones. Algunos clientes pueden tener sólo cuentas corrientes, otros, además, cuentas de ahorro, cuentas de tarjeta de crédito, o préstamos a largo plazo.

ID REGISTRO CLIENTE	PUNTERO AL REGISTRO FACTURA
Smith	5
Smith	7
Jones	4
Bean	6
Bean	8

DIRECCIÓN RELATIVA	REGISTRO PADRE	PUNTERO HIJO	PUNTERO HERMANO
1	Smith	5	2
2	Jones	4	3
3	Beans	6	0
4	Fact #1	0	0
5	Fact #2	0	7
6	Fact #3	0	8
7	Fact #4	0	0
8	Fact #5	0	0

0 = Fin de la Lista Invertida

El cajero puede probablemente llegar a acceder todos los registros requeridos, mediante el uso de una clave primaria, como, por ejemplo, el número de identificación del cliente. Un empleado de préstamos, sin embargo, puede a veces necesitar obtener aquellos registros que tengan un cierto límite de cuenta de crédito (digamos $2,000), o aquellos préstamos significativos que tienen balances de más de $50,000. El gerente de la división de créditos puede estar interesado en conocer el total de préstamos sin pagar de un cierto tipo: coche, reparaciones de la casa, personales, etc. Todas estas necesidades pueden ser resueltas mediante el uso de los métodos de acceso por las claves secundarias.

El acceso por la clave secundaria se logra estableciendo índices que recorran caminos deseados a través de los registros de datos físicos. Considere la siguiente consulta:

Identifica nombres de clientes que tengan préstamos de cuarenta y ocho meses.

Usando la Figura 10.30 como un simple ejemplo, mostramos cómo esta información sería proporcionada a través del uso de listas enlazadas. La clave primaria es CLIENTE-NOMBRE, y la clave secundaria es PRÉSTAMO-PERÍODO. Hay tres períodos de financiamiento para los préstamos: veinticuatro meses, treinta y seis meses, y cuarenta y ocho meses. Luego, la clave secundaria permite la obtención de conjuntos de registros asociados con cada período de financiamiento. Se requieren entonces tres punteros para identificar el registro de comienzo en cada conjunto (la cabeza de la lista). Estos punteros se almacenan separadamente de los registros de datos.

Para calcular la respuesta a la consulta, se accede a la lista de punteros cabezas, y se encuentra la dirección 1 como cabeza de la lista de registros que tienen como valor de período de financiamiento 48 . El registro localizado en la dirección 1 contiene un puntero al próximo registro que tiene ese valor, y así hasta que se encuentra todo el conjunto.

El lector perspicaz habrá notado que el acceso por la clave secundaria se usó antes en este capítulo cuando presentamos la idea de listas enlazadas. Recordará que en aquel caso estábamos utilizando las claves secundarias OFICIO y UBICACIÓN.

Las listas enlazadas de claves secundarias son particularmente útiles cuando los archivos físicos son muy grandes. Si en una versión extendida del archivo usado en la Figura 10.30 tuviéramos 50.000 registros de los cuales sólo 100 tuvieran períodos de financiamiento de cuarenta y ocho meses, el uso de la clave secundaria Préstamo-Período nos ahorraría la búsqueda por todo el archivo de 50.000, un factor de (50000/100 =) 500.

Como habrá supuesto, siempre que podamos representar listas con punteros dentro de los registros, podemos también tener la alternativa de eliminar los punteros y mantenerlos en una lista invertida. La Figura 10.31 muestra la lista invertida del ejemplo de la Figura 10.30.

LISTA CABECERA

24 = 3
36 = 2
48 = 1

DIRECCIÓN RELATIVA	NOMBRE-CLIENTE	TIPO PRÉSTAMO	PERÍODO-PRÉSTAMO	PUNTERO PERÍODO PRÉSTAMO
1	Nixon	Auto	48	6
2	Patton	Auto	36	4
3	Fortius	Embarcación	24	0
4	Wood	Auto	36	5
5	Cayman	Casa	36	7
6	Vance	Auto	48	8
7	Costas	Casa	36	0
8	Ubu	Auto	48	0

PERÍODO-PRÉSTAMO

24	3			
36	2	4	5	7
48	1	6	8	

▼ Optimización de las consultas

La optimización de las consultas es una consideración importante en un sistema de base de datos, ya que la diferencia en tiempo de ejecución entre una buena y una mala estrategia puede ser sustancial. Con las estructuras en red y jerárquicas, la optimización se le deja al programador, ya que las órdenes de manipulación de los datos están contenidas en un lenguaje de programación y están al nivel de la manipulación individual de los registros. Las consultas relacionales, sin embargo, pueden expresarse completamente en el lenguaje de consulta relacional y manipular de una vez conjuntos enteros de registros. Es, por lo tanto, posible y deseable optimizar de forma automática las consultas.

Nuestro propósito en esta sección es el de familiarizar al lector con las ideas fundamentales de la optimización de las consultas sin explorar todos los detalles de este enorme campo de estudio. Nos concentramos en la cuestión general de cómo producir eficazmente una respuesta a una consulta en álgebra relacional. Veremos estos tres operadores:

Select (Selección)
Project (Proyectar)
Join (Reunión)

Usaremos como ejemplo la siguiente base de datos:

```
EMPLEADO(EMP-NOMBRE, CALLE, EMP-CIUDAD)
TAREA(PROY-NOMBRE, EMP-NOMBRE, OFICIO, HORAS)
PROYECTO(PROY-NOMBRE, PRESUPUESTO, PROY-CIUDAD)
```

Combinar las operaciones de selección y reunión

Considere la siguiente consulta:

Encontrar los presupuestos y los nombres de todos los proyectos cuyos empleados vivan en Pasadena.

Una forma de escribir e interpretar esta consulta es como sigue:

```
A := Join(EMPLEADO, TAREA, PROYECTO)
B := Select(A: EMP-CIUDAD = "Pasadena")
C := B[PROY-NOMBRE, PRESUPUESTO]
```

Como Reunión opera sobre las tres relaciones en su totalidad, tienen que examinarse todas las tuplas en las tres relaciones. Luego, si hay n tuplas en EMPLEADO, m tuplas en TAREA, y k tuplas en PROYECTO, entonces hay que hacer $n \times m \times k$ inspecciones.

Compare lo anterior con la siguiente estrategia.

```
A := Select(EMPLEADO: Ciudad = "Pasadena")
B := Join(A, TAREA, PROYECTO)
C := B[PROY-NOMBRE, PRESUPUESTO]
```

Suponga que existen t empleados que viven en Pasadena ($t <= n$). Entonces se deben realizar $t \times m \times k$ inspecciones, donde

$$(t \times m \times k) <= (n \times m \times k)$$

Si t es considerablemente menor que n, esta última estrategia necesitaría considerar menos inspecciones. Por consiguiente, una ejecución eficiente realizaría las selecciones lo antes posible.

Combinar las operaciones proyección, selección y reunión

En el ejemplo anterior, todavía la segunda formulación de la consulta obtuvo información innecesaria con sus operaciones. La relación deseada necesita información sólo para los valores de dos atributos. PROY-NOMBRE y PRESUPUESTO. Todo lo que en realidad se necesita son los atributos necesarios para la consulta deseada y los atributos que intervienen en la reunión. Usando de forma correcta el operador de proyección se obtiene una consulta aún más eficiente, como sigue:

```
A := Select(EMPLEADO: EMP-CIUDAD = "Pasadena") [EMP-NOMBRE]
B := ASSIGNMENT[PROY-NOMBRE, EMP-NOMBRE]
C := PROJECT[PROY-NOMBRE, PRESUPUESTO]
D := Join(A, B, C) [PROY-NOMBRE, PRESUPUESTO]
```

En cada paso de esta solución hemos proyectado hacia afuera todos los atributos innecesarios, dejando sólo los necesarios para la solución. Entonces la estrategia consiste en aplicar las selecciones y proyecciones tan pronto como sea posible. La operación de reunión lleva mucho tiempo, por lo que se debe realizar lo más tarde posible.

En este capítulo se introdujeron conceptos básicos de organización y acceso a los archivos, que son de gran valor para los usuarios y los diseñadores de sistemas. Ambos grupos se familiarizaron con la terminología y los conceptos básicos de organización física y acceso a los archivos con el objetivo de comunicarse mejor con el personal técnico, preguntar cuestiones relevantes a los vendedores, tener conocimiento de alternativas, y por otro lado estar disponible para contribuir a una implementación efectiva del sistema de base de datos.

Primero estudiamos los medios de almacenamiento físicos que soportan las operaciones de los sistemas de base de datos. Examinamos la forma en que se almacenan los datos en el disco y definimos el proceso de acceso a los mismos. Más tarde se mostró cómo calcular el tiempo de acceso.

Se definieron tres métodos básicos de organización física de archivos: secuencial, secuencial indexada, y directa. La organización secuencial es eficiente cuando las aplicaciones solamente comprenden el procesamiento de un número notable de registros cada vez que se accede al archivo. La organización secuencial-indexada es efectiva cuando existen aplicaciones notables que necesitan el procesamiento secuencial y otras que necesitan el procesamiento directo. La organización directa es necesaria cuando aplicaciones más importantes necesitan el acceso directo a los registros. Ilustramos ambas formas de la organización directa de los archivos, estática y dinámica.

También examinamos las estructuras de datos físicas fundamentales que permiten la implementación de interrelaciones de datos lógicos. La herramienta más elemental es el

puntero. El puntero es un elemento de dato que contiene la dirección física de un registro almacenado. Los punteros deben estar incrustados dentro de los registros, para encadenarse juntos en una lista de registros relacionados. Alternativamente, un índice de valores, junto con los punteros a los registros que contienen dichos valores, se puede mantener separado de los registros. Esto se llama una lista invertida. Un esquema de indexación que mantiene su eficacia sin importarle las inserciones y eliminaciones se llama B^+-árbol. Se presentaron los conceptos del mismo y se mostró su utilización y cómo se construyen.

Se dio una idea acerca de los métodos mediante los cuales los modelos de datos lógicos se corresponden con las representaciones físicas. De esta forma, completamos una visión desde el modelo conceptual hacia el modelo de implementación lógica y de éste a la implementación física.

Después mostramos cómo se usan los punteros con respecto a las claves secundarias para facilitar la recuperación de datos.

Finalmente, estudiamos brevemente el tema de la optimización de consultas. A pesar de que un buen SGBDs relacional brinda la optimización necesaria, el conocimiento de causa y método puede darle ideas al usuario del sistema de base de datos.

1. Defina con sus propias palabras cada uno de los siguientes términos:
   a. selector de estrategia
   b. administrador de archivo
   c. memoria principal
   d. cilindro
   e. registro físico
   f. razón de transferencia de datos
   g. formato cuenta-clave
   h. lista cabecera
   i. lista invertida
   j. árbol enraizado
   k. hoja
   l. retraso de rotación
   m. controlador de disco
   n. agrupamiento (clustering)
   o. clave primaria
2. ¿Por qué la reservación de registros en bloque constituye un factor en el rendimiento del sistema de base de datos?
3. Compare la organización de archivos secuencial, secuencial-indexada y directa.
4. ¿Cuáles son las características que debe cumplir un buen algoritmo hash?
5. ¿Cuál es la función del intervalo en el formato de un registro?
6. Diferencias entre el formato cuenta-dato y cuenta-clave.
7. ¿Qué es latencia?
8. Describa las operaciones principales que se dan en la gestión de entrada/salida.
9. ¿Qué eficacia se complementa con el uso de los punteros?
10. Diferencias entre una lista enlazada simple y un anillo.
11. ¿Qué es un puntero nulo y cuál es su significado?

12. ¿Qué significa el término B^+-árbol? ¿Cuál es su propósito?

13. Diferencias entre una clave primaria y una secundaria.

1. Marque cada término con su selección.

—*administrador de buffer*	a. Un elemento de dato que contiene una dirección física.
—*diccionario de datos*	b. El tiempo necesario para posicionar las cabezas de lectura/escritura sobre un determinado cilindro.
—*tiempo de posicionamiento*	c. El tiempo necesario para activar las cabezas de lectura/escritura.
—*tiempo de activación de la cabeza*	d. Defina la estructura de los datos del usuario y cómo se usan.
—*puntero*	e. Software que controla el movimiento de los datos entre la memoria principal y el almacenamiento en disco.
—*lista enlazada*	f. Una secuencia de punteros que conectan registros indexados.
—*camino*	g. Registros físicos que se enlazan mediante punteros incrustados.
—*clave secundaria*	h. Un valor de un elemento de dato que identifica un conjunto de registros.

2. Suponga que tenemos almacenados registros en un dispositivo de disco que posee las siguientes características:

   tiempo de posicionamiento medio: 0,02 segundos
   velocidad de rotación del disco: 3.600 revoluciones por minuto
   velocidad de transferencia de datos: 312.000 bytes por segundo

   ¿Cuál es el tiempo de transferencia de datos esperado para un registro físico accedido aleatoriamente que ocupa 500 bytes?

3. ¿Cómo respondería la pregunta (2) suponiendo que se use un dispositivo de disco con las cabezas de lectura/escritura fijas, es decir, cada pista en cada cilindro tiene su propia cabeza de lectura/escritura?

4. Usando los mismos parámetros dados en (2), suponga que se guardan 10 registros físicos en una pista. ¿Cuáles serían los tiempos de transferencia de datos comparativos para: (a) 30 registros almacenados secuencialmente en el mismo cilindro, y (b) 30 registros almacenados en tres pistas que no se encuentran en el mismo cilindro?

5. Si las facilidades de compactación mejoran el almacenamiento y la recuperación de registros, ¿por qué no se almacenan archivos en un gran bloque?

6. Suponga que se almacenan registros lógicos en bloques de cuatro y que dos bloques sucesivos contienen los registros 11, 13, 14, 19, y 21, 23, 24, 26. Describa cómo el sistema operativo localizaría el registro 23.

7. ¿Por qué la organización de archivos secuencial sería eficiente para el procesamiento de las nóminas semanales? ¿Por qué no sería eficiente para responder las demandas de los usuarios?

8. ¿Por qué necesitamos otros métodos de organización de archivos, a pesar de que la organización secuencial-indexada posee ambos métodos de accesos a registro, secuencial y directo?

9. Use el algoritmo del cociente cuadrático de hash para calcular la dirección relativa en la que se almacenan los siguientes registros. Suponga un factor de carga de un 80 por 100. ¿Cuántos bloques se necesitan si éste es el archivo completo? (Asuma un registro por bloque.)

Clave	*Nombre*
14	Smith
24	Bean
28	Harris
23	Scott

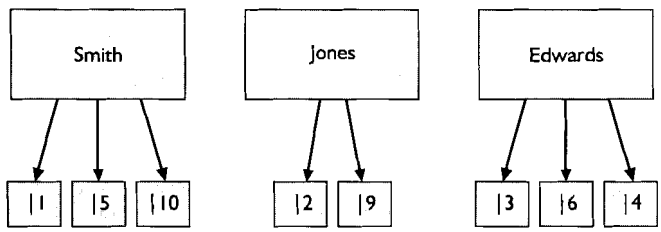
10. Suponga que un algoritmo hash dinámico obtiene los siguientes bits de mayor-orden para las claves de determinados registros. Muestre cómo estos registros se indexan y almacenan. Siga el ejemplo mostrado en el texto, y suponga que se insertan en el mismo orden seguido en el ejemplo:

$h(k_1) = 0011...$
$h(k_2) = 1100...$
$h(k_3) = 1001...$
$h(k_4) = 1110...$
$h(k_5) = 1011...$
$h(k_6) = 0010...$
$h(k_7) = 1010...$

11. ¿Cuál considera más fácil mantener: una lista invertida o una lista enlazada simple? Si su respuesta contiene "esto depende", ejemplifíquela.

12. Construya un B^+-árbol con los siguientes registros: 20, 63, 34, 56, 43, 89, 45, 68, 52, 54, 14, 19, 7, 70 y 82. Con $n = 3$.

13. Muestre cómo se representaría la red de la Figura 10.1E si:
    a. Usando punteros hijos-y-hermanos
    b. Usando un índice con punteros de dirección
    c. Una estructura de anillo



14. Constrnya un diagrama de un modelo de datos jerárquico y muestre cómo se correspondería con una representación física.

15. Construya un diagrama de una red simple y muestre cómo se correspondería con una representación física.

16. Construya un diagrama de una red compleja y muestre cómo se correspondería con una representación física.

17. Construya dos relaciones para las que sea necesario hacer una reuuión para satisfacer una demanda del usuario. Muestre cómo se usa el agrupamiento para hacer más eficiente la formulación de la demanda.

18. Construya un ejemplo de un archivo de persoual ordenado por el número de empleado e incluya listas enlazadas para brindar claves secundarias sobre el plan de pagos (existen tres tipos: *A*, *B*, y *C*), Tipo de empleado (por horas o salario) y el plan de jubilación (*X* o *Y*).

19. Repita el problema 17 usando una lista invertida.

20. Cousidere la siguiente base de datos:

```
CLIENTE(CLTE#, NOMBRE, CLTECIUDAD)
CUENTA(CTA#, TIPO, CLTE#, BALANCE, OFICINA#)
SUCURSAL(OFICINA#, FONDOS, CIUDAD)
```

Escriba las siguientes peticiones de la forma más eficieute:

a. Acceso y búsqueda de los fondos y del número de la oficina de todas las sucursales que tienen clientes viviendo en Midway.

```
A:=REUNIÓN(CLIENTE, CUENTA, UBICACIÓN)
B:=SELECT(A:CLTECIUDAD = "Midway") [OFICINA#, FONDOS]
```

b. Acceso y búsqueda de los fondos y del número de la oficina de todas las sucursales que tienen clientes viviendo en Midway con saldos de depósitos mayor que $1.500.

```
A:=REUNIÓN(CLIENTE, CUENTA, UBICACIÓN)
B:=SELECT(A:CLTECIUDAD = "Midway" and BALANCE >
    1,500) [OFICINA#, FONDOS]
```

1. Para cada uno de los siguientes ejemplos, escoja cuál sería el método de organización de archivo más apropiado:
   a. Un sistema de base de datos de un hospital que soporta sus operaciones de negocio
   b. Un sistema de pedidos para una firma de manufactura
   c. Una agencia de alquiler de coches
   d. Un distribuidor de productos farmacéuticos
   e. Un sistema de registros de estudiantes en una universidad
   f. Un sistema de reserva de habitaciones en un hotel

2. Ya que las estructuras de datos físicas no están determinadas ni por el analista del sistema ni por el usuario, ¿existe alguna ventaja en formarlos en este tema?

3. Un efectivo agrupamiento de los registros para facilitar las operaciones del lenguaje relacional puede depender del tipo y la frecuencia con que se realicen distintos tipos de consultas. Piense en un camino que nos pueda guiar en la determinación de cómo agrupar los registros.

4. Si tiene acceso a la información de un SGBDs, determine los tipos de estructuras de datos físicas que soporta.

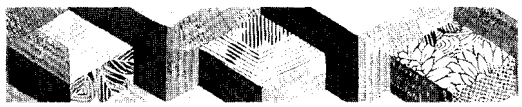
PARTE

CUATRO

ADMINISTRACIÓN DEL ENTORNO DE BASES DE DATOS

En la parte 4 aprenderá temas importantes pertenecientes al entorno de la base de datos, tales como la administración de la base de datos, sistemas distribuidos de bases de datos y selección del SGBD (DBMS). A medida que trabaje con estos capítulos, asentará y construirá el conocimiento desarrollado en partes anteriores del libro y aprenderá conceptos que son de importancia creciente para el uso eficaz de los sistemas de bases de datos en nuestra sociedad.

El capítulo 11 analiza las funciones del administrador de la base de datos (ABD, DBA) y examina las técnicas actuales para protección de la base de datos ante un acceso no autorizado, así como la manera de asegurar la integridad de sus datos. El capítulo 12 considera el rápido desarrollo de sistemas de bases de datos distribuidas, en los que se conectan las bases de datos por enlaces de comunicaciones para permitir una compartición más amplia de las bases de datos. El capítulo 13 considera el tema de la selección del SGBD. ¿Qué factores son importantes? ¿Cómo se pueden medir y evaluar, y cuestiones relacionadas?



C A P Í T U L O

11

LA ADMINISTRACIÓN Y EL CONTROL DE LA BASE DE DATOS

Administración de la base de datos: Una visión panorámica
Funciones del ABD
Comunicación con los usuarios
Establecimiento de normas y procedimientos
1. Análisis y rutas de los informes de problemas
2. Supervisión del Hardware y del Software
3. Pruebas
4. Seguridad
5. Copia de seguridad y recuperación
6. Evaluación del rendimiento
7. Comprobaciones de integridad

Objetivos del ABD
Integridad de la base de datos
Restricciones de integridad en SQL
Restricciones de integridad en Query-by-Example
Procesamiento de transacciones
Control de la concurrencia
Seguridad de la base de datos
Autentificación (Autenticación)
Autorización y vistas
Tipos de acceso a las vistas
Vistas y seguridad en SQL
Cifrados
Método de sustitución simple
Método de sustitución polialfabética
Recuperación de la base de datos
Fuentes de fallos
Procedimientos de recuperación
Diario de operaciones con actualizaciones diferidas
Diario de operaciones con actualizaciones inmediatas
Puntos de chequeo
Resumen
Preguntas de comprobación
Problemas y ejercicios
Proyectos y cuestiones profesionales

En la reunión semanal del personal de la compañía Servicios de Consultoría Manwaring, el asunto principal se refería al informe sobre el estado de los diversos proyectos bajo contrato. La mayoría de los proyectos involucraban el diseño y el desarrollo de bases de datos; sin embargo, Elmer Nordland dirigía un proyecto cuya atención principal estaba en los aspectos administrativos y de control. Joan Manwaring pensó que podría ser instructivo para los otros administradores de proyectos aprender sobre estos aspectos y cómo el equipo de Elmer los trataba.

Joan preguntó: "Elmer, ¿los problemas técnicos de los sistemas de bases de datos están separados de los aspectos administrativos y de control? He oído decir a varias personas que la buena gestión es la buena gestión, que los principios efectivos de la gestión no varían con la tecnología."

Elmer movió la cabeza. "Hay alguna justificación para ese punto de vista, pero, tal como veo las cosas, ambos no pueden considerarse realmente por separado. De hecho, encontramos que algunos problemas se pueden clasificar fácilmente en un campo o en el otro. Para mí las implementaciones exitosas de los sistemas de bases de datos están acompañadas siempre de una comprensión profunda de los aspectos administrativos y de control."

Los sistemas de información se consideran cada vez más como recursos que requieren una buena gestión, así como también de buenas características técnicas. Dado que los sistemas de bases de datos frecuentemente forman el núcleo del sistema de información de una organización, constituyen el foco de muchos aspectos administrativos a los que nos referiremos en este capítulo. Después de leer este capítulo, debería ser capaz de:

- Explicar la importancia de la gestión de las bases de datos.
- Indicar y describir las funciones del administrador de las bases de datos.
- Estudiar cómo puede mantenerse la integridad de los datos.
- Describir cómo puede implementarse la seguridad de los datos.
- Indicar las fnentes de fallos de las bases de datos y comparar los diversos métodos de recuperación.

▼ Gestión de la base de datos: Una visión panorámica

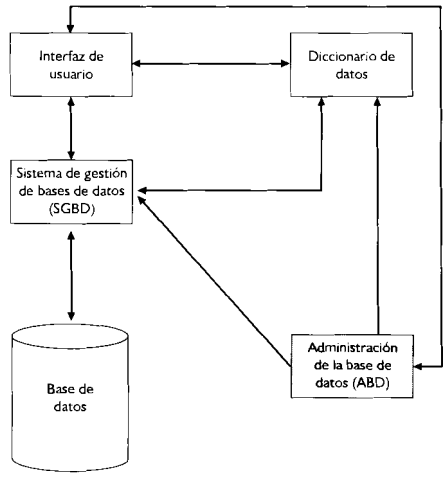
administrador de los datos (ABD, en inglés *DBA*). Directivo cuyas responsabilidades se centran en el establecimiento de las políticas y los procedimientos para el sistema de información de la organización.

administrador de la base de datos. Directivo cuyas responsabilidades se centran en la gestión de los aspectos técnicos del sistema de base de datos.

A la administración de la base de datos le concierne básicamente el asegurar que la información precisa y consistente esté disponible para los usuarios y para las aplicaciones cuando la necesiten y en la forma requerida. De este modo, el ABD interactúa tanto con el sistema como con los nsnarios (Figura 11.1). Algunas organizaciones han dividido la responsabilidad de administrar los recursos del sistema de información entre un **administrador de los datos** (AD) y un **administrador de la base de datos** (ABD). En estos casos, las responsabilidades de los AD se centran comúnmente en el desarrollo de los procedimientos y de las políticas generales para el sistema de información, mientras que las responsabilidades de los ABD tienden a ser más técnicas, tal como se sugiere en la Figura 11.1. El ABD se preocnpará por cuestiones tales como la determinación de las definiciones de los datos; el desarrollo de programas para generar la información necesaria; la inclusión y la eliminación de los datos de la base de datos; la implementación de los controles de seguridad e integridad; y la gestión de las operaciones de la base de datos.

Además de estas responsabilidades primarias, el ABD debe jugar un papel primordial en la planificación y el desarrollo de la base de datos, así como en la formación de los usuarios. Esta educación incluye áreas tales como:

1. La manera en que la tecnología de las bases de datos puede ayudar a los diversos niveles de dirección (importante para ganar y mantener el apoyo al sistema de base de datos por parte de la organización).



2. El desarrollo de expectativas realistas para el sistema de base de datos (importante para disminuir las quejas de los usuarios).
3. Procedimientos para resolver el problema de la información (importante para mantener la eficacia del sistema y la satisfacción del usuario).

Se centra la atención en el ABD, ya que las responsabilidades de esta función se asocian directamente con el sistema de base de datos, considerando que muchas de las responsabilidades del AD existen en cualquier organización que utiliza datos mecanizados. Por ejemplo, el establecimiento de los procedimientos para la obtención y la validación de los datos puede ser parte de la responsabilidad del AD, pero esta responsabilidad no depende del uso de un sistema de base de datos.

Aunque hay variaciones en cuanto a la posición exacta que ocupa el ABD en el diagrama orgánico, en la Figura 11.2 (a) se sugiere una localización razonable. Aquí el funcionario principal de información (CIO [1]) responde al funcionario administrativo principal y el ABD responde al CIO. Si la compañía tuviera un AD, el ABD podría trasladarse descendentemente una posición, respondiendo entonces al AD (Figura 11.2 (b)).

Examinemos ahora las funciones específicas que pueden constituir la responsabilidad del ABD.

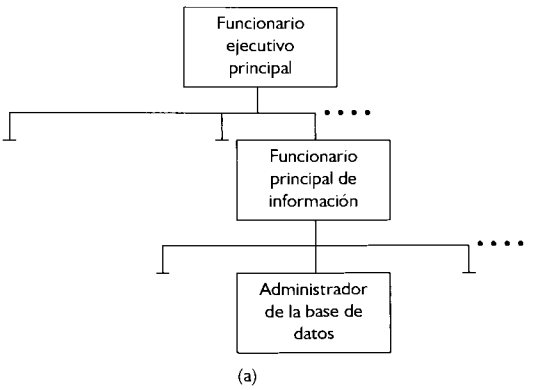
▼ Funciones del ABD

Las funciones del ABD generalmente pueden pertenecer a las áreas de comunicación con los usuarios de la base de datos; la planificación, el diseño y la implementación de los sistemas de bases de datos; el establecimiento de normas y procedimientos. La planificación, el diseño y la implementación de los sistemas de bases de datos se han estudiado en los capítulos anteriores y se excluyen de la discusión que sigue.

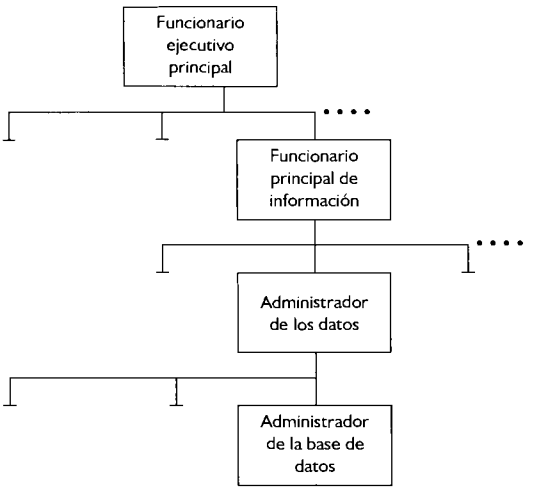
Comunicación con los usuarios

Los sistemas de bases de datos frecuentemente tienen tres componentes: el central, una base de datos ampliamente utilizada que contiene la mayoría de los datos de la firma; varias bases de datos funcionales (e. g., para la contabilidad) utilizadas por un conjunto

[1] CIO - *Chief Executive Officer* (N. del T.).



(a)



(b)

más limitado de programas; y, quizás, unas pocas bases de datos dedicadas, utilizadas para una aplicación única (e. g., una base de datos de facturas de materiales). Aquí el aspecto organizativo más importante consiste en que el impacto general al implementar un sistema de base de datos es la centralización de una porción significativa de los datos de la firma.

La centralización de los datos mediante un sistema de base de datos tiende a eliminar la propiedad local de los datos y a reducir la redundancia. La propiedad y el control se transfieren al diccionario de datos central, que almacena el registro de la propiedad y el uso de cada elemento de los datos. Ese cambio del control sobre los datos puede generar resistencia de algunos usuarios. Esta resistencia puede mitigarse educando activamente a los usuarios con respecto a las ventajas de aprender la tecnología de las bases de datos: cómo los puede hacer más efectivos y eficientes en sus tareas. El ABD, en cooperación con los directivos de más alto nivel, debe garantizar esta educación.

La decisión de implementar un sistema de base de datos generalmente exige un compromiso de cambios importantes en los niveles operativos de la organización. El ABD puede fomentar la aceptación de estos cambios promocionando el sistema de base de datos internamente *antes* de que comience su implementación. La importancia de preparar miembros claves del personal para la implementación de un sistema de base de datos es difícil de sobreenfatizar. Algunos aspectos de su trabajo actual pueden ser "asumidos" por

el sistema de base de datos y las estructuras estándares de diseño para los sistemas y los programas pueden cambiar significativamente. Cumplir con la preparación deseada pudiera incluir presentaciones especiales a los directivos, jornadas de entrenamiento para el personal clave y, posiblemente, el uso de consultores externos. Comúnmente, la capacitación es la faceta más importante de la preparación para el cambio.

El entrenamiento debe darle al personal una visión amplia de la función de un sistema de base de datos como parte integral del sistema de información de la empresa, así como también darle una guía específica con respecto a cómo puede utilizarse en las actividades diarias del usuario. Para tener éxito, el entrenamiento debe considerarse como un proceso contiuuo impulsado por nuevas contrataciones, nuevas versiones de software y el desarrollo de aplicaciones nuevas o mejoradas.

Establecimiento de normas y procedimientos

La administración efectiva de una base de datos requiere el establecimiento de normas y procedimientos uniformes. Su propósito consiste en mantener el control de la seguridad y la integridad de los datos de manera eficiente. Las normas se aplican particularmente al control del desarrollo y del uso de la programación y de las operaciones de la base de datos.

En el área de la programación, las normas se establecen para asegurar que los programas se revisen y se prueben adecuadamente antes de ponerlos en producción. Estas normas pueden requerir una revisión por una segunda parte competente, así como también el uso de datos de prueba para evaluar cómo un programa gestiona tanto los datos correctos como los erróneos. El procedimiento usual incluye la documentación de los resultados de las pruebas.

En el área de las operaciones, las normas pueden establecerse para mantener los diarios de operaciones de las transacciones y se crean los procedimientos para la corrección de los errores, para el tratamiento de los puntos de control y para garantizar la copia de seguridad y la recuperación.

Las organizaciones que tengan pocas normas y procedimientos podrían experimentar mayores dificultades al pasar a un ambiente de bases de datos, ya que las estadísticas demuestran que la gestión integrada de los datos, facilitada por los sistemas de bases de datos, requiere normas y procedimientos buenos y comprensibles. Examinar las normas en uso de otras organizaciones que ya utilicen sistemas de bases de datos puede serle útil a una organización que comience a implementar un sistema de base de datos. Por ejemplo, las funciones siguientes forman el núcleo de las normas y los procedimientos en la compañía Zeus Corporation.

1. *Análisis y rutas de los informes de problemas.* En Zeus se estableció un sistema formal para el informe de los problemas a fin de poner en conocimiento del ABD todos los errores. Los informes de los problemas se analizan para determinar la causa probable de cada problema reportado. Entonces, los informes se ponen a la disposición del directivo o del grupo de usuarios apropiado. Cada informe de los problemas contiene un diario de operaciones *(log)* completo (momento y localización del problema) y la información descriptiva. Cada informe requiere una respuesta formal del iniciador del mismo que especifique cómo se ha resuelto el problema.

2. *Supervisión del hardware y del software.* El estado de todo el hardware y el software se controla regularmente y se confeccionan informes de los fallos y de las acciones consiguientes para los directivos y los grupos de usuarios. Se debe hacer el análisis periódico de los requisitos de hardware y de software, que constituye la base para las decisiones acerca de su reemplazo y renovación y que incluye las necesidades de medios adicionales para el almacenamiento de las bases de datos.

3. *Prueba.* Las pruebas para la aceptación del rendimiento se dirigen a la evaluación de todos los procedimientos nuevos, el software y el hardware. Las comprobaciones de las estructuras y de la consistencia de la base de datos se realizan sobre bases regulares.

4. *Seguridad.* En consulta con los directivos de Zeus, las clasificaciones de la seguridad se implementaron de modo que identifican qué grupos de usuarios están autorizados a tener acceso a los elementos de los datos específicos en la base de datos y qué acciones pueden ejecutar sobre ellos. Las operaciones del computador se supervisan frecuentemente para asegurar que estos controles de acceso funcionan de la forma requerida.

5. *Copia de seguridad y recuperación.* Los procedimientos de respaldo y de recuperación se prueban regularmente para asegurar su eficacia en la restauración de la base de datos después de cualquier interrupción del servicio. Se ha diseñado un plan para casos de desastres (para responder a desastres naturales, tales como inundaciones o averías eléctricas) y se prueba periódicamente para garantizar que funcione.

6. *Evaluación del rendimiento.* Se han asignado prioridades a las actividades que compiten por los recursos de la base de datos, tales como el procesamiento de las transacciones, la generación de los informes y el tratamiento de las consultas. El rendimiento del sistema se controla a través de estadísticas sobre el volumen de las transacciones, el tiempo de respuesta, la tasa de errores y la utilización del hardware. La entrada se obtiene de los usuarios del sistema para controlar su satisfacción con el rendimiento del sistema. También se registran el crecimiento y el tamaño de la base de datos. Los programas de expansión de archivos y las reorganizaciones de la base de datos se ejecutan siempre que sea necesario. Los diarios de operaciones de las actividades y de las terminaciones anormales se revisan, extrayéndose de ellos resúmenes que se preparan para la valoración de los directivos.

7. *Comprobaciones de integridad.* En Zeus se han desarrollado planes para probar la integridad de los datos almacenados en la base de datos.

procesamiento concurrente (concurrencia). Ocurre cuando dos o más transacciones piden acceso concurrentemente a un mismo registro de la base de datos aproximadamente al mismo tiempo.

integridad de los datos. La exactitud y la consistencia de los datos almacenados en el sistema de base de datos.

seguridad de los datos. Se refiere a la protección del sistema de base de datos ante usos mal intencionados o no autorizados.

procedimientos de recuperación de la base de datos. Los medios mediante los cuales una base de datos que se ha corrompido por desperfectos puede restablecerse a un estado correcto y consistente.

▼ Objetivos del ABD

Como se puede ver, una gran parte del esfuerzo involucrado en la administración de la base de datos se refiere al aseguramiento de la calidad y de la disponibilidad del sistema de base de datos. Esto está en consonancia con los objetivos básicos del ABD: mantener la integridad, la seguridad y la disponibilidad de los datos.

Una base de datos tiene que protegerse de accidentes, tales como los errores en la entrada de los datos o en la programación, del uso mal intencionado de la base de datos y de los fallos del hardware o del software, que corrompen los datos. La protección contra accidentes, que ocasionan inexactitudes en los datos, es parte del objetivo de garantizar la integridad de los datos. Estos accidentes incluyen los fallos durante el procesamiento de las transacciones, los errores lógicos que infringen la suposición de que las transacciones preservan las restricciones de consistencia de la base de datos y las anomalías debido al acceso concurrente a la base de datos (**procesamiento concurrente**).

La protección de la base de datos de usos mal intencionados o no autorizados se denomina seguridad de los datos. Aunque la línea divisoria entre la integridad y la seguridad de los datos no es precisa, una definición de trabajo puede considerarse como se indica a continuación:

1. La **integridad** se encarga de asegurar que las operaciones ejecutadas por los usuarios sean correctas y mantengan la consistencia de la base de datos.
2. La **seguridad** se encarga de limitar a los usuarios a ejecutar únicamente las operaciones permitidas.

La posibilidad de fallos de hardware o de software requiere también la implementación de **procedimientos de recuperación de la base de datos.** Esto es, tienen que proporcionarse medios para el restablecimiento de las bases de datos, que se hayan corrompido por desperfectos del sistema, a un estado uniforme.

Estas facetas de la gestión de la base de datos se examinan en las secciones siguientes.

▼ Integridad de la base de datos

control de integridad (restricción). Una restricción aplicada a un conjunto determinado de datos; utilizado para minimizar los errores en la entrada de los datos.

Una condición o limitación que se aplica a un conjunto particular de datos usualmente se denomina **control de integridad** o **restricción**. Por ejemplo, los valores de las horas trabajadas en una semana podrían limitarse a 60 horas, o menos. Los controles de integridad se diseñan para minimizar las inconsistencias de los datos, ocasionadas cuando los usuarios o los programas de aplicación cometen errores en la entrada de datos o cambiando los datos de la base de datos.

Al imponer restricciones semánticas sobre los datos, es posible que confiemos razonablemente en que los contenidos de la base de datos son correctos y que no existen inconsecuencias entre los datos. Aunque las restricciones de integridad se aplican a todos los modelos de bases de datos, (e. g., se pueden aplicar restricciones de retención de conjunto en las redes), la mayoría de los desarrollos actuales se aplican al modelo relacional. A partir de ahora, el estudio se centrará en ello.

En la terminología del modelo relacional, los controles de integridad se pueden aplicar a (1) atributos individuales, (2) la interrelación entre dos atributos diferentes (quizás en relaciones diferentes), o (3) la relación entre tuplas de una o más tablas. Idealmente, la aplicación de las restricciones de integridad debería efectuarse por el SGBD automáticamente cada vez que se introduzca un nuevo elemento de los datos. (Los datos que no satisfagan las restricciones establecidas se rechazarían, por ejemplo, por medio de un mensaje de error.) Desafortunadamente, en la actualidad pocos SGBDs no ofrecen más que capacidades muy limitadas para imponer restricciones.

Una alternativa consiste en escribir programas de aplicación para controlar la entrada de todos los datos a una base de datos. Esto permite mayor flexibilidad y alcance al aplicar diferentes tipos de restricciones. Sin embargo, el desarrollo de tales programas puede consumir gran tiempo. A veces pueden utilizarse ambos métodos conjuntamente. El SGBD se utiliza para extender su capacidad para la supervisión de las restricciones semánticas; entonces, los programas de aplicación se desarrollan para las necesidades que no puedan ser manejadas por el SGBD.

Restricciones de integridad en SQL

Éstas se trataron con cierto detalle en los Capítulos 7 y 9.

Restricciones de integridad en Query-By-Example

Algunos sistemas que utilizan el Query-By-Example (QBE) mantienen una tabla de restricciones para cada relación. Para crear una restricción sobre una relación R, se crea un esqueleto de tabla para R. Se introducen una o más líneas para especificar las restricciones. Como se muestra en la Figura 11.3,

```
I. CONSTR (I., U.). I.
```

se introduce justamente debajo del nombre de la relación INVENTARIO.

INVENTARIO	NO_ARTÍCULO	NOMB_ARTÍCULO	BALANCE
I. CONSTRU (I., U.).I.			> = 0

La forma general de esta sintaxis es:

```
I. CONSTR (<lista de condiciones>). I. < restricciones >
```

"I. CONSTR (<lista de condiciones>)" indica que se está insertando una restricción. La segunda I. se refiere a las entradas que definen la restricción. La lista de condiciones puede contener I. (para insertar), D. (para borrar)[2] y U. (para actualizar o modificar)[3], o cualquier combinación de ellas. En la Figura 11.3, se muestra la restricción de los balances del inventario a números positivos en la columna BALANCE. CONSTR (I., U.) indica que la restricción debe verificarse siempre que ocurra una inserción (I.) o una actualización (U.) en la relación.

Las entradas para uno o más atributos se colocan en las filas del esqueleto. Si una entrada es una constante, entonces la tupla que se inserta, que se borra o que se actualiza debe tener dicho valor constante en ese atributo. En la Figura 11.3 se muestra una entrada de la forma *kc*, donde *c* es una constante (0) y *k* es un operador aritmético de comparación (> =). Esto indica que la componente correspondiente de una tupla debe encontrarse en la relación *k* con respecto a *c*, siempre que la restricción se aplique a la tupla. En el ejemplo, esto significa que siempre que se entre un valor en la columna BALANCE, ese valor no puede ser negativo.

Una entrada también puede ser vacía o contener un nombre de variable que comienza con el carácter de subrayado; ambos casos significan que el valor de ese atributo puede ser arbitrario. Esta característica se utiliza en el segundo ejemplo de QBE, en el que no se desea permitir un crédito de venta a menos que el cliente aparezca en la relación CRÉDITO_APROBADO. En la Figura 11.4 se muestra esta restricción de integridad utilizando QBE. Se solicitan los esqueletos VENTAS_A_CRÉDITO y CRÉDITO_APROBADO y se introduce la información mostrada en la Figura 11.4. Esta restricción garantiza que la tupla que se inserte, que determina un valor para "_nombre" en el atributo NOMB_CLIENTE de la tupla insertada, tenga el mismo valor "_nombre" que alguna tupla en la relación CRÉDITO_APROBADO.

Procesamiento de transacciones

transacción. Una unidad de programa cuya ejecución conserva la consistencia de la base de datos.

transacción atómica. Una transacción en la cual todas las acciones asociadas con ella se ejecutan completamente o ninguna se ejecuta.

Otro concepto asociado con la integridad de la base de datos se denomina transacción de la base de datos. Una **transacción** es una unidad de programa cuya ejecución conserva la consistencia de la base de datos. Si la base de datos está en un estado consistente antes de la ejecución de una transacción, entonces la base de datos debe continuar en un estado consistente después de su ejecución. Para asegurar estas condiciones, las transacciones tienen que ser **atómicas**, lo que significa que las acciones asociadas con la transacción o se ejecutan todas hasta el final o ninguna se ejecuta. Por ejemplo, una transacción para registrar el pago de un cliente de $500 puede incluir las acciones siguientes:

VENTAS_A_CRÉDITO	NOMB_CLIENTE	IMPORTE
I. CONSTRU (I.).I.	_nombre	

CRÉDITO_APROBADO	NOMB_CLIENTE	LÍMITE
	_nombre	

[2] D del inglés *Delete* (N. del T.)
[3] U del inglés *Update* (N. del T.)

1. Cambiar el registro del cliente reduciendo el balance de la cuenta en $500.
2. Cambiar el registro en caja aumentando su balance en $500.

Quizás el segundo paso falle. Entonces, las cuentas quedarían fuera de balance. La Figura 11.5 muestra qué sucede cuando estas acciones se ejecutan como una serie de pasos independientes (a) y cuando se ejecutan como una transacción atómica única (b). El punto clave está en que cuando las acciones se ejecutan automáticamente y una falla, entonces, ningún cambio se aplica a la base de datos. Se dice que dichas transacciones han sido **canceladas**.

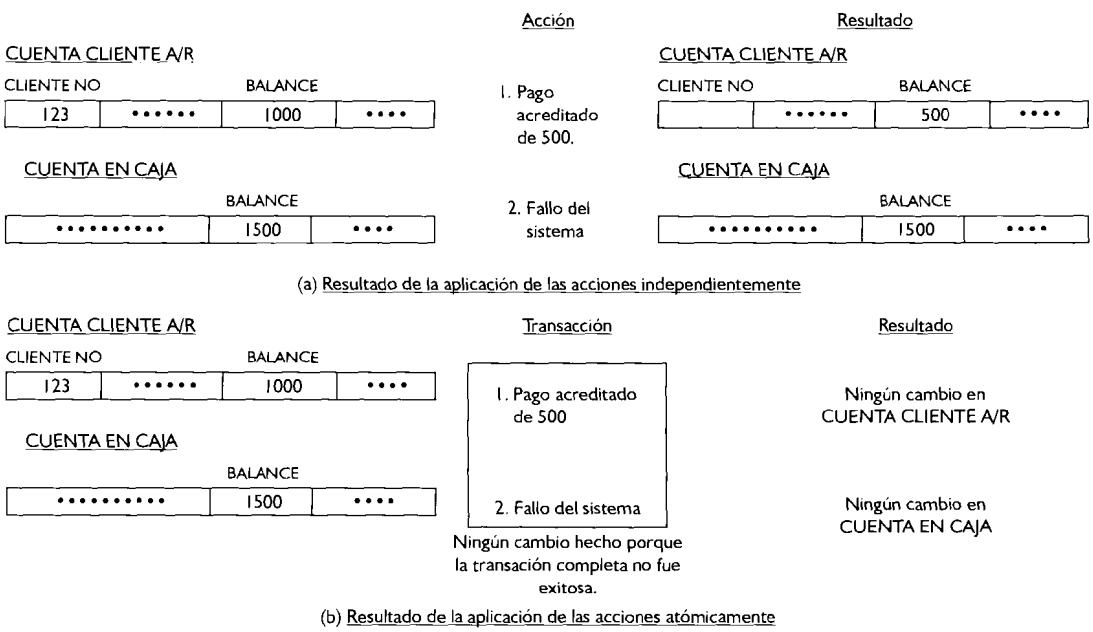
transacción cancelada Transacción que se cancela antes de aplicar los cambios a la base de datos.

Para soportar el procesamiento de una transacción, un SGBD debe mantener un registro de la transacción con *cada cambio* que se hizo en la base de datos. La manera en que esto se logra consiste en utilizar un **diario de operaciones** *(log)*. Cuando un cliente hace un pago de $500 sobre la cuenta, las acciones de la transacción incluyen (1) acreditar la cuenta del cliente y (2) debitar la cuenta en caja. Durante la ejecución de la transacción, todas las operaciones de escritura deben delegarse hasta que la última acción de la transacción haya sido ejecutada. Las actualizaciones resultantes se registran en el diario de operaciones de la transacción. Cuando todas las acciones hayan sido ejecutadas, se utiliza la información de actualización en el diario de operaciones para escribir la información actualizada en los registros apropiados de la base de datos. Si el sistema falla antes que la transacción se ejecute completamente, la información del diario de operaciones nunca se escribe en esos registros. Se profundizará más acerca de este diario de operaciones en la discusión de las técnicas de recuperación.

diario de operaciones. Un registro de todas las transacciones y los cambios correspondientes en la base de datos.

Control de la concurrencia

Supongamos que la Liga de Mujeres Votantes (LMV) en Smithville decide tener una Cena de Jamón y Pavo. La Cámara de Comercio (CC) en el cercano Johnstown decide que es también tiempo de premiar a sus miembros con una Cena de Jamón y Habas. Ambas organizaciones contactan con el centro de distribución de Alimentos de la Ciudad. La LMV quiere 25 jamones; la CC quiere 35 jamones. Ambas órdenes se transmiten al sistema de



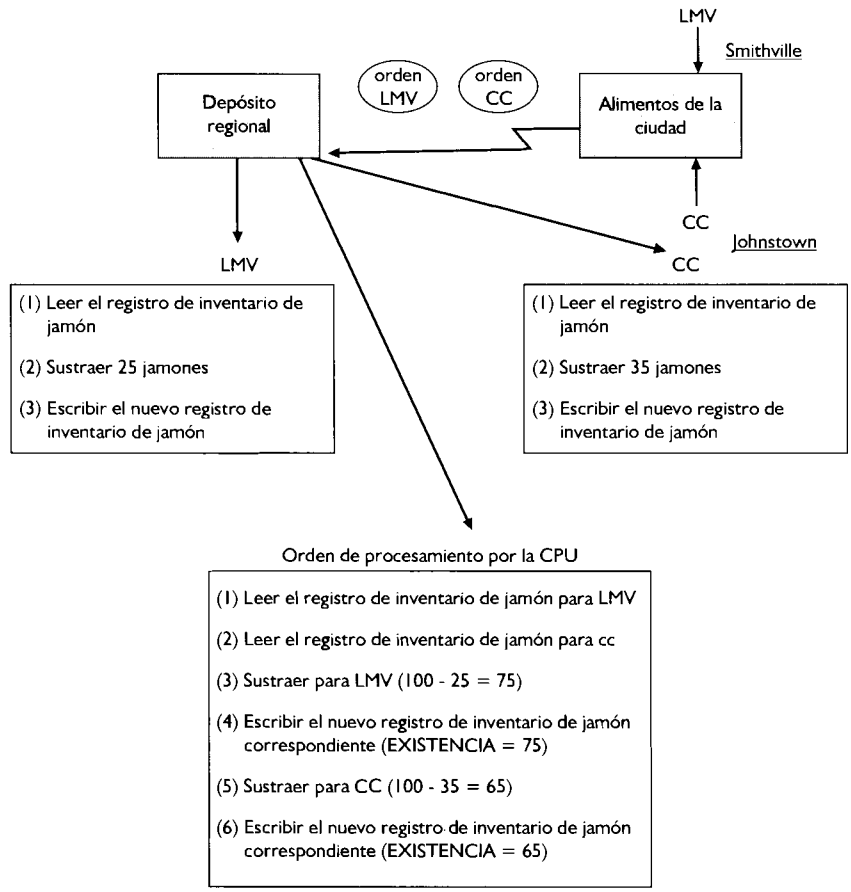
base de datos del depósito regional al mismo tiempo (Figura 11.6). La orden de la LMV llega una fracción de un segundo antes que la orden de la CC. Una imagen del registro de inventario para el jamón se coloca en la zona de trabajo del computador. El registro de inventario muestra 100 jamones en existencia. Pero antes de que se complete la transacción LMV y el registro de inventario se actualice, la transacción CC obtiene también una copia del mismo registro de inventario que muestra 100 jamones en existencia, que se coloca en su zona de trabajo. Ambos registros muestran que la orden en cuestión puede completarse.

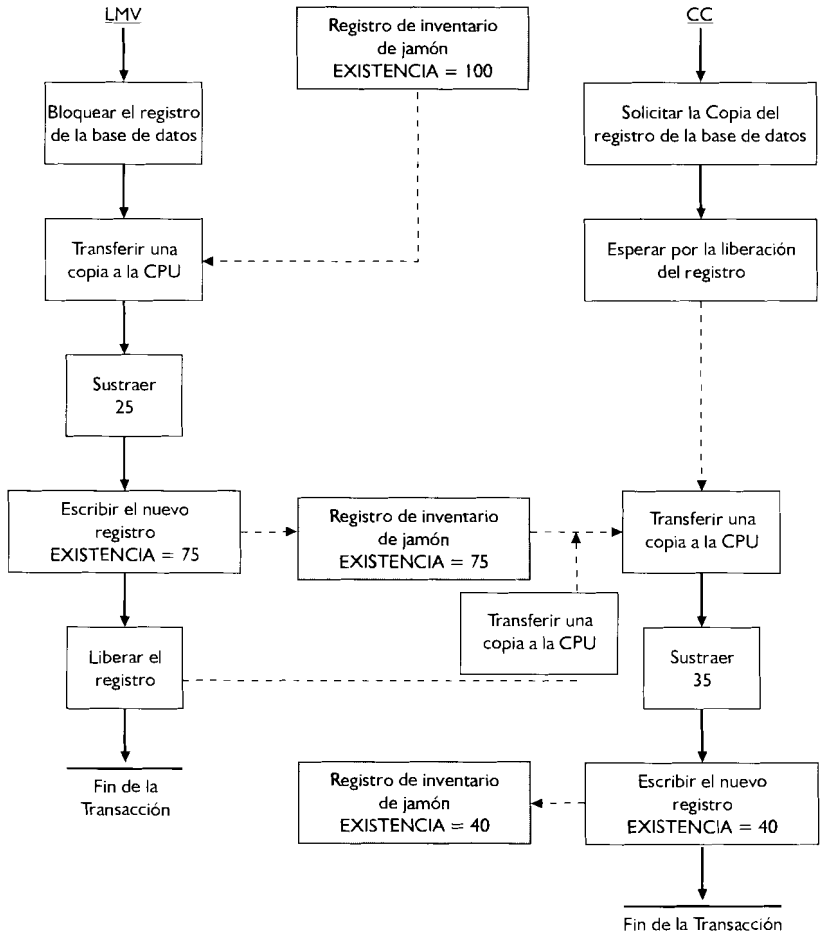
Supongamos que la transacción LMV se completa primero. El registro de inventario reescrito muestra que quedan en existencia 100 – 25 = 75 jamones. Al completarse la transacción CC, el registro de inventario se sobreescribe con 100 – 35 = 65 jamones en existencia. Lo que realmente ha sucedido es que se han vendido 60 jamones de una existencia de 100, dejando solamente *40* jamones en el depósito, pero el balance de inventario del sistema muestra *65* jamones disponibles. ¡Montones de problemas! Este ejemplo ilustra la naturaleza fundamental del procesamiento concurrente:

Si dos o más usuarios tienen acceso a la base de datos al mismo tiempo y las transacciones se entrecruzan (como en el ejemplo anterior), pueden ocurrir resultados indeseables.

bloqueo. Impide el acceso a un registro de la base de datos por una segunda transacción hasta que la primera transacción haya completado todas sus acciones.

Una manera usual de impedir los problemas de la concurrencia se logra mediante la aplicación de una simple política de bloqueo *(locking policy)*. En el ejemplo precedente, la primera transacción que llega **bloquearía** *(lock)* (impediría el acceso a cualquier otra transacción) el registro de inventario de los jamones (Figura 11.7) hasta que el procesamiento



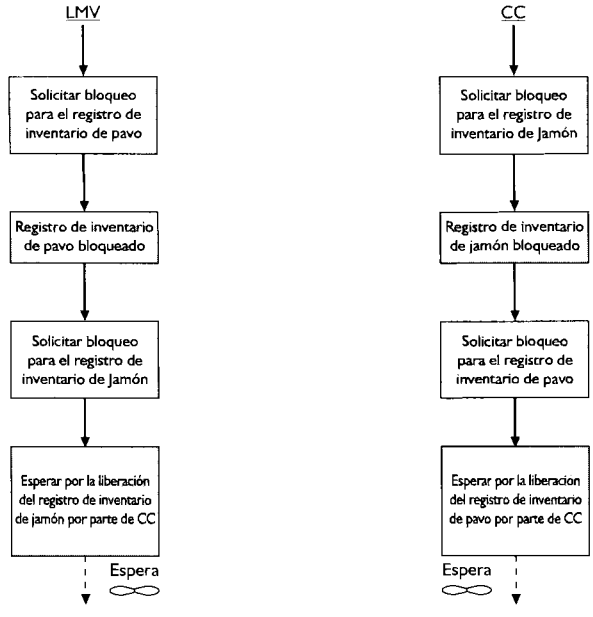


de la primera transacción se complete. Cuando un registro se bloquea por un usuario, ningún otro usuario puede tener acceso a él para su actualización. En este caso, la transacción CC no podría haber tenido acceso al registro hasta que la transacción LMV se completara.

Aunque este enfoque sea simple y útil, no es seguro. Supongamos que la LMV y la CC quieren ordenar jamón y pavo (Figura 11.8). Supongamos que la transacción LMV pide primero el registro de inventario de pavo. Como éste está bloqueado, la transacción CC tiene acceso primero al registro de inventario de jamón. Así, estos registros están bloqueados a los efectos de su uso por otros usuarios. En pocos momentos tanto la transacción LMV como la CC habrán completado el procesamiento de los registros de inventario de jamón y de pavo, respectivamente, y estarán listos para el acceso al otro registro. Sin embargo, ninguno de los dos usuarios ha completado su transacción, por lo que los registros que han accedido permanecen bloqueados. Estamos ante una situación de **interbloqueo** (***deadlock***), en la que ningún usuario puede proseguir.

interbloqueo. Dos transacciones se excluyen mutuamente del acceso al siguiente registro requerido para completar cada una de sus transacciones; también se denomina "abrazo mortal".

Existen varios métodos de tratar el interbloqueo. Un enfoque consiste en ordenar los registros que se bloquean. Esto es, si se tiene acceso a los registros A y B, siempre tienen que ser accedidos en ese orden. En el ejemplo precedente, las solicitudes de los registros de inventario de jamón y de pavo podrían requerir, en primer lugar, el acceso al registro de inventario de jamón y, en segundo lugar, al registro de inventario de pavo. Si así fuera, es



fácil ver que nunca podría ocurrir una situación de interbloqueo. Cuando la transacción LMV termine el procesamiento, todos los registros accedidos deberían ser liberados para su uso por la transacción CC. Sin embargo, esto puede demorar las operaciones, puesto que la espera de las transacciones puede convertirse en un cuello de botella serio. Además, si una transacción tiene que esperar demasiado por la liberación de un registro, se puede cancelar y se le puede solicitar que lo intente nuevamente más tarde. Esto tiende a formar en los usuarios una pobre opinión sobre el sistema de base de datos.

detección de interbloqueo. Una comprobación periódica del SGBD para determinar si la cola de espera para algunos recursos excede un límite predeterminado.

Algunos SGBDs realizan la **detección de interbloqueo** comprobando regularmente si la cola de espera correspondiente a un registro o a un recurso es demasiado larga. Otro método de detección se basa en trazar (literalmente) una flecha desde la transacción al registro que está siendo consultado y, entonces, trazar una flecha desde ese registro a la transacción que lo utiliza actualmente. Si el diagrama tiene ciclos, se detecta el interbloqueo. Esto se ilustra en la Figura 11.9. Los procedimientos de detección de interbloqueo completan su función cancelando una de las transacciones y avanzando a la transacción siguiente en la cola de espera.

bloqueo en dos fases. Un método de control del procesamiento concurrente en el cual todas las operaciones de bloqueo preceden a la primera operación de liberación.

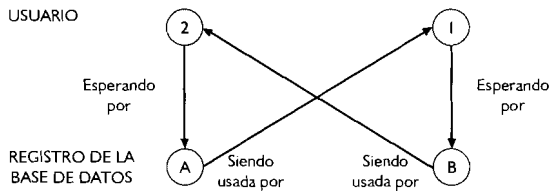
Otra opción para controlar el tratamiento concurrente es el **bloqueo en dos fases** *(two-phase locking)*. Se dice que una transacción sigue el protocolo de bloqueo en dos fases si todas las operaciones de bloqueo (bloqueo de lectura, bloqueo de escritura) preceden la primera operación de desbloqueo en la transacción. El **bloqueo de lectura** permite leer un registro, mientras que el **bloqueo de escritura**, como algo más que el bloqueo de lectura, permite tanto la lectura como la actualización de un registro.

bloqueo de lectura. El usuario tiene derecho a leer el registro dado.

Se ha probado que el bloqueo en dos fases garantiza la serialidad *(serializability)*, que significa que las transacciones pueden ejecutarse de tal manera que sus resultados son los mismos que si las acciones de cada transacción se ejecutaran en secuencia sin interrupciones. Sin embargo, el bloqueo en dos fases tiene su precio. Puede conducir a interbloqueo a menos que se utilice conjuntamente con un protocolo de prevención de interbloqueo.

bloqueo de escritura. El usuario tiene derecho a leer y a actualizar el registro dado.

Como un ejemplo, en la Figura 11.10 (a) se presenta un procedimiento de bloqueo en dos fases que conduce a interbloqueo. La transacción 1 (T_1) adquiere un bloqueo de lectura sobre J (Jamón) al mismo tiempo que T_2 adquiere un bloqueo de lectura sobre P (Pavo). Dos pasos más adelante, T_1 y T_2 buscan bloqueos de escritura sobre los registros de la base



de datos que en ese momento se encuentran respectivamente bloqueados en lectura por la otra transacción. Esta acción se deniega porque entonces un valor podría cambiarse por una operación de escritura ejecutada por la otra trausacción. En esta situación, el poseedor concurrente del bloqueo de lectura estaría leyendo un valor incorrecto.

En la Figura 11.10(b) se muestra el uso del bloqueo en dos fases con un protocolo de prevención de interbloqueo. Se le ha añadido el requisito de que cada transacción bloquee por adelantado todos los elementos de los datos que necesita. Si cualquiera de los elementos de los datos no puede obtenerse, entonces ninguno de los artículos se bloquea y la transacción se pone en espera hasta que todos los artículos que se necesitan estén disponibles. En la Figura 11.10(b), se supone que el pedido de T_1 ha comenzado un momento antes que el de T_2. T_2 no puede bloquear todos los elementos de los datos que requiere hasta que T_1 libere a J y a P. De esta manera, se evita el interbloqueo de la Figura 11.10(a); sin embargo, en este ejemplo el resultado ha sido simplemente el procesamiento de las transacciones en serie. Esto es específico para este ejemplo y podría ocurrir siempre que dos transacciones compitan por exactamente el mismo conjunto de registros.

T_1	Valores del Inventario de Pavo (P) y Jamón (J)	T_2	Valores del Inventario de Pavo (P) y Jamón (J)
Bloquear_lectura (J)		Bloquear_lectura (P)	
Leer_elemento (J)	J = 100	Leer_elemento (P)	T = 30
Bloquear_escritura (P)	DENEGADO	Bloquear_escritura (J)	
Desbloquear (J)	P = 30	Desbloquear (P)	
Leer_elemento (P)		Leer_elemento (J)	
T = T - 25		J = J - 25	
Escribir_elemento (P)	P = 5	Escribir_elemento (J)	
Desbloquear (P)		Desbloquear (J)	

T_1	Valores del Inventario de Pavo (P) y Jamón (J)	T_2	Valores del Inventario de Pavo (P) y Jamón (J)
Bloquear_lectura (J)			
Bloquear_escritura (P)		(esperando)	
Leer elemento (J)	J = 100		
Desbloquear (J)		Bloquear_escritura (J)	
Leer_elemento (P)	P = 30	Bloquear_lectura (P)	
P = P – 25		Leer_elemento (J)	J = 100
Escribir elemento (P)	P = 5	J = J – 25	
Desbloquear (P)		Escribir elemento (J)	J = 75
		Desbloquear (J)	
		Leer_elemento (P)	P = 5
		Desbloquear (P)	

T_1	Valores del Inventario de Pavo (P) y Jamón (J)	T_2	Valores del Inventario de Jamón (J) y Carne (C)
Bloquear_lectura (J)			
Bloquear_escritura (P)		(esperando)	
Leer_elemento (J)	J = 100		
Desbloquear (J)		Bloquear_escritura (J)	
Leer_elemento (P)	P = 30	Bloquear_lectura (C)	
P = P – 25		Leer_elemento (J)	J = 100
Escribir elemento (P)	P = 5	J = J – 25	
Desbloquear (P)		Escribir elemento (J)	J = 75
		Desbloquear (J)	
		Leer_elemento (C)	C = 60
		Desbloquear (C)	

En la Figura 11.11 se muestra un ejemplo demostrativo de operaciones paralelas que son serializables. En este caso, T_2 necesita solamente esperar que T_1 libere el artículo J. Entonces, T_2 puede bloquear todos los registros que necesita para su procesamiento, y ambos T_1 y T_2 pueden proceder en paralelo. Los resultados de T_1 y T_2 son los mismos que si se procesaran en serie.

▼ Seguridad de la base de datos

Los problemas de integridad de la base de datos pueden ser desafiantes, pero en general son más fáciles de manejar que el acceso mal intencionado a la base de datos, que incluye lo siguiente:

1. El latrocinio de información.
2. La modificación no autorizada de los datos.
3. La destrucción no autorizada de los datos.

Así, los métodos de seguridad de la base de datos se enfocan hacia la prevención del acceso a la base de datos de usuarios no autorizados. Debido a que las características de los SGBD que facilitan el acceso y la manipulación de las bases de datos también abren sus puertas a intrusos, la mayoría de los SGBDs incluyen herramientas de seguridad que permiten el acceso a los datos únicamente a los programas o a las personas autorizadas y, además, restringen los tipos de procesamiento que se pueden realizar una vez que se tenga el acceso.

Autentificación (Autenticación)

El acceso a la base de datos comúnmente requiere la autorización y la autentificación del usuario. Para la autentificación del usuario, el primer nivel de seguridad establece que la persona que solicita la entrada al sistema es un usuario autorizado. Su identidad puede establecerse mediante (1) algo que el usuario conoce, como un número de acceso y una contraseña, (2) algo que el usuario posee, como una tarjeta plástica de identificación, o (3) una representación física del usuario, como su huella digital o un registro de su voz.

Las contraseñas, evidentemente el método de seguridad más común y frecuentemente menos caro, son adecuadas para muchas aplicaciones. Sin embargo, una vez que un intruso potencial conoce la longitud de la contraseña y el alfabeto del que se deriva, las contraseñas pueden generarse e intentarse una vez tras otra en cierto período de tiempo hasta que eventualmente se obtenga el acceso. Para algunos esquemas de contraseña, el tiempo promedio requerido para ello puede ser suficientemente largo como para desalentar a los intru-

sos casuales. Sin embargo, si la ganancia potencial es suficientemente grande, el ABD debe idear un esquema de contraseña que no brinde oportunidades a los intrusos.

El esquema ideal de contraseña limita los accesos no autorizados al sistema, creando una contraseña que es difícil de adivinar, pero, no obstante, fácil de recordar por el usuario. Para algunas aplicaciones, puede ser suficiente especificar parámetros de contraseña, tales como la longitud de la contraseña y el alfabeto a utilizar y, entonces, dejar que el usuario idee su contraseña. Si se utiliza este enfoque, puede ser aconsejable nombrar un supervisor de contraseñas para asegurar que los parámetros de contraseña están satisfechos y que se evitan las duplicaciones.

Tanto si los usuarios seleccionan sus contraseñas propias como si se emiten por una autoridad central, puede ser útil tener un supervisor que mantenga un listado cifrado de todas las contraseñas sobre un disco, accesible solamente por ese supervisor. Además, cuando un empleado termina su contrato o se transfiere a una nueva área funcional, deberían cambiarse todas las contraseñas a las que tuvo acceso dicho empleado. Esto es importante, ya que muchos ex-empleados descontentos se han dedicado a sabotear los sistemas de sus anteriores empresas.

Para sistemas sumamente sensibles, deben considerarse esquemas más sofisticados. Un esquema de este tipo programa el computador para llevar a cabo una sesión de preguntas y respuestas con el usuario, aprovechando las preguntas y las respuestas que dicho usuario ha almacenado anteriormente en el sistema. Las preguntas son comúnmente de naturaleza personal, de modo que es muy probable que solamente un usuario válido las conteste correctamente. La secuencia de preguntas podría ocurrir como se indica a continuación:

- ¿Cuál es el segundo nombre de su abuelo?
- ¿Cuándo es el cumpleaños de su hija?
- ¿Qué es lo que tiene de especial el 18 de septiembre?

Cuando el usuario pide acceso y entra un número personal de identificación, el computador le plantea preguntas aleatoriamente seleccionadas a partir de los datos almacenados. La selección también puede variar de tiempo en tiempo para limitar la posibilidad de que otras personas puedan observar las respuestas a todas las preguntas.

Otro método más para la protección contra los accesos no autorizados consiste en el uso de un algoritmo previamente acordado. Esto es, el computador da al presunto usuario un número de autenticación elegido al azar y pide una respuesta. El usuario aplica una transformación, previamente acordada, sobre el número y transmite el resultado. El computador compara el valor recibido con el que ha calculado. Si ambos coinciden, entonces se otorga el acceso. Esto es particularmente útil si un posible intruso hubiera transgredido una línea de comunicación. Todo lo que se puede observar es el resultado del algoritmo aplicado al número. La dificultad de deducir el algoritmo a partir de esta información es enorme.

El control del acceso se perfecciona si todas los intentos de acceso denegados producen una entrada en el diario de operaciones y un tiempo de retraso (esto último incrementa el tiempo requerido para obtener un acceso no autorizado). El sistema debe terminar automáticamente la comunicación con usuarios que sean incapaces de suministrar una contraseña válida dentro de un tiempo determinado o en un cierto número de intentos. También es deseable que las contraseñas no se visualicen en la pantalla y que los usuarios no descuiden el sistema después de establecida la comunicación y recibido el permiso de acceso.

Un aspecto importante de la seguridad de la base de datos es determinar cuándo la autenticación del usuario es suficiente por sí sola y cuándo se debe combinar con la identificación específica del terminal. Con la incidencia creciente de los intrusos ilegales, llamados *hackers*, ha habido casos en los que la autorización de un usuario se satisface por alguien que no se comunica desde ninguno de los terminales legítimos del sistema. Además, a veces se conmutan mecanismos falsamente conectados. El ruido en una línea de comunicación puede ocasionar que un mecanismo de llamada selectiva conecte incorrectamente un dispositivo debido a una modificación de la dirección. Con frecuencia tales errores no se detectan, lo que degrada la fiabilidad y puede comprometer la seguridad. Así, la identificación automática del terminal se recomienda siempre que se utilice un equipo de telecomunicación conmutada.

Alternativamente, la intrusión externa indeseable puede prevenirse implementando procedimientos de llamada/contestación *(dial-up/call-back procedures)*. Con esta estrategia, alguien intentando obtener el acceso al sistema suministra la información necesaria de autenticación. El sistema entonces cuelga y revierte la llamada a la localización de un terminal legítimo, suponiendo que éste fue dado. De esta manera, si ocurrió una intrusión, tiene que emanar de un microcomputador legítimo de la red. Sin embargo, las capacidades de llamada/contestación pueden representar un gasto adicional significativo.

El acceso no autorizado al sistema puede controlarse adicionalmente, permitiendo que sólo ciertos tipos de transacciones se transmitan desde un terminal determinado. Restringir ciertas terminales a ejecutar una gama estrecha de tareas puede ser un control efectivo y fácil de aplicar. Por ejemplo, las terminales en áreas relativamente públicas pueden limitarse a funciones de sólo-lectura (read-only).

Autorización y vistas

vista Un subconjunto restringido de una relación almacenada.

Una **vista** es un medio de proporcionar al usuario un modelo personalizado de la base de datos. Es también una manera útil de limitar el acceso de un usuario a diversas porciones de la base de datos: Los datos que un usuario no necesita ver son simplemente ocultados por la vista. Esto simplifica el uso del sistema a la vez que promueve la seguridad. Las vistas se pueden representar por la ejecución de restricciones, proyecciones y reuniones (joins) sobre las relaciones existentes. El usuario podría también estar restringido a ver una parte de una relación existente o a ejecutar reuniones sobre relaciones determinadas.

La creación de vistas diferentes para diversas clases de usuarios logra, automáticamente, un alto grado de control de acceso. Aunque a un usuario se le niegue el acceso directo a una relación base, el usuario puede tener acceso a una parte de esa relación mediante una vista. Por consiguiente, una combinación de la seguridad en el nivel relacional y de la seguridad en el nivel de las vistas se puede utilizar para limitar el acceso de un usuario precisamente a los datos que necesita.

Tipos de acceso a vistas. Se pueden permitir diferentes tipos de autorizaciones de acceso para una vista particular, tales como las siguientes:

1. Autorización de lectura: permite la lectura, pero no la modificación de los datos.
2. Autorización de inserción: permite la inserción de nuevos datos, pero no la modificación de los datos existentes.
3. Autorización de actualización: permite la modificación de los datos, pero no la eliminación.
4. Autorización de eliminación: permite la eliminación de datos.

Estos tipos de autorizaciones se hacen típicamente asignando contraseñas diferentes a una vista. Por ejemplo, supongamos que en una relación PROYECTOS (Figura 11.12) se

PROYECTOS NUMPROY	CLIENTE	LOCALIDAD
1	BURGER, W.	VAUDEVILLE
2	DOWNING, B.	SERIES CITY
3	MCENROE, J.	OUTLAND
4	DEXTER, M.	MARTINSVILLE
5	JOINER, W.	ALPENHAGEN
6	NIXON, R.	SAN CLEMENTE
7	MARX, K.	OUTLAND
8	BOND, J.	MARTINSVILLE
9	ELWAY, J.	ALPENHAGEN
10	LETTERMAN, D	OUTLAND

NUMPROY_LOC NUMPROY	LOCALIDAD
1	VAUDEVILLE
2	SERIES CITY
3	OUTLAND
4	MARTINSVILLE
5	ALPENHAGEN
6	SAN CLEMENTE
7	OUTLAND
8	MARTINSVILLE
9	ALPENHAGEN
10	OUTLAND

quiso restringir a un usuario particular, Harry Bean, a tener una autorización de acceso, solamente de lectura, a los atributos NUMPROY y LOCALIDAD. Esto podría realizarse creando una vista que sólo incluya los atributos NUMPROY y LOCALIDAD, tal como se muestra en la Figura 11.13. Entonces, se puede definir una contraseña para el acceso en lectura para NUMPROY_LOC, como se indica a continuación:

```
GRANT READ ACCESS ON NUMPROY_LOC TO HARRY BEAN
```

La forma general para las autorizaciones en SQL se obtiene utilizando la instrucción GRANT:

```
GRANT < lista de privilegios > ON < nombre de relación o de vista > TO
< lista de usuarios >
```

La lista de privilegios permite otorgar varios privilegios (e. g., de lectura, de eliminación y/o de actualización) en una sola instrucción.

Vistas y Seguridad en SQL. Considerando que el SQL ha devenido en la norma de *facto* para los lenguajes de base de datos relacionales y que proporciona diversas facilidades para utilizar las vistas para la seguridad, esta sección se centrará en el SQL como modelo. La sintaxis general de las instrucciones SQL que crean vistas es:

```
CREATE VIEW nombre_de_vista (lista de atributos deseados, si son
diferentes de la tabla base)
        AS consulta
```

Consideremos un ejemplo concreto que utiliza la relación base PERSONAL que tiene el esquema PERSONAL (ID, NOMBRE, DIRECCIÓN, HSALARIO, NUMDEPT). Supongamos que se quiere limitar el acceso del usuario U_1 únicamente a los empleados del Departamento 35 de la tabla PERSONAL. Esta vista se podría expresar como:

```
CREATE VIEW DEPT_35
     AS SELECT ID, NOMBRE, DIRECCIÓN, HSALARIO, NUMDEPT
          FROM PERSONAL
          WHERE NUMDEPT = 35
```

Los privilegios de lectura se otorgan a U_1 de la manera usual y, ahora, U_1 puede tener acceso a la vista DEPT_35 para ver el subconjunto requerido de la tabla base, PERSONAL. Aunque los valores de PERSONAL están almacenados realmente en el disco, cualquier vista derivada de PERSONAL (e. g., DEPT_35) se crea en el momento que se utiliza. Sin embargo, esto es transparente para U_1 y, en general, de ningún interés para el usuario.

Supongamos que U_1 se limita a una vista de PERSONAL que excluye la información salarial (HSALARIO). Esta vista se crearía de la manera siguiente:

```
CREATE VIEW PERS_SIN_SAL
     AS SELECT ID, NOMBRE, DIRECCIÓN, NUMDEPT
          FROM PERSONAL
```

Estos dos ejemplos demuestran la creación de vistas que son, respectivamente, subconjuntos fila y columna de la relación base PERSONAL. El siguiente ejemplo muestra cómo se crea una vista de modo que es un subconjunto fila y un subconjunto columna de la relación base PERSONAL.

```
CREATE VIEW DEPT_35
AS SELECT ID, NOMBRE, DIRECCIÓN, NUMDEPT
     FROM PERSONAL
     WHERE NUMDEPT = 35
```

Ahora U_1 puede tener acceso a una vista de PERSONAL que incluye ID, NOMBRE y DIRECCIÓN (pero no HSALARIO) de los empleados asignados al Departamento 35.

A algunos usuarios se les puede responsabilizar con el mantenimiento de los elementos de los datos y por eso se les permite tener acceso a las tablas del sistema SYSTABLES, de las cuales son responsables. Como las tablas del sistemas SYSTABLES son en sí mismas relaciones, también se pueden crear vistas para ellas. A continuación se presenta un ejemplo:

```
CREATE VIEW MIS_TABLAS
AS SELECT *
     FROM SYSTABLES
     WHERE CREATOR = USER
```

USER es una palabra clave que requiere la asignación de un valor durante la ejecución. Así, si el usuario U_1 introduce la instrucción

```
SELECT *
FROM MIS_TABLAS
```

el sistema ejecutará la consulta como si estuviera escrito

```
SELECT *
FROM SYSTABLES
WHERE CREATOR = U1
```

Se dice que las vistas como ésta son *dependientes del contexto*, ya que el resultado que se produce es función del contexto (U_1) en el que se está utilizando.

Las vistas se pueden construir también a partir de más de una relación base. Consideremos las relaciones base PERSONAL y PLANTA con los esquemas siguientes:

```
PERSONAL (ID, NOMBRE, DIRECCIÓN, HSALARIO, NUMDEPT, ID_PLANTA)
PLANTA (ID_PLANTA, CIUDAD, NUMDEPT).
```

Se quiere crear una vista que identifique cada empleado con la ciudad en la que trabaja como sigue:

```
CREATE VIEW LOCALIDAD_EMPLEADO
     AS SELECT ID, NOMBRE, CIUDAD
          FROM PERSONAL, PLANTA
          WHERE PERSONAL.ID_PLANTA = PLANTA.ID_PLANTA
```

A veces se puede permitir que un usuario tenga acceso a resúmenes de los datos que se mantienen en una relación base, pero no se le autoriza el acceso a los valores individua-

les. Por ejemplo, un usuario puede permitir únicamente el acceso al promedio de la tarifa salarial por hora de la relación PERSONAL. Esta restricción podría garantizarse creando la vista siguiente:

```
CREATE VIEW PROM_TARIFA_HR (ID, NOMBRE, TARIFAPROM, NUMDEPT)
     AS SELECT ID, NOMBRE, AVG (HSALARIO), NUMDEPT FROM PERSONAL
     GROUP BY NUMDEPT
```

Observe qué está sucediendo aquí. Se está creando un atributo en la vista (TARIFAPROM) que no existe en la tabla de la base, PERSONAL. Sus valores se crean en la instrucción SELECT especificando que los valores en el atributo HSALARIO se promedian según el número de departamento.

Aunque el uso de las vistas puede ser un medio efectivo para garantizar la seguridad, el sistema tiene que ser capaz de adaptarse a los cambios en los requisitos a través del tiempo. SQL proporciona esta capacidad mediante el uso de los privilegios GRANT y REVOKE [1]. Aquí se presentan algunos ejemplos:

```
GRANT SELECT ON TABLE PERSONAL TO JOHN, SYLVIA
```

Esto significa que John y Sylvia están autorizados a ejecutar cualquier operación SELECT sobre la tabla PERSONAL.

```
GRANT SELECT, UPDATE (HSALARIO) ON TABLE PERSONAL TO HARVEY
```

Esto significa que Harvey tiene el derecho de ejecutar las operaciones SELECT sobre PERSONAL, así como también el derecho de actualizar el atributo HSALARIO.

```
REVOKE SELECT ON TABLE PERSONAL FROM JOHN
```

Esto significa que, en adelante, John no está autorizado a ejecutar las operaciones SELECT sobre la tabla PERSONAL.

Estos ejemplos ilustran algunas posibilidades que brindan los privilegios para otorgar o revocar privilegios de autorización. El listado de los privilegios, tanto para las relaciones de la base como para las vistas, incluye SELECT, UPDATE, DELETE e INSERT.

La opción GRANT puede pasar de un usuario a otro en cascada. Por ejemplo, si John tiene derecho a otorgar la autoridad A a otro usuario Sylvia, entonces Sylvia tiene derecho a otorgar la autoridad A a otro usuario Dale y así sucesivamente. Considere el ejemplo siguiente.

```
John:
GRANT SELECT ON TABLE PERSONAL TO SYLVIA WITH GRANT OPTION
Sylvia:
GRANT SELECT ON TABLE PERSONAL TO DALE WITH GRANT OPTION
```

Mientras un usuario haya recibido la opción GRANT OPTION, puede conferir la misma autoridad a otros.

Si luego John desea revocar la opción GRANT OPTION, podría hacerlo de la manera siguiente:

```
REVOKE SELECT ON TABLE PERSONAL FROM SYLVIA
```

Esta revocación se aplicaría a Sylvia, así como también a cualquiera a quien ella le hubiera transferido la autoridad, y así sucesivamente.

[1] Otorgar y revocar (N. del T.).

Encriptación

cifrado (*encriptación*, en la jerga informática). Convertir un texto legible en un texto ilegible mediante el uso de un algoritmo; utilizado para proteger datos sensibles.

Las diversas medidas de autenticación y de autorización que son las normas para proteger el acceso a las bases de datos pueden no ser adecuadas para datos altamente sensibles. En tales casos, puede ser deseable **cifrar** *(to encrypt)* los datos. Los datos cifrados no pueden ser leídos por un intruso, a menos que conozca el método de cifrado. Importantes investigaciones se han dedicado al desarrollo de los métodos de cifrado. Algunos son tan simples que son fáciles de descifrar. Otros son muy difíciles de descifrar y proporcionan un alto nivel de protección.

Primero mostraremos un esquema simple de cifrado. Después mostramos un método más complejo, pero más seguro. Una fuente provechosa de este material para el lector interesado se puede encontrar en Hoffman (1979).

texto legible. Texto que se puede leer.

Método de sustitución simple. Suponga que deseamos cifrar el mensaje (**texto legible** [5])

```
Think snow.
```

Un método de sustitución simple podría ser el desplazamiento de cada letra a su sucesor inmediato en el alfabeto. Se supone que el *espacio* aparece inmediatamente antes de la letra *a* y que sigue a la letra *z*. "Think snow" se cifra (**texto encriptado** [6]) a

texto cifrado. Texto legible cifrado.

```
uijolatopx.
```

Si un intruso únicamente ve el mensaje "uijolatopx," probablemente no tendría información suficiente para romper el código. Sin embargo, si se examina un número grande de palabras, estadísticamente es posible determinar la frecuencia con que aparecen los caracteres y, por medio de ello, romper fácilmente el código. Los mejores esquemas de cifrado utilizan una clave de cifrado, como se demuestra en la sección siguiente.

Método de sustitución polialfabética. Suponga que queremos cifrar el mismo mensaje, pero ahora damos la clave de cifrado, digamos, "securityse." Procedemos como se indica a continuación:

1. Alineamos la clave debajo del texto legible, repitiéndola tantas veces como sea necesario para "cubrirlo" completamente. En este ejemplo, tendríamos

   ```
   Think snow
   securityse.
   ```

2. Supongamos que el *espacio* ocupa la vigésimo séptima, y última, posición en el alfabeto. Para cada carácter, sumamos la posición alfabética del carácter del texto legible y la del carácter de la clave, dividimos por 27 y guardamos el resto. Reemplazamos el carácter del texto legible con el carácter encontrado en la posición calculada en el resto. Para este ejemplo, la *T* se encuentra en el vigésimo lugar en el alfabeto, mientras que la *s* se encuentra en la decimonovena posición. Así,

   (20 + 19) = 39. El resto de la división por 27 es 12. (Este proceso se denomina *división módulo* 27.)

L es la letra en la duodécima posición en el alfabeto. Así, la letra *T* en el texto legible se cifra como la letra *L* en el texto cifrado.

---

[5] *Plaintext* (N. del T.).
[6] *Ciphertext* (N. del T.).

Este método es aún demasiado simple para ser de uso amplio, pero sirve para ilustrar la estrategia general que se utiliza en las aplicaciones.

▼ Recuperación de la base de datos

Debido a que la información almacenada sobre los medios de cómputo está sujeta a la pérdida o a la corrupción ocasionada por una gama amplia de sucesos, es importante proporcionar medios para restaurar los datos correctos en la base de datos. No siempre es posible la restauración de la base de datos cabalmente al mismo estado en que se encontraba en el momento del fallo de sistema, pero los procedimientos de recuperación de la base de datos pueden restablecerla al estado en que se encontraba poco tiempo antes del fallo e identificar el estado del procesamiento de la transacción en el momento del fallo. Con esta capacidad, las transacciones no procesadas pueden ejecutarse contra la base de datos restablecida para regresar a un estado actual completo.

Fuentes de fallo

Una clasificación útil de los tipos de fallo incluye los siguientes:

1. Errores del sistema. El sistema ha entrado en un estado indeseable, como el interbloqueo *(deadlock)*, que evita que el programa continúe con el procesamiento normal. Este tipo de fallo puede o no provocar corrupción en los archivos de datos.
2. Fallos del hardware. Dos de los tipos más comunes de fallos del hardware son el fallo del disco y la pérdida de la capacidad de transmisión en un enlace de transmisión. En el primer caso, la causa está normalmente en que la cabeza de lectura/escritura del disco entra en contacto físico con la superficie del disco (una "rotura de la cabeza").
3. Errores lógicos. Las condiciones más comunes que pueden impedir que un programa continúe con su ejecución normal son los datos erróneos o inexistentes.

En la sección que sigue se discuten los procedimientos de recuperación que son apropiados para diferentes tipos de fallos.

Procedimientos de recuperación

La integridad de los datos exige que una transacción esté en uno de los dos estados siguientes:

1. *Abortada (Aborted).* Una transacción puede no siempre completar su proceso exitosamente. Para asegurar que la transacció n incompleta no afecte el estado consistente de la base de datos, dichas transacciones tienen que abortarse, restaurando la base de datos al estado en que estaba antes de que la transacción en cuestión comenzara su ejecución. Tal restauración se logra mediante la operación de *rollback* (que se discutirá en breve).
2. *Cerrada (Committed).* Una transacción que completa exitosamente su procesamiento se dice que está *cerrada*. Una **transacción cerrada** deja la base de datos en un estado consistente.

transacción cerrada. Una transacción que completa exitosamente todas sus acciones.

El diario de operaciones *(log)* [7] juega un papel clave en la recuperación de los fallos. El diario de operaciones es un historial de todos los cambios hechos a la base de datos, así como también del estado de cada transacción. Obviamente, es importante que los datos del

[7] En ocasiones se usará sólo diario para mayor brevedad (N. del T.).

diario de operaciones no se pierdan ni se destruyan. Por consiguiente, la información del diario de operaciones se guarda en un «almacenamiento estable» mítico que sobreviva a todos los fallos. En la práctica, esto se logra manteniendo varias copias en disco.

Una estrategia de recuperación puede desarrollarse utilizando uno de los dos enfoques siguientes, registrar las actualizaciones diferidas o registrar las actualizaciones inmediatas. Los puntos de chequeo *(checkpointing)* proporcionan eficiencia adicional.

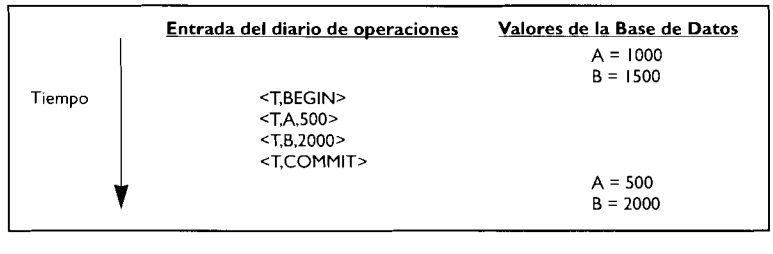
Diario de Operaciones con Actualizaciones Diferidas. El registro de las actualizaciones diferidas funciona como se indica a continuación:

1. Cuando comienza una transacción T, un registro "<T,>", o equivalentemente, "<T,START>", se escribe en el diario de operaciones.
2. Durante la ejecución de la transacción T, la escritura de un valor nuevo cualquiera, a_i, para el atributo A, denotado por "WRITE(A,a_i)," provoca la escritura de un nuevo registro en el diario de operaciones.
3. Cada registro del tipo descrito en (2) está formado por los campos siguientes:
   a. El nombre de la transacción, T.
   b. El nombre del atributo, A.
   c. El nuevo valor del atributo, a_i.
4. Si todas las acciones que comprende T se ejecutan exitosamente, se dice que T se cierra parcialmente, y se escribe el registro "<T,COMMIT>" en el diario de operaciones. Después que la transacción T se cierra parcialmente, se utilizan los registros asociados con T en el diario de operaciones para ejecutar las operaciones de escritura en los registros apropiados de la base de datos.

Ilustramos esto utilizando, una vez más, el ejemplo del pago de un cliente. Recuerde que se está haciendo un pago de $500 sobre una cuenta.

En la Figura 11.14 se muestran las acciones comprendidas en esta transacción. Las entradas correspondientes del diario de operaciones se muestran en la Figura 11.15. Al utilizar el diario de operaciones, el SGBD puede gestionar cualquier fallo que no provoque la pérdida de la información del propio diario de operaciones. La prevención de la pérdida del diario de operaciones se resuelve mediante su duplicación en más de un disco. Como la probabilidad correspondiente a la pérdida del diario es muy pequeña, este método se considera comúnmente como un método de *almacenamiento estable.*

T: READ (A, a_1)	Leer el balance del cliente actual
a_1: = a_1 – 500	Reducir el balance deudor en $500
WRITE (A, a_1)	Escribir el nuevo balance
READ (B, b_1)	Leer el balance efectivo actual
b_1: = b_1 + 500	Incrementar el balance de la cuenta en $ 500
WRITE (B, b_1)	Escribir el nuevo balance

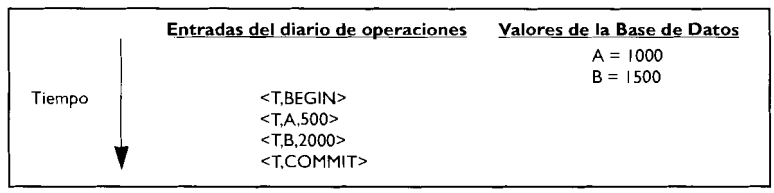


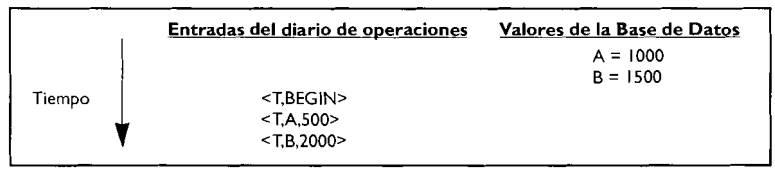
Después que ocurre un fallo, el SGBD examina el diario de operaciones para determinar qué transacciones se necesita rehacer *(redo)*. Una transacción T tiene que rehacerse si el diario de operaciones contiene el registro "<T,BEGIN>" y el registro "<T, COMMIT>." Esto es, la base de datos puede haberse corrompido, pero el procesamiento de la transacción se completó y los nuevos valores para los elementos de los datos correspondientes están contenidos en el diario de operaciones. Esto excluye la necesidad de reprocesar la transacción. El diario se utiliza para restablecer el estado del sistema utilizando un procedimiento REDO(T), que fija el valor de todos los elementos de los datos actualizados por la transacción T a los nuevos valores que estaban registrados en el diario de operaciones.

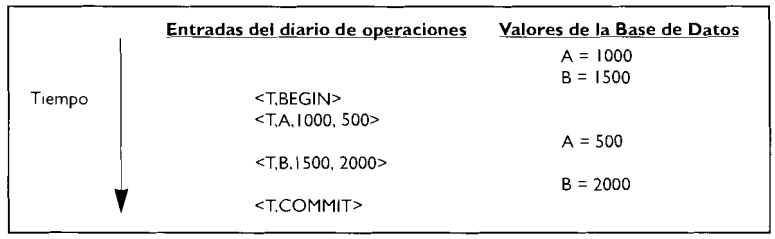
Como ilustración, supongamos que el fallo ocurrió simplemente después que el registro "<T, COMMIT>" entró en el diario de operaciones y antes que se escribieran los registros actualizados en la base de datos. El diario aparece como se muestra en la Figura 11.16. Cuando se comienza a restablecer el sistema, se necesita efectuar la acción de recuperación para T, ya que "<T, COMMIT>" aparece en el diario. La operación REDO se ejecuta, dando como resultado que los valores \$500 y \$2,000 se escriben en la base de datos como los valores actualizados de A y B.

Ahora supongamos que el fallo ocurre simplemente antes de la ejecución de la acción WRITE(B,b_1). El diario de operaciones en el momento del fallo se muestra en la Figura 11.17. Cuando se comienza a restablecer el sistema, no es necesaria ninguna acción, ya que en el diario no aparece ningún registro COMMIT para T. Los valores de A y B en la base de datos permanecen como \$1,000 y \$1,500. En este caso, la transacción tiene que reiniciarse.

Diario de Operaciones con Actualizaciones Inmediatas. Un método alternativo que utiliza un diario de operaciones para la recuperación consiste en hacer todas las actualizaciones de la base de datos inmediatamente y llevar un registro de todos los cambios en el diario. Al igual que en el método anterior, si ocurre un fallo, el diario se utiliza para restablecer el estado de la base de datos a un estado previo consistente. Similarmente, cuando comienza una transacción, en el diario de operaciones se escribe el registro "<T,BEGIN>". Durante la ejecución de T, cualquier operación "WRITE(A,a_i)" está *precedida* por la escritura de un nuevo registro en el diario de operaciones. (Recuerde que en la estrategia previa, no se aplicaba ninguna operación de escritura a la base de datos hasta que T se cerrara parcialmente.) Cada registro del diario de operaciones se escribe de la manera siguiente <T,A,a_i,a_j,>, donde A es el nombre del atributo, a_i es el valor viejo del atributo y a_j es el valor nuevo del atributo. Cuando la T se cierra parcialmente, "<T,COMMIT>" se escribe en el diario de operaciones.







El método de recuperación para este tipo de registro requiere dos procedimientos: (1) REDO(T) (la misma operación que antes); (2) UNDO(T), que restablece los valores de todos los atributos actualizados por T a sus valores viejos.

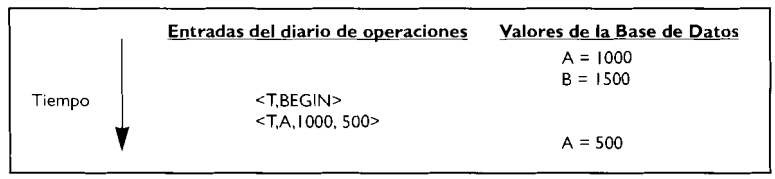
Estos procesos se ilustran con el mismo ejemplo. Primero, observe la Figura 11.18. Se presenta cómo se vería el diario de operaciones después de la ejecución de T. Después que ha ocurrido un fallo, el sistema de recuperación examina el diario para identificar aquellas transacciones que necesitan ser deshechas (UNDO) o rehechas (REDO). La lógica de este proceso se indica a continuación:

1. Se necesita deshacer T si el diario contiene el registro "<T,BEGIN>" pero no contiene "<T,COMMIT>." Se necesita que los valores viejos de los elementos de los datos afectados sean restablecidos y que la transacción se reinicie.
2. Se necesita rehacer T si el diario contiene los dos registros precedentes. No tiene que reiniciarse la transacción.

Se ilustrarán ambas situaciones.

Supongamos que el sistema falla de nuevo justamente antes de la acción "WRITE(B,b_1)" en la Figura 11.14. La Figura 11.19 ilustra cómo se vería el diario en el momento del fallo. Cuando el sistema vuelve a ser operable, encuentra el registro "<T,BEGIN>" pero no su correspondiente "<T,COMMIT>". Esto significa que debe deshacerse la transacción, luego se ejecuta "UNDO(T)". Ello restablece el valor de A a \$1.000 y la transacción puede reiniciarse.

Ahora, supongamos que el problema ocurre simplemente después que "<T,COMMIT>" se escribe en el diario de operaciones, pero antes que los nuevos valores se escriban en la base de datos. En la Figura 11.20 se muestran las entradas correspondientes al diario. Cuando el sistema vuelve a ser operable otra vez, una lectura del diario de operaciones muestra los registros correspondientes "<T,BEGIN>" y "<T,COMMIT>". De este modo, puede ejecutarse "REDO(T)", después de lo cual los valores de A y B son \$500 y \$2. 000.



	Entradas del diario de operaciones	Valores de la Base de Datos
		A = 1000
		B = 1500
Tiempo	<T,BEGIN>	
	<T,A,1000, 500>	
		A = 500
	<T,B,1500, 2000>	
		B = 2000
	<T,COMMIT>	

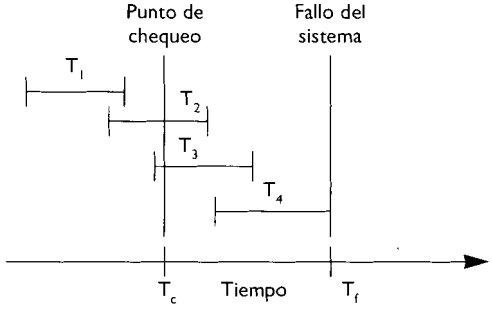
Puntos de comprobación (Checkpointing). De los procedimientos precedentes, uno podría concluir que la recuperación solamente requiere examinar el diario de operaciones para las entradas correspondientes a la transacción más reciente o a algunas transacciones recientes. En principio, no debe haber límite con relación a cuánto debe mirar el sistema hacia atrás en el diario, ya que los errores pueden haber comenzado con la primera transacción. Esto puede provocar un gran consumo y despilfarro de tiempo. Una mejor manera consiste en encontrar hacia atrás un punto suficientemente alejado como para asegurar que cualquier elemento antes de ese punto se ha escrito correctamente y se ha almacenado sin riesgo. Este método se denomina de **puntos de comprobación**.

puntos de comprobación. Copias de la base de datos en momentos predeterminados durante el procesamiento; la recuperación de la base de datos comienza o termina en el punto de chequeo más reciente.

Durante la ejecución, el SGBD mantiene el diario de operaciones como se ha descrito, pero periódicamente establece puntos de comprobación de la manera siguiente:

1. Detiene temporalmente el inicio de cualquier transacción nueva hasta que se cierren o se aborten todas las transacciones activas.
2. Realiza una copia de seguridad de la base de datos.
3. Escribe todos los registros del diario de operaciones que radican actualmente en la memoria operativa en un almacenamiento estable.
4. Añade al final del diario de operaciones un registro que indica la ocurrencia de un punto de chequeo; entonces lo escribe en el disco.

Supongamos que se utiliza un diario de operaciones con actualizaciones inmediatas y consideremos el gráfico de tiempo para las transacciones T_1 - T_4 mostrado en la Figura 11.21. Cuando el sistema falla (t_f), se necesita examinar el diario de operaciones hacia atrás únicamente hasta el punto de comprobación más reciente (t_c). T_1 es correcta, a menos que haya habido un fallo en el disco que la destruyera a ella (y posiblemente) a otros registros con anterioridad al último punto de comprobación. En ese caso, la base de datos se carga de nuevo desde la copia de seguridad que se hizo en el punto de chequeo anterior. En



ambos casos. T_2 y T_3 se rehacen (REDO) a partir del diario y T_1 se deshace (UNDO) también a partir del diario de operaciones.

El conocimiento de los elementos administrativos y de control es necesario para la gestión efectiva de un sistema de base de datos. El mantenimiento de la seguridad y de la integridad de la base de datos es a la vez complejo y esencial, debido al acceso de múltiples usuarios en forma concurrente a los sistemas de base de datos.

Este capítulo se dedicó primero a establecer la diferencia entre la administración de los datos y la administración de la base de datos, concentrándose en la administración de la base de datos y en sus objetivos de mantener la integridad, la seguridad y la disponibilidad de la base de datos. La integridad de los datos, que es el mantenimiento de su exactitud y su fiabilidad, se garantiza desarrollando controles de seguridad de los datos. La disponibilidad de los datos se aumenta desarrollando procedimientos de seguridad y de recuperación, los que pueden incluir un plan para desastres.

Luego se demostraron algunas capacidades de los lenguajes comerciales de consulta, tales como el SQL y el QBE, en cuanto a la integridad. Estas capacidades no proporcionan aún todas las características deseadas, pero se espera que esta situación mejore en el futuro próximo.

Otra manera útil de mantener la consistencia de la base de datos es la transacción atómica, que requiere que todo el procesamiento de la transacción se complete o que la transacción se interrumpa. Cuando se usa con un diario de operaciones *(log)*, esto asegura que los fallos operativos no provoquen el almacenamiento de datos erróneos.

El procesamiento concurrente puede corromper la base de datos. Los procedimientos simples de bloqueo *(locking)* pueden impedir el procesamiento concurrente, pero pueden restringir mucho la eficiencia operativa del sistema de base de datos. Métodos tales como el bloqueo en dos fases están disponibles para limitar los problemas asociados con los bloqueos simples, a la vez que mantienen la integridad de la base de datos.

La seguridad de la base de datos, en principio, está relacionada con la determinación de quién tiene acceso legítimo a qué datos y, entonces, con la seguridad se refuerza la legitimidad. La autenticación se refiere a los métodos para restringir el acceso al sistema. La autorización se refiere a los métodos de control de los recursos, a los que se puede tener acceso una vez que se haya tenido acceso al sistema, así como qué se puede hacer con esos recursos.

En los sistemas de base de datos relacionales, el uso de las vistas puede ser una manera efectiva de restringir lo que el usuario ve y manipula, sin tener que alterar las relaciones almacenadas. Utilizando SQL, se mostraron diferentes maneras en que se pueden aplicar las vistas.

El cifrado *(encryption)* permite que los datos sensibles puedan almacenarse y transmitirse de forma ininteligible, excepto para sus usuarios legítimos.

Todos los sistemas de bases de datos deben tener procedimientos seguros de copias de seguridad y de recuperación para evitar ineficacias y, aún más, pérdidas catastróficas. Estos métodos deben probarse periódicamente para asegurar que funcionen completamente y que sean fiables. Los diarios de operaciones de transacciones con actualizaciones diferidas o inmediatas, los puntos de comprobación y las copias de seguridad *(backup)* de la base de datos son elementos esenciales para su seguridad y recuperación.

1. Defina con sus palabras cada uno de los términos siguientes:
   a. administrador de los datos
   b. integridad de los datos

c. seguridad de los datos
d. control de integridad
e. transacción atómica
f. diario de operaciones
g. interbloqueo
h. bloqueo en dos fases
i. bloqueo de escritura
j. cifrado (*encriptación*)
k. texto cifrado
l. punto de comprobación

2. ¿Cuáles son las funciones principales del ABD? ¿Cuáles son los objetivos del ABD?

3. ¿Cuál es la diferencia entre un bloqueo de lectura y un bloqueo de escritura?

4. Compare y contraste el significado de los términos *seguridad* e *integridad*. Dé un ejemplo de cómo la base de datos podría afectarse por una violación de la integridad y por una violación de la seguridad.

5. Describa el papel del administrador de la base de datos en la implementación de normas y procedimientos.

6. ¿Cuál es el significado de copia de seguridad y de recuperación?

7. Explique cómo la concurrencia puede conducir a una inconsistencia en la base de datos.

8. Desarrolle un ejemplo, distinto al que se encuentra en el texto, que muestre cómo un control de bloqueo simple puede conducir a un interbloqueo *(deadlock)*.

9. Estudie dos formas en las cuales se puede detectar el interbloqueo. ¿Cuáles son sus ventajas y desventajas?

10. Explique cómo trabaja el bloqueo en dos fases.

11. Establezca la diferencia entre la autorización y la autentificación.

1. Haga corresponder los términos siguientes con sus definiciones:

—*administrador de la base de datos*	a. Los medios mediante los cuales una base de datos que ha sido corrompida por desperfectos del sistema puede restablecerse a un estado consistente
—*texto legible*	b. Dirigente cuyas responsabilidades se centran en la administración de los aspectos técnicos del sistema de base de datos
—*procesamiento concurrente*	c. Ocurre cuando dos o más transacciones solicitan el acceso concurrentemente al mismo registro de la base de datos prácticamente al mismo tiempo
Π*procedimientos de recuperación de*	d. Un subconjunto restringido de una relación almacenada *la base de datos*

—*transacción*

—*transacción abortada*

—*bloqueo*

—*bloqueo de lectura*

—*detección de interbloqueo*

—*vista*

e. El usuario tiene el derecho para leer un registro determinado

f. Un programa cuya ejecución preserva la consistencia de la base de datos

g. Texto que se puede leer

h. Transacción que se cancela antes que los cambios se apliquen a la base de datos

i. En el procesamiento de transacciones, impide el acceso a un registro de la base de datos por una segunda transacción hasta que la primera transacción haya completado todas sus acciones

j. Una comprobación periódica del SGBD para determinar si la cola de espera de algún recurso excede un límite predeterminado

2. Utilizando un ejemplo, muestre un problema de integridad de los datos que puede ser ocasionado por un proceso concurrente de actualización.

3. Ilustrar cómo trabaja el bloqueo en dos fases utilizando un ejemplo.

4. Dos transacciones se dice que son serializables si se pueden ejecutar en paralelo (intercaladas), de tal forma que sus resultados sean idénticos a los que se lograrían si una de las transacciones se ejecuta completamente antes del inicio de la otra. Considere las dos transacciones intercaladas siguientes y suponga que una condición de consistencia requiere que A o B deben ser siempre iguales a 1. Suponga que $A = B = 1$ antes de la ejecución de estas transacciones.

```
T1                                  T2
Leer_elemento(A)
                                    Leer_elemento(B)
                                    Leer_elemento(A)
Leer_elemento(B)
Si A = 1
entonces B : = B + 1
                                    Si B = 1
                                      entonces A : = A + 1
                                    Escribir_elemento(A)
Escribir_elemento(B)
```

¿Quedará satisfecho el requisito de consistencia? Justifique su respuesta. ¿Existe un esquema de tratamiento intercalado que garantiza la serialidad? Si es así, demuéstrelo. Si no, explique por qué.

5. Utilizando el método polialfabético de sustitución y la clave de cifrado, SECURITY, cifre el texto legible "SELL ALL STOCKS."

6. Suponga que se tiene un diario de operaciones con actualizaciones inmediatas, cree las entradas del diario correspondientes a las siguientes acciones de una transacción:

```
T: read (A, a1)          Leer el balance del cliente actual
a1: = a1 - 800           Debitar la cuenta en $800
write (A, a1)            Escribir el nuevo balance
read (B, b1)             Leer el balance actual de las cuentas de pagos
b1: = b1 + 800           Acreditar el balance de la cuenta en $800
write (B, b1)            Escribir el nuevo balance
```

7. Suponga que en (6) ocurre un fallo justamente después que se haya escrito el registro del diario de operaciones para la acción

$$\underline{\text{write}}\ (B, b_1)$$

   a. Muestre el contenido del diario de operaciones en el momento del fallo.
   b. ¿Qué acción es necesaria y por qué? ¿Cuáles son los valores resultantes de A y B?

8. Suponga que en (6) ocurre un fallo justamente después que el registro "<T,COMMIT>" se escriba en el diario de operaciones.
   a. Muestre los contenidos del diario en el momento del fallo.
   b. ¿Qué acción es necesaria y por qué? ¿Cuáles son los valores resultantes de A y B?

9. En el diario de operaciones para la recuperación, considere las entradas siguientes en el momento del fallo del sistema de base de datos:

```
<T1, BEGIN>              <T1, BEGIN>              <T1, BEGIN>
<T1, A, 500,395>         <T1, A, 500,395>         <T1, A, 500, 395>
<T1, B, 800,950>         <T1, B, 800,950>         <T1, B, 800, 950>
<T1, COMMIT>             <T1, COMMIT>
<T2, BEGIN>              <T2, BEGIN>
<T2, C, 320, 419>        <T2, C, 320, 419>
<T2, COMMIT>
```

(A) *(B)* *(C)*

   a. Suponga un diario con actualización diferida. Describa para cada caso (A, B, C) qué acciones de recuperación son necesarias y por qué. Indique cuáles son los valores para los atributos dados después que las acciones de recuperación se completen.
   b. Repita el requisito de (a) suponiendo un diario con actualización inmediata.

1. Realice un breve estudio de investigación de qué métodos utilizan las grandes firmas de contabilidad para hacer auditorías de los sistemas de bases de datos.

2. Compare y contraste los controles de integridad disponibles en SQL y en QBE.

3. Compare y contraste los controles para la seguridad de los datos disponibles en SQL y en QBE.

CAPÍTULO

12

SISTEMAS DE BASES DE DATOS DISTRIBUIDAS

¿Por qué las bases de datos distribuidas?
Un modelo general de SBDD
Diseño de sistemas de bases de datos distribuidas
Estrategias y objetivos
Transparencia local
Transparencia de la duplicación
Independencia de la configuración
SGBDs no homogéneos
Duplicación de datos
Partición de la base de datos
Fragmentación de los datos
Distribución de archivos no-fragmentados
Procesamiento distribuido de consultas
Semirreuniones (*semijoins*)
Integridad de los datos en los sistemas de bases de datos distribuidas
Protocolo de cierre de dos-fases
Bloqueo distribuido
Ejemplo 1
Bloqueo distribuido de dos-fases
Ejemplo 2
Marcas de tiempo
Ejemplo 3
Recuperación de la base de datos
Sistemas cliente/servidor
Conceptos generales
Aplicaciones de bases de datos
SQL server para Windows NT
Resumen
Preguntas de comprobación
Problemas y ejercicios
Problemas y cuestiones profesionales

Cordelia Molini y Reggie están sentadas almorzando y hablando sobre el progreso que se ha dado en la implementación de sistemas de información en la IPD.

Reggie comentó: "Pienso que estamos haciendo un buen trabajo. Nosotros tenemos en casa la última tecnología relacional, hemos educado a los usuarios en cada área funcional del negocio y hay evidencias de que nuestros sistemas de información se usan cada vez más para apoyar la toma de decisiones en todas partes dentro de la compañía."

"En general, estoy de acuerdo contigo", respondió Cordelia. "Pero siento que aún hay mucho por hacer. Muchos de nuestros jefes se encuentran en sitios remotos en todas partes del país y he oído rumores de que a ellos les gustaría tener control sobre las partes del sistema de base de datos que tienen que ver con sus operaciones. De hecho, una de las jefas me comentó que no entendía por qué los datos deben mantenerse en la base de datos colectiva cuando era principalmente actualizada y usada por ella."

"Suena como si nosotros debiéramos examinar la distribución de nuestra base de datos," sugirió Reggie. ¿Qué tipo de cambios habría que hacer si nos movemos en esa dirección?"

"Bueno", dijo Cordelia, "Como primera medida, tendríamos que decidir qué datos pueden mantenerse localmente y qué datos deberían hacerse centralmente. Y si aún quisiéramos hacer que ciertos datos corporativos estén disponibles en sitios remotos, nuestros sistemas para controlar el acceso a los datos tendrían que ser más sofisticados para permitir que sean ejecutadas consultas que requieran datos de más de un sitio. Hay otras cuestiones, pero con nuestro historial de éxitos creo que estamos en una buena posición para comenzar a pensar sobre un sistema de base de datos distribuida".

Este capítulo brinda una introducción a la excitante área de los sistemas de bases de datos distribuidas (SBDD, DDS en inglés). Debido a que los SBDD son aún un área en desarrollo, están jugando ya un papel importante en operaciones de negocios. En el futuro, los SBDD deben convertirse en una tecnología necesaria para todas las organizaciones. Con los extraordinarios avances que están ocurriendo en las tecnologías de las comunicaciones, las posibilidades son impresionantes. En este capítulo aprenderá:

- ¿Por qué una SBDD es valioso para una organización?
- Los fundamentos tecnológicos y la terminología de un sistema de base de datos distribuida (SBDD).
- Elementos del diseño de un SBDD.
- Estrategias y objetivos que son importantes para los SBDD.
- Configuraciones opcionales que están disponibles para los SBDD, junto con sus ventajas y desventajas.
- Los problemas de control fundamentales asociados con los SBDD y los métodos de tratar con ellos.
- Los fundamentos de los sistemas cliente/servidor.

▼ ¿Por qué sistemas de bases de datos distribuidas?

base de datos distribuida. Una base de datos segmentada ubicada en distintas localizaciones unidas todas ellas a través de una red.

Con la amplia prevalencia de los sistemas de bases de datos centralizados, los usuarios y los programas de aplicación acceden a la base de datos desde sitios locales, así como desde localidades remotas. En contraste, una **base de datos distribuida** (*distributed database*) no se almacena completamente en una localidad central, sino que se distribuye en una red de localidades que pueden estar geográficamente separadas y conectadas por enlaces de comunicaciones. Cada localidad tendrá su propia base de datos y está capacitada para acceder a los datos de otras localidades.

sistema de base de datos distribuida. Una colección de localidades, en cada una de las cuales opera un sistema de base de datos local y que pueden participar en la ejecución de transacciones que acceden datos de varias localidades.

Hay varias razones para desarrollar y usar un **sistema de bases de datos distribuida** entre las que se incluyen las siguientes:

1. A menudo, las organizaciones tienen ramales o divisiones en diferentes localidades. Para una localidad dada, L, pudiera haber un conjunto de datos que use frecuentemente y quizás exclusivamente. Además L puede usar datos que se usen con más frecuencia en otra localidad L'.

   Por ejemplo, en un negocio de venta al por menor cada tienda se puede beneficiar de su propia base de datos de inventarios, ventas, cuentas de clientes y empleados. Durante las operaciones de negocios, las transacciones se procesan convenientemente en la base de datos local. Al final de las operaciones del día se debe trasmitir un resumen con los resultados a las oficinas centrales de la corporación en donde se mantiene una base de datos general. Periódicamente cada tienda puede beneficiarse de tener acceso a la base de datos general y poder comparar sus ventas y ganancias con las de las demás tiendas como medida de su rendimiento.

   En un sistema de base de datos centralizado cada sitio tendría que estar usando enlaces de comunicación con la base de datos para ambos tipos de información, por lo que las comunicaciones pueden convertirse en un cuello de botella importante.
2. Permitir que cada sitio almacene y mantenga su propia base de datos facilita el acceso inmediato y eficaz a los datos que se usan con más frecuencia. Tales datos podrían usarse en otros sitios, pero con menos frecuencia. Datos similares almacenados en otras localidades también podrían accederse si fuese necesario.
3. Las bases de datos distribuidas pueden mejorar la fiabilidad. Si los computadores de un sitio fallan, o si se queda fuera de servicio algún enlace de comunicación, el resto de la red puede seguir funcionando. Es más, si los datos están duplicados en más de un sitio, los datos necesarios pueden estar disponibles en otro lugar que aún se mantiene operativo.
4. Permitir el control local de los datos que se usan con más frecuencia en un sitio puede mejorar el grado de satisfacción de los usuarios con relación al sistema de bases de datos. Esto es, las bases de datos locales pueden reflejar mejor la estructura administrativa local y, por lo tanto, brindar un mejor servicio a las necesidades de la gestión.

Estas son algunas de las ventajas que se obtienen comúnmente de los sistemas de bases de datos distribuidas. Sin embargo, hay circunstancias en las que un sistema distribuido puede no presentar ventajas. En situaciones donde tenga lugar una gran cantidad de comunicación entre los sitios, el sobrecosto en que se puede incurrir por las coordinaciones y las tareas de control puede degradar severamente el rendimiento. Esto puede ser especialmente cierto cuando se mantienen duplicados de los datos en sitios diferentes y, por consiguiente, se necesitan recursos extra para asegurar que las actualizaciones que se hagan de modo concurrente sean consistentes.

Resumamos algo este último aspecto. La ventaja de replicar los datos es la velocidad que se gana en procesamiento en los sitios en donde se mantengan dichos duplicados, así como en que permite disponer de copias de seguridad por si ocurre un fallo en alguna localidad. Tales duplicaciones implican usar almacenamiento extra y el procesamiento de las transacciones y la recuperación de datos se hace más difícil. Cuando se procese una transacción ésta puede provocar un requisito de leer y actualizar datos en los diferentes sitios y en transmitir los mensajes respectivos entre ellos. Después de que cerrarse una transacción, el sistema de bases de datos distribuida (SBDD) debe asegurar que todos los sitios relevantes hayan completado su procesamiento. Sólo si el procesamiento concluyó normalmente en cada lugar debería cerrarse la transacción. De lo contrario, la transacción debería deshacerse en cada uno de los sitios participantes. Por último, conseguir que los datos estén disponibles a los usuarios a través de una red hace que el problema de seguridad sea inherentemente más complejo en los sistemas distribuidos que en los sistemas centralizados.

datos locales. Datos en mismo sitio o localidad con su propia base de datos.

datos globales. Datos que se mantienen en una base de datos cuya ubicación es diferente para al menos uno de los usuarios.

enlace. Canales de comunicación entre dos sitios de la red y que proveen las capacidades de transferencia de datos.

sistema de gestión de bases de datos distribuida (SGBDD). El software que gestiona el sistema de bases de datos distribuida.

System for Distributed Databases (SSD). SGBDD comercializado por Computer Corporation of America.

R*. Un SGBDD comercializado por IBM.

Distributed INGRESS. Un SGBDD comercializado por Relational Technology.

transacción. La ejecución de un programa de usuario que interactúa con el SGBDD.

agente. Un proceso que coopera para ejecutar una transacción.

transacción local. Una transacción que requiere un único agente.

transacción global. Una transacción que requiere varios agentes.

Mientras que estas limitaciones sugieren precaución en la planificación e implementación de los SBDDs, debe decirse que la tecnología para mitigar tales limitaciones mejora con rapidez. Si se quiere que las firmas sean responsables, productivas y competitivas, los SBDDs deberán seleccionarse de forma creciente, como el componente central de una estrategia efectiva para los sistemas de información.

▼ Un modelo general de SBDD (*DDS*)

Un SBDD está formado por una colección de sitios, cada uno de los cuales opera un sistema de base de datos para el procesamiento de las actividades que sólo requieren de **datos locales**. Adicionalmente cada lugar puede procesar transacciones que requieren datos que están almacenados en otros sitios (**datos globales**). Esto requiere que las bases de datos locales sean capaces de comunicarse datos entre sí. Las conexiones de comunicación que proporcionan las capacidades necesarias de transferencia se llaman **enlaces** (*links*). La estructura de enlace brinda la arquitectura básica de un **sistema de gestión de base de datos distribuida (SGBDD)** [1], que es el sistema de software que gestiona las bases de datos distribuidas.

Varios SGBDDs han sido implementados o están en desarrollo. Entre éstos se incluyen **System for Distributed databases (SDD)** (Computer Corporation America). **R*** (IBM) y **Distributed INGRESS** (Relational Technology). El popular DB2 de IBM incluye algún soporte de distributividad, lo mismo que INGRESS/STAR y Oracle7.

Un SGBDD es un sistema compuesto de varios SGBDs operando en sitios locales y que están conectados por facilidades de manejo de mensajes. El diccionario de datos del SGBDD incluye la información usual necesaria para la gestión de los datos junto con la información concerniente a cada localidad, a la duplicación y a la fragmentación de los datos en varias relaciones. En la medida en que se procesen las consultas para recuperar o actualizar datos, el diccionario de datos del SGBDD puede proporcionar la información requerida sobre la localidad y la duplicación, mientras que asegura que las actualizaciones se propagan a las localidades apropiadas. El diccionario de datos del SGBDD puede mantenerse en un lugar centralizado, o subconjuntos del mismo pueden distribuirse entre varios sitios de acuerdo a las necesidades. Una copia completa del diccionario de datos podría entonces obtenerse haciendo una unión de todos los subconjuntos distribuidos.

Los usuarios interactúan con el SGBDD ejecutando programas que se llaman **transacciones**. Las transacciones en tales sistemas no están ya restringidas a un solo proceso controlado por un módulo de software, sino que pueden invocar un conjunto de procesos cooperantes, funcionando en varios sitios y controlados por módulos independientes de software.

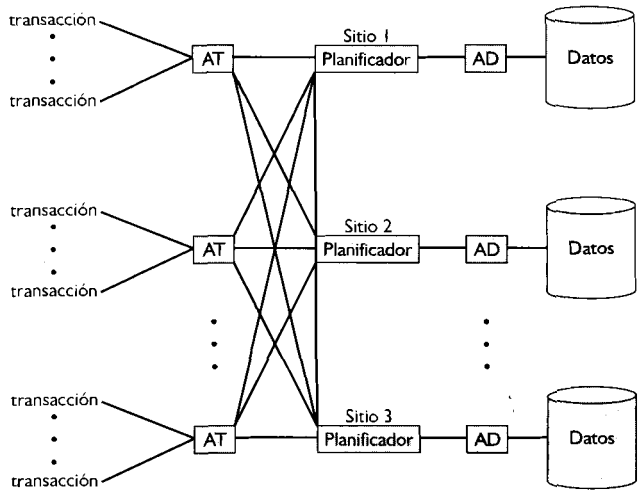
Cada uno de estos procesos cooperantes en la transacción se llama un **agente**. Una transacción que requiere sólo un agente se llama **transacción local**. Una transacción que requiere varios agentes se denomina **transacción global**.

Un agente dado puede sólo acceder a los datos controlados por su software de gestión de datos local. Acceder a los datos de otro sitio requiere de la cooperación de los agentes de esos sitios. Un agente que inicia una transacción se llama agente iniciador. El agente iniciador puede demandar la activación de agentes de otro sitios para poder acceder a los datos necesarios. Una vez que estén activados, dos o más agentes se pueden comunicar mediante intercambio de mensajes.

↖ Las transacciones acceden a los registros mediante operaciones *read* (lectura) y *write* (escritura). Read (x) retorna el valor actual de x. Write (x, nuevo valor) actualiza el valor actual de x al nuevo valor. Una transacción expide las órdenes read y write al SGBDD y ejecuta las entradas y salidas por el terminal.

Cada sitio participante en el SGBDD típicamente ejecuta uno o más de los siguientes módulos de software: un administrador de transacciones (AT), un administrador de datos (AD) o un planificador. La Figura 12.1 ilustra sus interrelaciones. Las transacciones se

[1] Conocido en inglés por las siglas DDBMS *(distributed database management system)* (N. del T.).

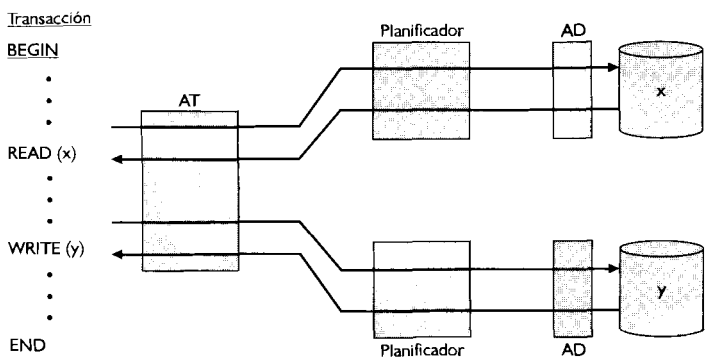


comunican con los ATs; los ATs con los planificadores; los planificadores se comunican entre ellos y también con los ADs y los ADs administran los datos. Estos conceptos se ilustran aún más en la Figura 12.2.

Cada transacción comunica todas sus lecturas y escrituras a un único AT. Una transacción también expide una operación *begin* (empezar) a su AT cuando comienza su ejecución y luego un *end* (terminar) cuando ésta finaliza. El AT comunica cada lectura y escritura al planificador. La selección del planificador se determina por un algoritmo de control concurrente, aunque el planificador que se escoge más a menudo está en el mismo sitio que los datos sobre los que se está operando.

El planificador controla la secuencia en la cual los ADs procesan las órdenes de lectura y escritura y mantiene el control de concurrencia. Cuando un planificador recibe una instrucción de lectura o escritura, el planificador puede procesar la instrucción inmediatamente, demorar el procesamiento guardando la instrucción para su acción posterior, o rechazar la instrucción en el caso de un error de transmisión, violación de acceso o algún problema similar.

El AD ejecuta cada lectura y escritura que recibe. Para una lectura, el AD recorre su base de datos local y retorna el valor solicitado. Para una escritura, el AD modifica la base de datos local y retorna un reconocimiento al planificador, que lo envía de regreso al AT y éste lo regresa a la transacción.



red de área amplia (WAN). Una red de computadoras en la que los sitios están bastante dispersos geográficamente.

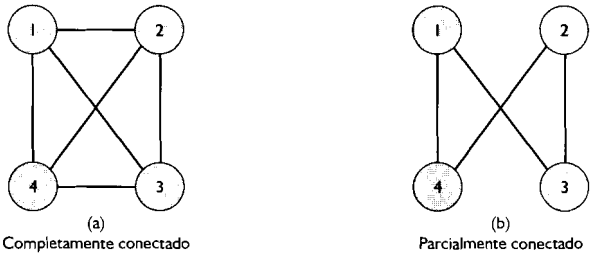
red de área local (LAN). Una red de computadoras en la que los sitios están ubicados a corta distancia (por lo general menos de una milla) uno de otro.

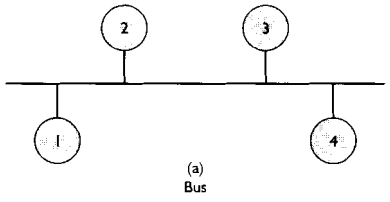
Los SBDDs a menudo enlazan sitios bastante dispersos geográficamente. Estos sistemas se denominan **redes amplias de datos** (*wide area network*) (**WAN**)[2]. Alternativamente, los sistemas de base de datos distribuidas pueden organizarse para servir a **redes de área local** (*local area network*) (**LAN**)[3], donde las estaciones de trabajo de las oficinas son los sitios y hay enlaces entre las estaciones de trabajo. En ambas variantes los sitios pueden enlazarse de varias maneras.

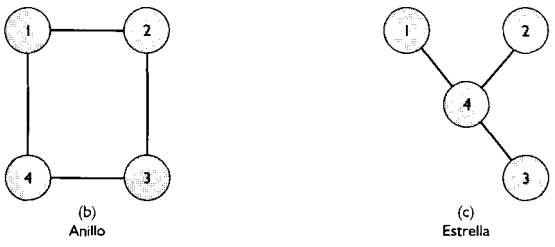
Los enlaces entre los lugares se corresponden con la estructura de un grafo: Los sitios se corresponden con los vértices de un grafo y los enlaces entre los sitios se corresponden con las aristas de un grafo. Los enlaces usuales de un WAN son las líneas telefónicas, canales de satélites o enlaces por microondas. Los enlaces típicos de los LAN son de alta velocidad y están sujetos a bajos índices de error. La cercanía (menos de una milla) de las estaciones de trabajo de un LAN permiten usar enlaces por cables coaxiales o de fibra óptica además de cables de par trenzado.

En la Figura 12.3 se ilustran dos configuraciones genéricas de WANs. La configuración completamente conectada requiere de $n(n\text{-}1)/2$ enlaces. Como este término es del orden de n^2, el número de enlaces crece rápidamente según aumente el número de sitios y, por lo tanto, aumenta el costo de instalación y mantenimiento. Aunque una configuración totalmente conectada nos brinda una gran flexibilidad y fiabilidad, la mayoría de las instalaciones usan redes parcialmente conectadas por su mejor costo-efectividad del diseño de implementación.

Los diseños LAN usualmente siguen arquitecturas como las que se muestran en la Figura 12.4.







[2] Por ser siglas conocidas se mantienen en inglés (N. del T.).
[3] Por ser siglas conocidas se mantienen en inglés (N. del T.).

Cada localización en un sistema de base de datos distribuida debería ser capaz de procesar transacciones que accedan a datos propios, así como también transacciones que requieran datos de varias localizaciones. Los sitios de una red pueden estar conectados físicamente de varias maneras. En la Figura 12.3, un enlace entre dos sitios significa una conexión directa entre ellos. La selección de una configuración sobre otra es normalmente una función de:

1. El costo de enviar un mensaje de una estación *A* a una estación *B*.
2. La frecuencia con la cual un enlace o una estación falla (fiabilidad).
3. El grado en el que se pueden acceder los datos independientemente de los fallos de algunos enlaces o estaciones (disponibilidad).
4. La frecuencia y el volumen de datos que se debe acceder.
5. El costo de enlazar físicamente las estaciones en el sistema.

▼ Diseño de un sistema de base de datos distribuida

El diseño de un sistema de base de datos distribuida puede resultar una tarea compleja. Se deben hacer consideraciones muy cuidadosas sobre los objetivos y las estrategias que deben servir al diseño y se deben, en paralelo, tomar decisiones sobre cómo hay que distribuir los datos entre los sitios de la red.

Estrategias y objetivos

En esta sección se analizan algunas de las estrategias y objetivos que son comunes a la mayoría de las implementaciones de los sistemas de bases de datos distribuidas.

Transparencia de la ubicación. La transparencia de la ubicación permite a un usuario acceder a los datos sin conocer, o tener que ver con los sitios en los que residen los mismos. Las ubicaciones de los datos están ocultas al usuario.

Transparencia de duplicación. La transparencia de duplicación de los datos significa que si existe más de una copia de los datos, una sola copia se debe escoger cuando se van a recuperar datos y todos las copias se deben actualizar cuando se hagan cambios. Estar escogiendo una copia de los datos para la recuperación y asegurando que siempre se actualizan todas las copias de los datos puede ser una carga para los usuarios. Un SGBDD debe hacerse cargo de tales requisitos y liberar de esto a los usuarios, permitiéndoles concentrarse en sus necesidades de información.

Independencia de la configuración. La independencia de la configuración permite a la organización añadir o reemplazar hardware sin tener que estar cambiando los componentes de software existentes en el SGBDD. La independencia de la configuración significa tener un sistema que se pueda expandir cuando el hardware se sature.

SGBDD no homogéneos. A veces es deseable integrar bases de datos mantenidas por diferentes SGBDD sobre computadores diferentes. A menudo los SGBDD son suministrados por diferentes fabricantes y pueden soportar diversos modelos de datos. Un enfoque para integrar estas bases de datos es proporcionar una única interfaz de usuario que pueda ser usada para acceder a los datos mantenidos por los SGBDD no homogéneos. Los diferentes modelos de datos soportados por los SGBDD no homogéneos se le ocultan al usuario mediante esta única y amplia interfaz.

Duplicación de datos. La duplicación de datos ocurre si el sistema mantiene varias copias idénticas de una relación, *R*, con cada copia almacenada en un sitio diferente. Típicamen-

te la duplicación se introduce para aumentar la disponibilidad del sistema: cuando una copia no esté disponible debido a un fallo de un sitio, sería posible tener acceso a otra copia.

La duplicación también puede mejorar el rendimiento, puesto que las transacciones tienen mayor probabilidad de encontrar una copia localmente. El inconveniente está en el costo extra del almacenamiento adicional y en el mantenimiento de la consistencia mutua entre las copias. La actualización de una copia local impone el sobrecosto de transmitir la actualización a todas las copias.

Formalmente las ventajas de la duplicación están en:

1. Si una de las estaciones que contiene una relación R falla, la relación puede recuperarse de otro sitio y el sistema puede continuar cualquier procesamiento que involucre a R. Por tanto, se mejora la disponibilidad de la base de datos.
2. Si la mayoría de los accesos a R sólo involucran una lectura de la relación, entonces varios sitios pudieran estar procesando consultas, que involucren a R, concurrentemente. Mientras más copias haya de R a lo largo de la red, mayor es la posibilidad de que una consulta pueda ejecutarse sin requerir de transmisión de datos entre las estaciones. Lo que significa ganancia en los costos y en tiempo.

La desventaja principal es que el sistema debe asegurar que son idénticas todas las copias de R. Por tanto, cuando ocurra una actualización de R, tal actualización deberá propagarse a todas las localidades con el correspondiente aumento de coste.

Para las bases de datos con duplicación hay al menos dos implementaciones alternativas. En una de éstas se mantiene una base de datos centralizada de la cual se extraen copias de porciones para uso local. Esta redundancia se ve compensada por la reducción de los costos de comunicaciones debido a que los datos están almacenándose localmente. La fiabilidad también se mejora, puesto que la pérdida de los datos en una localidad puede restaurarse por la copia de los datos que se mantiene en otra localidad.

La segunda alternativa es omitir la base de datos centralizada y duplicar segmentos de la base de datos en los sitios que tienen los usuarios más frecuentes. Con esto se evitan los costos de mantener la base de datos centralizada, pero pueden crecer los costos en comunicaciones.

Particionamiento de la Base de Datos. Podemos lograr una mayor eficacia con una estrategia que implemente una base de datos particionada. Con este enfoque, la base de datos se distribuye de modo que no haya solapamiento o duplicación de los datos manteuidos en las diferentes localidades. Puesto que no hay duplicaciones de los datos, se evitan los costos asociados con el almacenamiento y mantenimiento de datos redundantes. Sin embargo, la disponibilidad de los datos se ve limitada si un mismo segmento de dato se usa en más de una localidad. La fiabilidad también puede verse afectada, puesto que un fallo de un sistema de cálculo significa que los datos almacenados en esa localización no están disponibles para los usuarios en cualquier parte en el sistema.

Puesto que el entorno distribuido del SGBDD permite que una base de datos esté físicamente particionada, la seguridad de los datos se puede mejorar, en especial cuando los segmentos de la partición tienen diferentes necesidades de seguridad.

La forma más elemental de implementar una base de datos particionada es tratar a ésta como una serie de sistemas de bases de datos que operan independientemente con capacidades de acceso remoto. Una alternativa, que es en parte más compleja, es una en la cual los sistemas de bases de datos se integran, de modo que una única consulta del usuario puede requerir acceso a más de uno de los sistemas de bases de datos. Mientras que las complejidades subyacentes quedan transparentes al usuario, realmente pueden estar implicadas bastantes operaciones. Pensemos, por ejemplo, en una operación relacional como una reunión (*join*) que requiere de tablas que están mantenidas en dos localidades diferentes. Este problema se verá con más detalle dentro de poco.

fragmentación de datos. La partición de una relación en un SBDD.

Fragmentación de datos. La fragmentación de datos se aplica a los sistemas de bases de datos relacionales. Esta se refiere a la forma en la cual las relaciones se pueden subdividir y distribuir entre los sitios de la red. Esta es una extensión de la estrategia de particionar los datos, la cual por lo general tiene que ver con la ubicación de relaciones (o archivos) completas de la base de datos en varios sitios.

Si una relación R se fragmenta, ésta se divide en subconjuntos (fragmentos) R_1, R_2,.....R_n. La unión de esos fragmentos reconstruye la relación original R. Esta fragmentación puede ser *horizontal*, la cual asigna tuplas a varios fragmentos o puede ser *vertical* en la que los atributos seleccionados se proyectan en varios fragmentos.

¿Por qué considerar la fragmentación? Según señala Maier (1983), una base de datos de las reservas de una aerolínea puede usarse con más efectividad si los subconjuntos de la base de datos se almacenan en la localidad en la que se originan los vuelos respectivos. Similarmente, para una base de datos de un banco, puede ser útil almacenar los subconjuntos de la base de datos en las localidades en cuyas sucursales están localizadas las respectivas cuentas.

Si se permite la fragmentación, una relación dada no necesariamente se almacena completamente en un único sitio. Sus subconjuntos se distribuyen entre varios sitios por consideraciones de rendimiento. Es más, estos subconjuntos pueden duplicarse.

fragmentación horizontal. Partición de una relación en subconjuntos de sus tuplas.

Fragmentación horizontal es la división de las tuplas de una relación en fragmentos. Usualmente estos fragmentos son disjuntos y pueden estar duplicados. De esta manera, la duplicación se gestiona al nivel de la fragmentación y no a nivel de tuplas individuales.

Para ilustrar la fragmentación horizontal suponga que la relación R es la relación VUELO de la Figura 12.5. Esta relación puede dividirse en dos fragmentos, cada uno conteniendo las tuplas que identifican un origen común del vuelo (SF y Den) [4]. Tal fragmentación se define formalmente como una SELECCIÓN sobre la relación R; esto es,

```
R1 = SELECT (R : <condición sobre uno o más atributos>)
```

Para nuestro ejemplo podría ser

```
VUELO-DESDE-DEN = SELECT (VUELO: ORIGEN = 'Den')
VUELO-DESDE-SF = SELECT (VUELO: = 'SF')
```

En la Figura 12.6 se muestran los fragmentos resultantes. El fragmento VUELO-DESDE-SF se almacena en el aeropuerto de San Francisco y el fragmento VUELO-DESDE-DEN se almacena en el aeropuerto de Denver. La relación original se puede restaurar llevando a cabo una operación de UNIÓN de VUELO-ORIGEN-SF y VUELO-ORIGEN-DEN.

Los fragmentos anteriores son horizontalmente disjuntos. Esto es, no ocurren instancias de tuplas en más de un fragmento. Sin embargo, podríamos tener que una tupla particular de R aparece en más de una *Ir*. Por ejemplo, en San Francisco podemos querer almacenar las tuplas que se originan en San Francisco, así como aquellas que llegan de LA [5].

VUELO

Origen	*Destino*	*Hora de partida*	*Hora de llegada*
Den	SF	9am	10am
Den	SLC	7am	8am
Den	SF	1pm	2pm
Den	SF	6pm	7pm
SF	Den	8am	11am
SF	Den	12pm	3pm
SF	Den	5pm	8pm
SF	SLC	9am	11pm

[4] Se respetan las abreviaturas en inglés del original, posiblemente de las ciudades San Francisco y Denver (N. del T.).

[5] Siglas por las que se conoce en Estados Unidos a la ciudad de Los Angeles (N. del T.).

VUELO_ORIGENSF

Origen	*Destino*	*Hora de partida*	*Hora de llegada*
SF	Den	8am	11am
SF	Den	12pm	3pm
SF	Den	5pm	8pm
SF	SLC	9am	11pm

VUELO_ORIGEND

Origen	*Destino*	*Hora de Partida*	*Hora de Llegada*
Den	SF	9am	10am
Den	SLC	7am	8am
Den	SF	1pm	2pm
Den	SF	6pm	7pm

fragmentación vertical. Particionar una relación por la proyección del subconjunto de sus atributos.

Fragmentación vertical es la división del conjunto de atributos de un objeto en subconjuntos, que pueden solaparse. Los fragmentos se obtienen proyectando la relación original sobre cada conjunto de atributos. Para asegurar que las proyecciones son una descomposición sin pérdida de la relación, cada fragmento vertical contendrá normalmente una clave para la relación.

La fragmentación vertical se define como

R_i = R [<lista de nombres de atributos>]

R puede reconstruirse de fragmentos mediante una reunión natural (*natural join*):

R = JOIN (R_1, R_2, . . ., R_n)

La fragmentación vertical requiere de la adición de un atributo especial para identificar la tupla. El atributo de identificación de la tupla (AIT) es una dirección de una tupla. Puesto que las direcciones son únicas, el AIT funciona como una clave para el esquema ampliado para R. En la Figura 12.7 se muestra esta adición de un atributo AIT. En la Figura 12.8 mostramos la relación VUELO descompuesta verticalmente en los esquemas VUELO1 y VUELO2. Estas relaciones son el resultado de calcular:

```
VUELO1 = VUELO [ORIGEN, HORA-PARTIDA, AIT]
VUELO2 = VUELO [DESTINO, HORA-LLEGADA, AIT]
```

Para reconstruir la relación VUELO original a partir de los fragmentos, se calcula:

```
VUELO = JOIN (VUELO1, VUELO2)
```

VUELO

Origen	*Destino*	*Hora de partida*	*Hora de llegada*	*AIT*
Den	SF	9am	10am	1
Den	SLC	7am	8am	2
Den	SF	1pm	2pm	3
Den	SF	6pm	7pm	4
SF	Den	8am	11am	5
SF	Den	12pm	3pm	6
SF	Den	5pm	8pm	7
SF	SLC	9am	11pm	8

VUELO1

Origen	*Hora de partida*	*AIT*
Den	9am	1
Den	7am	2
Den	1pm	3
Den	6pm	4
SF	8am	5
SF	12pm	6
SF	5pm	7
SF	9am	8

VUELO2

Origen	*Hora de partida*	*AIT*
SF	10am	1
SLC	8am	2
SF	2pm	3
SF	7pm	4
Den	11am	5
Den	3pm	6
Den	8pm	7
SLC	11:30pm	8

▼ Distribución de archivos no fragmentados[6]

En la sección anterior hemos analizado las formas de particionar las relaciones entre sitios de la red. En esta sección se considerará el caso en el que relaciones completas se ubican en los sitios. Este es siempre una cuestión central en el diseño e implementación de los sistemas de bases de datos distribuidas. La razón principal es que la mayoría de los archivos son no fragmentados porque hay muchos accesos importantes que lo que necesitan es recuperar todos los registros en los archivos.

El método que destacaremos para determinar la asignación de los archivos ha sido adaptado de Bell y Grimson (1991). La función objetiva de este método es maximizar el acceso local, sujeto a restricciones de almacenamiento. Esto es realista, ya que el costo de añadir una relación a un sitio no incurre en un costo marginal significativo en tanto no exceda la capacidad de almacenamiento. Por el contrario, el costo de acomodar los requisitos adicionales de almacenamiento se considera excesivo si es a corto plazo. Esto es, aumentar las capacidades de almacenamiento puede ser necesario para un negocio creciente con grandes horizontes. Esto no hace daño a nuestras motivaciones y análisis.

Comenzamos definiendo la notación a utilizar en la forma siguiente:

N = el número de sitios indexados por j.
c_j = la capacidad de disco del sitio j (en Mbytes)
A = el número de archivos, indexados por i
s_i = los requisitos de almacenamiento del archivo i (en Mbytes)
T = el número de transacciones, indexado por k
f_{jk} = frecuencia de una transacción k emanando de un sitio j
n_{ki} = número de accesos (para actualizaciones y recuperaciones) requeridos por la transacción k sobre el archivo i
d_{ij} = una variable de decisión a la que se le asigna 1 si el archivo i se asigna a la estación j; 0 en caso contrario.

[6] Esta sección puede omitirse sin pérdida de continuidad.

De aquí se desprende que $\sum_j d_{ij} = 1$ para todos los i tales que $1 <= i <= F$. Las restricciones sobre el espacio de almacenamiento se pueden expresar por $\sum_i d_{ij}$. $s_i <= c_j$ para todo j tal que $1 <= j <= N$. Es decir, los requisitos de almacenamiento de todos los archivos asignados a una estación determinada no pueden exceder las capacidades de almacenamiento de la estación. Es más, j debe ser una de las N estaciones en la red.

La función objetiva es maximizar $(\sum_{ij} D_{ik} V_{ij})$ donde $V_{ij} = \sum_k f_{ki}(n_{ki})$. Note que V_{ik} es una medida del total de accesos asociados con un archivo dado y una transacción. La función objetiva intenta poner los archivos en aquellas localidades que tienen los valores V_{ij} más altos. Puesto que el acceso remoto es más costoso que el acceso local, se desea maximizar el acceso local. Esto se aclara mejor con un ejemplo.

Supongamos que se desean asignar ocho archivos entre cinco sitios, cada sitio con 20-Mbytes de almacenamiento en disco. La Figura 12.9 muestra la razón de acceso de las transacciones a los archivos (n_{ki}). Por ejemplo, cada vez que se inicia la transacción 1 se requieren 10 accesos al archivo 1, 10 accesos al archivo 2, 10 accesos al archivo 5 y 22 accesos al archivo 8.

La Figura 12.10 muestra la frecuencia con la cual se generan las diferentes transacciones desde los cinco sitios de la red, por períodos de tiempo uniformes. Por ejemplo, considere el sitio 1. Este genera 12 transacciones de tipo 2, 20 de tipo 6, 2 de tipo 7, 13 de tipo 8 y 4 de tipo 9.

La Figura 12.11 muestra el resultado de multiplicar el número de filas accedidas por transacción por el número de transacciones generadas en cada sitio V_{ij}.

Seguimos entonces por pasos el procedimiento siguiente:

Paso 1: Calcular $J(i) = \{ j' \mid V_{ij} = \max V_{ij} \}$. Dicho con palabras, para cada archivo identificar el sitio que requiere el mayor número de accesos. Estas son las cifras en

	Archivo (tamaño)							
Transacciones	1 (5)*	2 (10)	3 (15)	4 (10)	5 (10)	6 (7)	7 (4)	8 (5)
1	10	10			10			22
2					21			8
3			70	70	145	14		
4			5	5	11	9		
5			4	4			10	10
6	3	9	5	1				12
7	9	2	2	4				
8	6	5	2	4				
9	1	2						
10								

	Sitio				
Transacciones	1	2	3	4	5
1		25			
2	12	30			
3		3	5	4	5
4			10	10	10
5			12	8	4
6	20	150	100	2	2
7	2	100		30	40
8	18	32	10	12	10
9	4	2			
10		4			

	Archivo (tamaño)							
Sitios	1 (5)	2 (10)	3 (15)	4 (10)	5 (10)	6 (7)	7 (4)	8 (5)
1	190	282	140	100	234	0	0	240
2	**1800**	**1964**	924	**888**	**1315**	42	0	**1800**
3	572	950	**972**	596	835	**160**	**120**	1320
4	348	138	456	532	690	146	80	64
5	426	148	526	618	835	160	50	64

negrita de la Figura 12.11. De esta manera, se identifica el sitio que tiene el máximo número de accesos para cada archivo.

Paso 2: Asignar $d_{ij} = 1$ para todas las entradas en negrita y $d_{ij} = 0$ en los demás casos.

Paso 3: Comprobar para ver que no se haya excedido la capacidad de almacenamiento del sitio. Al sitio 2 se le han asignado archivos que requieren 40 Mbytes de memoria, pero sólo hay disponibles 20 Mbytes.

Paso 4: El máximo V_{ij} que se puede alcanzar al almacenar cualquier archivo del sitio 2 se obtiene almacenando los archivos 1, 2 y 8.

Paso 5: La nueva tabla V_{ij} (Figura 12.12) se obtiene eliminando la fila 2 y las columnas 1, 2 y 8 de la Figura 12.11 y recalculando $J(i)$. Estas son las entradas en negrita de la Figura 12.12.

Paso 2': Asignar $d_{ij} = 1$ a las entradas en negrita; $d_{ij} = 0$ en los demás casos.

Paso 3': El sitio 3 tiene ubicados los archivos 3, 5, 6 y 7 que requieren una capacidad total de almacenamiento de 36 Mbytes. El sitio está sobrecargado.

Paso 4': El máximo V_{ij} que se obtiene de almacenar cualquier archivo en el sitio 3 se alcanza almacenando los archivos 3 y 7.

Paso 5': La nueva tabla V_{ij} (Figura 12.13) se obtiene eliminando la fila 2 y las columnas 1 y 5 de la Figura 12.12. El nuevo $J(i)$ se indica por los valores en negrita de la Figura 12.13.

Paso 2'': Asignar 1 a d_{ij} para las entradas en negrita y 0 al resto.

Paso 3'': Al sitio 5 se le asignan archivos que requieren 27 Mbytes. El sitio está sobrecargado.

Paso 4'': El máximo V_{ij} que se obtiene en el sitio 5 es asignándole los archivos 4 y 5.

	Archivo (tamaño)				
Sitios	3 (15)	4 (10)	5 (10)	6 (7)	7 (4)
1	140	100	234	0	0
3	**972**	596	**835**	**160**	**120**
4	456	532	690	146	80
5	526	**618**	835	160	50

	Archivo (tamaño)		
Sitios	4 (10)	5 (10)	6 (7)
1	100	234	0
4	532	690	146
5	**618**	**835**	**160**

	Total de Almacenamiento	
Sitio	*Archivo*	*MBytes usados*
1	—	0
2	1, 2, 8	20
3	3, 7	19
4	6	17
5	4, 5	20

Es fácil observar que el borrar la fila 3 y las columnas 1 y 2 de la Figura 12.12 nos deja que el archivo 6 sea asignado al sitio 4. La asignación final se resume en la Figura 12.14.

▼ Procesamiento distribuido de las consultas

Algunos sistemas de bases de datos soportan bases de datos relacionales cuyas partes están físicamente separadas. Diferentes relaciones pueden residir en sitios diferentes, múltiples copias de una sola relación pueden distribuirse entre los diferentes sitios, o una relación puede estar particionada en subrelaciones y estas subrelaciones estar distribuidas. Con vistas a evaluar una consulta que se plantee en un sitio dado, pudiera ser necesario transferir datos entre varios sitios. La consideración clave aquí es que el tiempo requerido para procesar tal consulta está grandemente comprometido por el tiempo que se gasta en la transmisión de los datos entre los sitios, más que por el tiempo que se emplea en recuperar datos del almacenamiento secundario o del que se tarda en los cálculos.

Semirreuniones (*semijoins*)

Supongamos que las relaciones *R* y *S* que se muestran en la Figura 12.15 se almacenan en los sitios 1 y 2, respectivamente. Si deseáramos responder a una consulta del sitio 1 que requiera el cálculo de:

```
JOIN (R, S)
```

podríamos transmitir todo *S* del sitio 2 al sitio 1 y calcular la reunión (*join*) en el sitio 1. Esto implicaría la transmisión de todos los 24 valores de *S*.

Sitio 1 *R*		*Sitio 2* *S*		
A1	*A2*	*A1*	*A2*	*A3*
1	3	3	13	16
1	4	3	14	16
1	6	17	13	17
2	3	10	14	16
2	6	10	15	17
3	7	11	15	16
3	8	11	15	16
3	9	12	15	16

Otra forma sería calcular

```
T = R [A2]
```

en el sitio 1 y luego enviar T (6 valores) al sitio 2 para calcular

```
U = JOIN (T, S);
```

y, finalmente, enviar U (9 valores) al sitio 1. Podemos entonces computar el deseado

```
JOIN (R, S)
```

como

```
JOIN (R, U)
```

En la Figura 12.16 se muestran estos pasos y sus resultados. Observe que con este enfoque transmitiríamos sólo 15 valores para completar la consulta.

Este ejemplo sienta las bases para definir una semirreunión. La semirreunión de R con S es

```
SEMIJOIN (R, S) = < una proyección de aquellos atributos de R que se
intersectan con los de S>,
```

lo cual no es más que la parte de R que se reúne (*join*) con S. Por lo tanto,

```
JOIN (R, S) = JOIN (R, SEMIJOIN (R, S), S) )
```

Si R y S están en sitios diferentes, calcular la reunión de R y S permite ahorrar en la transmisión de datos siempre y cuando R y S no se reúnan completamente, como queda demostrado en la Figura 12.16.

```
1. T = PROJECT (R: A2) = A2
                         3
                         4
                         6
                         7
                         8
                         9

2. U = JOIN (T, S) = A2  A3   A4
                     3   13   16
                     3   14   16
                     7   13   17

3. JOIN (R, U) = A1  A2  A3   A4
                 1   3   13   16
                 1   3   14   16
                 3   7   13   17
```

Integridad de los datos en los sistemas de bases de datos distribuidas

Un sistema de base de datos distribuida difiere de un sistema de base de datos centralizado en que la base de datos reside en un conjunto S de sitios. Como puede esperarse, el control de la integridad de los datos se convierte en un problema complicado en el entorno de la red. Las transacciones dejan de ser secuencias lineales y ordenadas de las acciones sobre los datos. Esto quiere decir que, puesto que los datos están distribuidos, las actividades de la transacción pueden tener lugar en diferentes sitios y puede ser difícil de mantener un orden de tiempo entre las acciones.

El problema más común es cuando dos (o más) transacciones se ejecutan al mismo tiempo y ambas requieren acceso al mismo registro de datos con vistas a completar su procesamiento. Este problema de concurrencia para las sistemas de bases de datos centralizadas se examina en el Capítulo 17. El problema, en cierto modo, se exacerba en un sistema distribuido, puesto que puede haber muchas copias de un mismo registro. Todas las copias deben tener los mismos valores todo el tiempo o de lo contrario las transacciones operarían sobre datos imprecisos.

La mayoría de los algoritmos para el control de concurrencia de los sistemas de bases de datos distribuidas usan alguna forma de comprobación, para ver si el resultado de la transacción es el mismo que si las acciones se ejecutasen secuencialmente. Para implementar el control de concurrencia se debe conocer lo siguiente:

1. El tipo de algoritmo de planificación utilizado.
2. La localización del planificador.
3. Cómo se controlan los datos duplicados.

Estos factores proporcionan las bases para la construcción de reglas que determinen cuándo un algoritmo de control de concurrencia es correcto. Las reglas se basan en la **teoría de la serialización** (*theory of serializability*). Cuando las transacciones se llevan a cabo secuencialmente, primero se realizan todas las acciones de una transacción y luego se ejecutan todas las acciones de la transacción siguiente. No hay concurrencia por lo que esto se llama **ejecución serial**. Cuando las transacciones se ejecutan secuencialmente, no pueden interferir entre sí porque sólo hay una activa cada vez. Se vio en un ejemplo inicial (Capítulo 11) con demandas concurrentes a la base de datos de la Liga de las Mujeres por el Voto y de la Cámara de Comercio que las transacciones usualmente no se ejecutan en secuencia, sino que se intercalan. Una ejecución intercalada de las transacciones se dice serializable si produce el mismo resultado que el que produciría una ejecución serializada de las mismas transacciones.

teoría de la serialización. Establece que un algoritmo de control de concurrencia es correcto cuando sus resultados son los mismos que si se hubiese procesado serialmente.

ejecución serial. Acciones que se ejecutan una después de otra —no hay acciones en paralelo.

Para forzar la secuencialización, los ATs ubicados en cada sitio deben cooperar para proporcionar el control de concurrencia, usando mecanismos de cierre o bloqueo (*locking*) o poniendo marcos de tiempo (*timestamping*). Como antes, se deben proporcionar los medios para prever las anomalías que pueden surgir en contra de la base de datos cuando las transacciones se ejecutan en paralelo.

Extenderemos nuestra consideración de estos aspectos examinando algunos de los métodos principales para mantener la integridad en los sistemas de bases de datos distribuidas.

Protocolo de cierre de dos-fases

Desde temprano hemos aprendido que el mantenimiento de la integridad de los datos requiere del procesamiento atomizado de las transacciones. En los sistemas centralizados del Capítulo 11 esto se lograba retardando los cambios a la base de datos hasta que se completase una transacción. En un sistema de base de datos distribuida hay más cosas involucradas. Antes de cerrar las actualizaciones de una transacción, cada subtransacción (aquella parte de la transacción que se ejecuta en un sitio dado) debe mostrar que está preparada para el cierre. De lo contrario, la transacción y todos sus cambios se abortan completamente. La existencia de subtransacciones en varios sitios necesita de esta regla.

Para que una transacción esté lista realmente para el cierre, todas sus acciones deben haber culminado completa y satisfactoriamente. Si alguna subtransacción indica que sus acciones no pudieron terminarse, entonces todas las subtransacciones se abortan y ninguno de los cambios se hacen efectivos. Esta idea se ilustra en la Figura 12.17.

protocolo de cierre de dos-fases. Un protocolo que consiste en una fase de preparado-para-el-cierre y una fase de decisión (cerrar o abortar).

Formalmente, el **protocolo de cierre de dos-fases** [7] se ejecuta de la forma siguiente:

Fase 1. Considere una transacción T_i que se inicia en un sitio e invoca transacciones en otros lugares, así como en el sitio local (en el que se inició T). T consiste completamente de subtransacciones, cada una ejecutándose en un sitio diferente. Las subtransacciones del sitio local se denota como la C_i coordinadora. Las otras subtransacciones se denominan participantes. Cada subtransacción T_i de T decide si cerrar o abortar. C_i envía un mensaje de *preparado-para-el-cierre* a todos los sitios en los que está ejecutándose un T_i. T_i le responde a C_i con un mensaje de *voto-de-cierre* o *voto-de-aborto*.

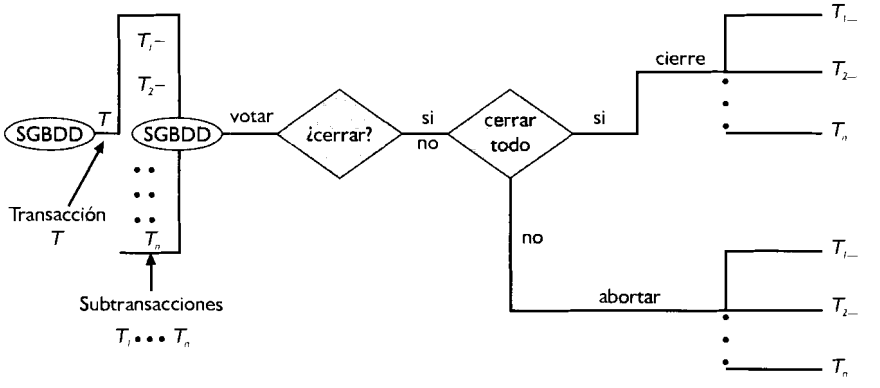
Fase 2. Se basa en la información que recibe de la fase 1, C_i determina si T puede cerrarse o no. De acuerdo a esto, C_i envía mensajes de *cerrar* T o *abortar* T a todos los sitios T_i. Puesto que se requiere consenso en la votación de cierre para poder cerrar una transacción T, T no podrá cerrar si hay algún sitio que votó por abortar.

La mayor limitación del protocolo de cierre de dos fases es que un fallo del coordinador puede resultar en que la transacción quede bloqueada hasta que éste restaure. Se ha propuesto un protocolo de tres-fases que evita esta limitación, pero añade gastos y costes. En la práctica, la probabilidad de que ocurra este bloqueo es lo suficientemente baja como para que no se justifique el costo del protocolo de tres-fases.

Bloqueo distribuido

Las transacciones globales pueden involucrar varias transacciones locales que se ejecutan en sitios diferentes. El SGBDD debe asegurar que estas transacciones se ejecuten en la secuencia apropiada.

Ejemplo 1. Suponga que la transacción T_1 resta 25 pavos del inventario y que la transacción T_2 resta 35 pavos del inventario. Es más, suponga que T_1 se inició en el sitio 1, T_2 se inició en el sitio 2 y que las copias del Registro de Inventario de los Pavos (RIP) se mantiene en los sitios 3 y 4. Las transacciones globales T_1 y T_2 consisten de transacciones locales en los sitios 1, 2, 3 y 4. T_1 debe iniciar dos subtransacciones que deducen 25 del RIP de los sitios 3 y 4 al igual que debe hacer T_2. El efecto de estas transacciones debe estar coor-



[7] *Two phase commit protocol.* No hay un término aceptado en español para *commit*, pudiera decirse también de *terminación*, de *clausura* (N. del T.).

dinado, de modo que el cambio en una copia del RIP no se haga si no se garantizan los cambios en las otras copias del RIP.

El SGBDD de cada localidad mantiene un administrador de bloqueo (*lock manager*), que administra las demandas de bloqueo y desbloqueo de los elementos de datos almacenados en cada sitio. Los bloqueos se pueden aplicar de dos modos: compartido y exclusivo. Si una transacción bloquea un registro en modo compartido, puede leer este registro, pero no actualizarlo. Si una transacción bloquea un registro en modo exclusivo, puede lo mismo leer que actualizar el registro y ninguna otra transacción puede acceder al registro mientras esté bloqueado de este modo. Dos transacciones no pueden tener a la vez bloqueo exclusivo sobre un mismo registro. Sin embargo, cualquier número de transacciones pueden tener a la vez bloqueos compartidos de un mismo registro.

Si hay una única copia del registro entonces el registro lógico es idéntico a su única copia física. Los bloqueos adecuados se mantienen enviando mensajes de demandas de bloqueo al sitio en el que reside la copia. El AT de dicho sitio puede otorgar o denegar el bloqueo, retornando dicho resultado al usuario.

Sin embargo, cuando hay varias copias de un registro, la traducción de bloqueo físico a bloqueo lógico se puede ejecutar de diferentes maneras.

Bloqueo distribuido de dos-fases

Recordemos que el bloqueo de dos-fases (B2F)[8] sincroniza las lecturas y escrituras detectando y previniendo explícitamente los conflictos entre las operaciones coucurrentes. Antes de leer el elemento de dato x, una transacción debe tener un *bloqueo-lectura* (*readlock*) sobre x. Antes de escribir en x, debe tener un *bloqueo-escritura* (*writelock*) sobre x. La propiedad del bloqueo está generalmente gobernada por dos reglas:

1. Transacciones diferentes no pueden poseer simultáneamente bloqueos que entren en conflicto.
2. Una vez que una transacción renuncia a la propiedad de un bloqueo nunca puede obtener bloqueos adicionales.

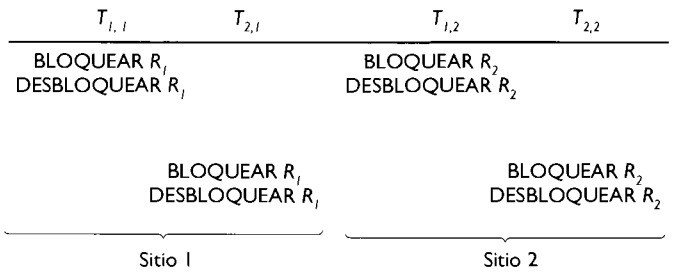
La base de este método es que un paso siempre puede proseguir a menos que entre en conflicto con un paso previo de otra transacción activa que no sea ella misma. La verificación de esto en un sistema de bases de datos distribuida es exactamente la misma; la cuestión es cuál es la mejor manera de llevar esto a cabo. Una forma es que el SGBDD compruebe cuándo un registro accedido por el paso en cuestión, ha sido accedido por una transacción activa. Usando este enfoque, se le solicita al AT que obtenga los bloqueos respectivos antes de leer o escribir datos. Esto es, el AT debe haber recibido un bloqueo-lectura del SGBDD local desde el cual se lee el dato. Similarmente, antes de actualizar un registro, un AT debe haberse provisto de un bloqueo-escritura de todas las bases de datos que almacenan el registro en cuestión. Ilustremos estas ideas con un ejemplo.

Ejemplo 2. Suponga que una transacción T está compuesta de dos acciones de la forma siguiente:

1. $T_{1.1}$, que opera en el sitio 1 y escribe un nuevo valor para la copia R_1 del registro R, y
2. $T_{1.2}$, que opera en el sitio 2 y escribe el mismo valor que en (1) para la copia R_2 de R.

Además considere una transacción T_2 que tiene dos subtransacciones $T_{2.1}$ funcionando en el sitio 1 y escribiendo un nuevo valor en R_1 y $T_{2.2}$ operando en el sitio 2 y escribiendo el mismo valor en R_2. El bloqueo de dos-fases se ilustra en la Figura 12.18. Observe que podrían ocurrir simultáneamente pares de eventos en cada línea. Los eventos en el sitio 1 sugieren que $T_{1.1}$ debe preceder a $T_{2.1}$. En el sitio 2, $T_{1.2}$ debe preceder a $T_{2.2}$.

[8] Conocido en inglés por las siglas 2PL (*two-phase locking*) (N. del T.).



Una técnica de bloqueo de dos-fases primero pasa por una fase de adquisición de todos los bloqueos de la transacción y luego por una fase de liberar estos bloqueos: No se puede adquirir ningún nuevo bloqueo después que se haya liberado algún bloqueo. Papadimitriou (1988) probó que los bloqueos-lectura y los bloqueo-escritura generarán cronogramas consistentes si y sólo si las transacciones procesan en dos fases. Esto requiere que el AT evalúe y reordene, si es necesario, la secuencia de pasos que arriban a cada sitio. El AT debe ser un programa distribuido con un subprograma llamado módulo, ejecutando en cada sitio. El cómputo de un módulo es una serie de acciones tales como:

1. Recepción de una acción y otorgar su ejecución.
2. Enviar un mensaje a otro sitio.
3. Recibir un mensaje.

Para mostrar que esta estrategia siempre genera cronogramas consistentes, es necesario probar que el orden de los bloqueos permanece igual al de un cronograma serial.

Marcas de tiempo

marca de tiempo. Un método para identificar los mensajes con la hora de transmisión.

Con un protocolo de **marca de tiempo** (*timestamp*), a cada transacción T_i se le asocia una marca de tiempo $ST(T_i)$ cuando entra al sistema. Si T_i entra al sistema antes que T_j, entonces $ST(T_i) < ST(T_j)$. Las marcas de tiempo se conforman a partir de una secuencia creciente de enteros. El protocolo gestiona la ejecución concurrente, de modo que sea equivalente a una predeterminada ejecución seriel. Dicha ejecución se define por el orden creciente de las marcas de tiempo.

¿Cómo fuerzan las marcas de tiempo la serialización del procesamiento de las transacciones? Con cada registro (o elemento de dato) X en la base de datos, se identifican dos momentos: el tiempo de lectura ($STL(X)$), que es la marca de tiempo más alta poseída por alguna transacción que tenga que leer el registro; y el tiempo-escritura ($STE(X)$), que es la marca de tiempo más alta poseído por alguna transacción que tenga que escribir en el registro. La serialización se garantiza de las siguientes formas:

1. Una transacción T_i, que tenga una marca de tiempo $ST(T_i)$, no puede leer o escribir en X si $ST(T_i) < STE(X)$.
2. Una transacción T_i, con marca de tiempo $ST(T_i)$, no puede escribir en un registro X con tiempo de lectura $STL(X)$ si $ST(T_i) < STL(X)$.

Para mayor claridad, para cualquier transacción T_i, si $ST(T_i) < STE(X)$, entonces cualquier operación de lectura realizada por T_i, si se permitiese, resultaría en que T_i leería un valor que ha sido sobrescrito por la última transacción. Todo intento de escritura

por parte de T_i, si se permitiese, resultaría en escribir un valor obsoleto de X. Ambos casos violan la serialización, por lo que deberían rechazarse todas las demandas de ese tipo. En tales instancias, T_i debería abortarse, asignársele una nueva marca de tiempo y reiniciarse.

Similarmente, si $ST(T_i) < STL(X)$, entonces cualquier operación de escritura por parte de T_i, si se permitiese, significa que el valor de T_i está intentando escribir fue necesario previamente. Como antes, T_i debería abortarse, asignársele una nueva marca de tiempo y reiniciarse.

Hay dos métodos principales para asignar marcas de tiempo únicas. Uno es centralizado y el otro es descentralizado. En el método descentralizado se le da a único sitio la responsabilidad de asignar marcas de tiempo a las transacciones. Con el esquema descentralizado, a cada sitio se le permite generar una única marca de tiempo local. Una marca de tiempo global se obtiene de concatenar los sellos de tiempo locales con (seguido por) un identificador único del sitio. Si la concatenación se invirtiese, el sitio con el mayor identificador generaría consistentemente marcas de tiempo mayores que los demás sitios.

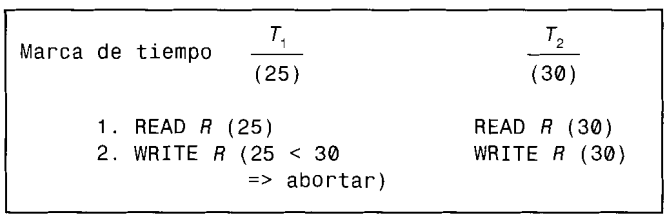
Las marcas de tiempo de las transacciones pueden crearse haciendo que el AT lleve la cuenta del número de transacciones que ha planificado y que le asigne el próximo número a cada nueva transacción. De esta manera, dos transacciones no pueden recibir la misma marca de tiempo y su orden relativo es consistente con el orden en el cual se inician las transacciones. Un segundo enfoque es usar el valor del reloj interno de la máquina en el momento en que se inicia el proceso.

Ilustraremos esto con un ejemplo.

Ejemplo 3. Examine la Figura 12.19. A T_1 se le da una marca de tiempo de 25 y a T_2 uno de 30. (El tiempo inicial de lectura y de escritura de los registros X y Y se asumen que es 0). Cuando T_1 lee X a éste se le da una marca de tiempo de lectura de 25 ($STL(X) = 25$). Cuando T_2 escribe en X se le da un $STE(X) = 30$. Cuando T_2 escribe en Y tenemos que $STE(Y) = 30$. Cuando T_1 intenta leer Y, tenemos que $ST(T^1) = 25 < STE(Y) = 30$. T_1 debe abortarse, de lo contrario estaría leyendo un valor que ha sido escrito después que el valor T_1 se deba leer.

Para implementar este esquema en el entorno de la base de datos distribuida, se necesitan seguir los pasos siguientes:

1. Las transacciones pueden ejecutar en cualquier sitio. Cuando ellas leen o escriben alguna copia del registro, se captura la marca de tiempo en la copia del registro en el sitio.
2. Si una transacción escribe un nuevo valor para la copia del registro en el sitio, el mismo valor se tiene que escribir en todas las copias de dicho registro.



▼ Recuperación de la base de datos

En el Capítulo 11 se destacaron estrategias de recuperación que tienen su fundamento en un diario de transacciones. En un sistema distribuido, se debe mantener un diario en cada sitio. Además de la información especificada previamente, el diario del sitio debe registrar cada mensaje que éste envía y recibe. En un entorno de bases de datos distribuida deben existir medios para detectar los fallos y restablecer el estado del sistema en el momento del fallo cuando el sistema haga vuelta atrás. El diario del sitio indica qué subtransacciones fueron iniciadas pero no se completaron en dicho sitio.

▼ Sistemas cliente/servidor

Los sistemas cliente/servidor (CS) fueron estudiados con algún detalle en el Capítulo 9. Se discutirán brevemente los CS en este capítulo por dos razones: (1) Son un tipo especial de sistema de base de datos distribuida y (2) para brindar una introducción a aquellos que se hayan saltado los capítulos anteriores. El estudio de los CS se hará a tres niveles. Primero, se destacarán los conceptos generales que distinguen al CS, luego se relacionarán brevemente esos conceptos con los sistemas de bases de datos. Luego se analiza un tipo particular de CS: el SQL Server para Windows NT. Todo esto en conjunto debe darle una comprensión general de CS y de su potencial para dar soporte a sistemas de bases de datos, así como algunas características de una implementación en particular.

Conceptos generales

El término *cliente/servidor* (CS) no es nuevo, puesto que los sistemas CS involucran a muchas computadoras conectadas en una red, es un concepto que nos estará dando vueltas por algún tiempo. Un concepto distintivo de los sistemas CS es que una o varias de estas computadoras pueden funcionar como proveedores de servicios a los restantes computadores. El cliente y el servidor han sido definidos formalmente como sigue (Ullman, 1993):

Cliente: Una computadora o una estación de trabajo conectado a una red y que se utiliza para acceder a los recursos de la red.

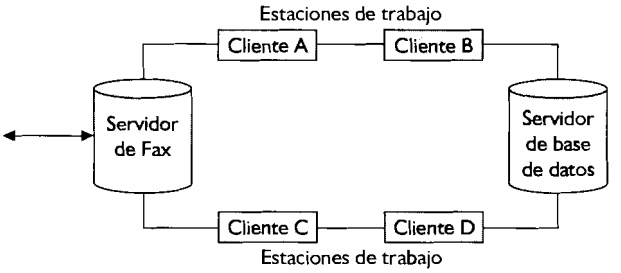
Servidor: Una computadora que suministra a los clientes servicios tales como bases de datos, conexiones a red o grandes controladores de disco. Los servidores pueden ser *grandes computadoras*, minicomputadoras, estaciones de trabajo o dispositivos LAN. Más de un servidor puede estar involucrado en suministrar servicios a los clientes.

La argumentación para adoptar una tecnología CS es doble. Primero, ésta puede reducir los costos a largo plazo. Segundo, la tecnología CS facilita el desarrollo de plataformas orientadas a las aplicaciones específicas. Esta última característica puede mejorar la capacidad de una empresa para responder con rapidez a las condiciones cambiantes de los negocios.

La Figura 12.20 muestra un sistema CS con dos servidores, una base de datos servidor y un servidor de fax. Suponga que un Cliente A requiere datos que están contenidos en el servidor de la base de datos. El usuario expresa la solicitud de los datos en forma de una consulta (por ejemplo, SQL) y transmite ésta al servidor de la base de datos. El servidor de la base de datos ejecuta la consulta sobre sus datos y transmite el resultado al Cliente A. Al mismo tiempo, los Clientes B y C pueden solicitar datos del servidor de bases de datos y el servidor de la base de datos puede dar servicio a varias solicitudes en paralelo.

El servidor de fax puede suministrar números de fax, así como transmitir y recibir los fax para los usuarios.

Los sistemas CS intentan ser sistemas abiertos en el sentido de que permiten a la organización escoger entre varios fabricantes los productos y componentes del CS.



Los clientes y servidores deben ser capaces de comunicarse entre sí. La tecnología actual abarca dos modos de comunicación: mensajes y llamadas a procedimientos remotos. El intercambio de mensajes básicamente significa que el cliente envía una demanda (mensaje) y el servidor responde a tal demanda (mensaje). Una llamada a un procedimiento remoto permite que una aplicación en un computador llame a un procedimiento que puede residir en otro cliente del sistema.

Los clientes se comunican con los servidores a través de un programa de interfaz de la aplicación (API)[9], a veces denominado intermediario. Esto es, las llamadas a los procedimientos remotos y los intercambios de mensajes se manejan por los APIs. APIs efectivos pueden facilitar la implementación de nuevas aplicaciones acomodando diferentes protocolos de red y sistemas operativos. En la medida en que la tecnología de los CS vaya madurando, las características de los APIs probablemente pasarán a formar parte de los sistemas operativos.

Aplicaciones de bases de datos

Actualmente, las aplicaciones más comunes cliente/servidor usan una base de datos relacional. Puede referirse de nuevo a la Figura 12.20. El conjunto de transacciones involucradas son las demandas del usuario y las respuestas del sistema de base de datos. Ésta se facilita por el software, que habilita al cliente para hacer las consultas al servidor de la base de datos. Un ejemplo ampliamente utilizado es el lenguaje SQL.

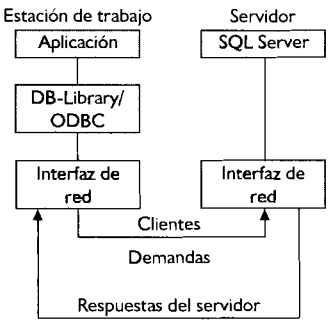
Esto suena bastante simple. Mientras SQL proporciona medios estandarizados de acceder a una base de datos relacional, algunos fabricantes le han añadido modificaciones a sus versiones, que han hecho que dos productos basados en SQL lleguen a ser incompatibles. Suponga que una base de datos cliente/servidor ha sido desarrollada para dar soporte a la gestión de proyectos y que el título y ubicación del proyecto se tienen en una base de datos XYZ y el costo del proyecto se tiene en una base de datos ABC. Ambas bases de datos usan SQL, pero no son completamente compatibles. Cuando los usuarios requieran datos de ambas bases de datos no quieren echarse encima tales incompatibilidades. Las necesidades inmediatas de los usuarios son de información que le ayude en sus tareas. Es aquí en donde el API encaja al brindar la capa de software, transparente al usuario, que permite un acceso uniforme a todas las bases de datos.

SQL server para Windows NT

SQL Server (SQLS) es un servidor de bases de datos que usa el modelo relacional estándar. SQLS usa convenciones para el procesamiento atómico de transacciones que se analizó en el Capítulo 11, junto con el diario de operaciones de escribir para la recuperación.

Para el cliente, el SQLS permite dos opciones de API:

[9] Por ser siglas establecidas, se mantiene en inglés API (*Application Program Interface*) (N. del T.).



1. DB-Library™ que es una API para clientes Windows, DOS y OS/2 de una plataforma cruzada de SQL Server.
2. ODBC [10] que es un API universal de acceso a datos para clientes Windows. Esencialmente, ODBC es API abierto, neutral (no ligado a ningún fabricante), que permite que las aplicaciones basadas en Windows puedan acceder a bases de datos heterogéneas.

La Figura 12.21 ilustra la arquitectura utilizando estos APIs. El DB-Library es un subconjunto de subrutinas escritas en C que suministran a los programadores de aplicación de interfaces estándares entre sus aplicaciones y SQLS. DB-Library libera al programador de los detalles de la red y de los protocolos del servidor.

ODBC es un API de conectividad de base de datos. ODBC permite que las aplicaciones clientes establezcan conexiones directas e intercambien información con las bases de datos SQLS y las pasarelas (*gateways*) del Open Data Services.

En este capítulo se ha presentado una introducción a los sistemas de bases de datos distribuidas. Primero se introdujo cierta terminología y se presentó la arquitectura general del modelo de los SGBDD.

La estructura que subyace en un SGBDD es un sistema de varios SGBD ejecutándose en sitios locales separados, los cuales están conectados por facilidades de envío de mensajes. Cada sitio de un SGBDD ejecuta uno o más módulos de software, incluyendo un administrador de transacciones, un administrador de datos y un planificador.

Veamos los principales factores de motivación en la implementación de los sistemas de bases de datos distribuidas. La fiabilidad, rendimiento de la respuesta, independencia de la ubicación y de la duplicación, independencia de la configuración, SGBDs no homogéneos, seguridad y congruencia organizacional, son todos factores que por sí solos o combinados pueden motivar que se use un sistema de base de datos distribuida.

Las características del diseño de los sistemas de bases de datos distribuidas fueron introducidas centrando la atención en las alternativas siguientes:

1. Base de datos duplicada y una base de datos centralizada.
2. Base de datos duplicada sin una base de datos centralizada.
3. Una base de datos particionada implementada como una serie de sistemas de bases de datos operados independientemente, pero que permiten el acceso remoto desde otros sitios.

[10] Siglas de *Object Data Base Connectivity*; por ser siglas establecidas, se mantienen en inglés (N. del T.).

4. Una base de datos particionada implementada de forma integrada, de modo que una consulta que requiera el acceso a datos que estén en más de un sitio sea tratada por el SGBD y su funcionamiento sea transparente al usuario.

Se sugirieron varios criterios para ayudar a determinar las configuraciones apropiadas.

Se analizó el uso de las relaciones fragmentadas como un medio de sacar ventaja de las capacidades de los sistemas de bases de datos distribuidas. Se definieron la fragmentación vertical y la horizontal y se demostró su utilización. También se demostró el uso de las semirreuniones (*semijoin*) como medio de limitar los costos de comunicaciones. Sin embargo, muchos archivos no pueden estar sujetos a tales fragmentaciones. En tales casos, un aspecto del diseño debe ser dónde ubicar los archivos completos. Se presentó un método práctico de análisis.

Se presentó el control de concurrencia distribuida junto con el uso del protocolo de dos-fases para prevenir los problemas de integridad de los datos. También se trataron las marcas de tiempo, las copias de seguridad y la recuperación.

Los sistemas de bases de datos distribuidas pueden ser complejos, pero ofrecen capacidades que amplían las ventajas de la tecnología de base de datos. Puede esperarse su uso generalizado en organizaciones de muchos tipos.

Por último, se destacaron los conceptos de los sistemas cliente/servidor. En términos de los sistemas de bases de datos, este es un concepto nuevo muy importante porque facilita a las aplicaciones acceder a los datos de uno o más sistemas de bases de datos, brindándole un poder de recuperación de información considerable a los usuarios.

1. Defina con sus propias palabras cada uno de los términos siguientes:
   a. un sistema de base de datos distribuida
   b. datos globales
   c. sistema de gestión de base de datos distribuida
   d. R*
   e. transacción
   f. transacción local
   g. WAN
   h. teoría de la serialización
   i. fragmentación de datos
   j. fragmentación vertical
   k. protocolo de cierre (*commit*) de dos-fases
   l. marca de tiempo (*timestamping*)

2. Describa en qué difieren los sistemas de bases de datos distribuidas y los sistemas de bases de datos centralizadas.

3 Describa los módulos de software de un sistema de gestión de base de datos distribuida y cómo operan al ejecutar una transacción que requiera datos de más de un sitio.

4. Explique varias de las ventajas de los sistemas distribnidos.

5. ¿Bajo qué condiciones puede un sistema distribuido no funcionar tan bien como uno centralizado?

6. Describa seis objetivos/estrategias de la implementación de un sistema de base de datos distribuida.

7. Contraste los conceptos de las bases de datos particionadas y duplicadas. ¿Cuándo se prefiere una u otra?

8. Compare la semirreunión (*semijoin*) con la reunión natural (*natural join*).

9. ¿Cuál es el propósito del protocolo de cierre de dos-fases? ¿Cómo trabaja?

10. ¿Cómo se efectúa la recuperación en un sistema de base de datos distribuida?

1. Haga corresponder cada uno de estos términos con su definición:

—*datos locales*	a. Un proceso que coopera para completar una transacción.
—*enlace*	b. Una transacción que requiere datos de más de una localidad.
—*SSD*	c. Un canal de comunicaciones entre dos sitios en una red de computadores.
—*Distributed INGRESS*	d. Los datos que se mantienen en la base de datos de un sitio en un sistema de base de datos distribuida.
—*agente*	e. Las acciones invocadas por una transacción se ejecutan en fila.
—*transacción global*	f. Una red de computadores que está limitada a un área geográfica pequeña.
—*LAN*	g. Un SGBDD suministrado por Relational Technology.
—*ejecución serial*	h. Un SGBDD suministrado por Computer Corporation of America.
—*fragmentación horizontal*	i. Particionar una relación en el subconjunto de sus tuplas.

2. Constituya una base de datos ejemplo para una empresa que tenga una base de datos distribuida en tres localidades. Escoja un plan de distribución de datos para esta base de datos y justifique su elección.

3. Pruebe cómo se puede aplicar la fragmentación vertical al plan desarrollado en (2).

4. Pruebe cómo se puede aplicar la fragmentación horizontal al plan desarrollado en (2).

5. Considere la relación.

```
EMPLEADO (ID, nombre, dirección, oficio, núm_proyecto)
EQUIPAMIENTO (NUMID, tipo, proyecto)
```

Suponga que la relación EMPLEADO se fragmenta horizontalmente por núm_proyecto y que cada fragmento se almacena localmente como el proyecto del sitio. Suponga que la relación EQUIPAMIENTO se almacena completamente en la localidad de Seattle. Describa una buena estrategia para procesar las consultas siguientes:

a. Encuentre la reunión de empleados y equipamiento.
b. Obtenga todos los empleados para los proyectos que utilicen los Camiones R2.
c. Obtenga todas las máquinas del Proyecto Parowan.
d. Encuentre todos los empleados del proyecto que usan la máquina número 12.

6. Dada la información de las Figuras 12.1E y 12.2E, determine la distribución óptima de los ocho vuelos entre los cinco sitios, asuma una capacidad de disco de 25 Mbytes en cada lugar.

	Archivo (tamaño)							
Transacciones	1 (8)	2 (8)	3 (16)	4 (9)	5 (10)	6 (7)	7 (4)	8 (5)
1	10	9				10		20
2						20		9
3				60	70	140	16	
4				6	5	15	10	
5				4	4			10
6	4	10		6	1			15
7	9	2		3	4			
8	6	5		2	4			
9	2	2						
10								

	Sitio				
Transacciones	1	2	3	4	5
1		25			
2	12	30			
3		3	5	4	5
4			10	10	10
5			12	8	4
6	20	150	100	2	2
7	2	100		30	40
8	18	32	12	12	10
9	4	2			
10		4			

7. Calcule una semirreunión a partir de las relaciones siguientes:

R_1 =	X	Y	Z
	c	b	a
	d	e	f
	c	b	d
	e	a	b
	h	j	k

R_2 =	Z	V	W
	a	d	e
	a	f	h
	b	a	b
	c	d	c
	c	b	a

8. Examine la Figura 12.3E. Dada la historia de transacciones que se presenta aquí, suponga que se ha aplicado un protocolo de marca de tiempo. Indique dónde deben abortarse las transacciones y explique por qué.

Los sellos de tiempo son $T_1 = 1$, $T_2 = 2$ y así sucesivamente.

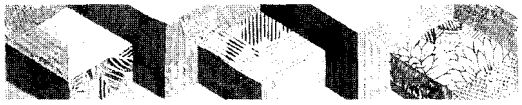
T_1	T_2	T_3	T_4	T_5
				read(y)
	read(x)			
read(x)				
		write(x)		
		write(z)		
				read(z)
	read(z)			
	abortar			
read(y)				
			write(z)	
			abortar	
				write(y)
				write(z)

1. Escriba un ensayo corto sobre el futuro de los sistemas de bases de datos distribuidas.

2. Haga un proyecto de revisión bibliográfica para obtener ejemplos de implementaciones actuales de sistemas de bases de datos distribuidas. ¿Puede encontrar algún material que analice los aspectos de rendimiento?

3. Contacte con una empresa profesional de contabilidad e indague sobre los controles que deberían presentarse en un sistema de procesamiento distribuido. ¿Hay áreas en las que no se usan los controles adecuados?

4. Una de las ventajas de los sistemas CS es que éstos proporcionan los medios para que una empresa pueda cambiar la forma en que opera (cómo hace los negocios).Vea si puede encontrar uno o dos artículos que describan los cambios que se han facilitado por la implementación de un sistema CS.



CAPÍTULO

13

SELECCIÓN E IMPLEMENTACIÓN DE UN SGBD

Análisis de las necesidades de información para la gestión.
- Determinar los requisitos de la aplicación.
- Mantener la consistencia de los datos.
- Requisitos de los tiempos de respuesta.

Funciones y capacidades de los SGBD.
- El diccionario/directorio de datos.
- Seguridad e integridad de los datos.
- Capacidades de consulta, de manipulación de datos y de informes.
- Soporte a los requisitos de programación especializada.
- Opciones de organización física de los datos.

Clasificación de los requisitos de las propiedades de un SGBD.

Reunir datos sobre la existencia de propiedades y el rendimiento.
- Adquirir datos de los vendedores.
- Pruebas de evaluación

Modelos de evaluación.
- Modelo de puntuación.
- Análisis del recubrimiento de datos.
- Fundamentos.
- Aplicación.

Cuestiones de implementación.
- Administración de las bases de datos.
  - Requisitos del servicio al usuario final.
  - Asegurando la integridad y la seguridad de las bases de datos.
  - Copias de seguridad y recuperación.
- Pruebas a la base de datos.
- Preparar a los usuarios para el cambio.
- Cargar la base de datos.
- Mantenimiento de la base de datos.
  - Administrando los recursos.
  - Copias de seguridad y recuperación.
  - Administrando los cambios al sistema de base de datos.
  - Facilidades para supervisar los SGBD.

Resumen.

Preguntas de comprobación.

Problemas y ejercicios.

Proyectos y cuestiones profesionales.

Steve Blue, vicepresidente ejecutivo de la corporación Zeus, estaba siendo entrevistado por Honoria Remington, una estudiante de Sistema de Información de una universidad cercana. Como proyecto de curso, Honoria intentaba desarrollar un estudio sobre cómo Zeus había procedido en la selección e implementación de un SGBD: qué factores se habían considerado importantes, cómo fueron medidos y qué métodos de análisis se usaron.

En respuesta a las preguntas de Honoria, Steve dijo: "Pensamos que la selección de un SGBD sería simple. Que nos haríamos visitar por un par de representantes de ventas, que nos contarían sus productos, que nos darían un precio y entonces tomaríamos una decisión. Obviamente, estábamos siendo ingenuos. Cuando los representantes de ventas empezaron a llenar el ambiente con su jerga y declaraciones promocionales, comprendimos rápidamente que necesitábamos retirarnos y emprender un enfoque más razonable e informado".

"¿Y cómo lograron eso?", preguntó Honoria.

"Bueno, teníamos mucha confianza en nuestro consultor, Linda Kelly. Había hecho un buen trabajo para nosotros anteriormente, así que le preguntamos qué hacer. Esencialmente, delineó cuáles eran las funciones básicas que debían brindarse en un SGBD. Después nos mostró cómo redefinir esas funciones, en términos de sus características subyacentes. Lo hizo para dividir el análisis en capacidades que fueran objetivamente medibles. Entonces sugirió formas de clasificar nuestras necesidades y vías para obtener información. Finalmente, nos enseñó cómo podíamos evaluar las capacidades de cada SGBD y usar un modelo formal para producir una apreciación general de éste."

"Parece ser una gran tarea," dijo Honoria. "¿Linda les cobró bastante?"

Steve sonrió y dijo: "Pues sí, pero valía la pena. Trabajamos bastante para satisfacer su plan, pero cuando obtuvimos los resultados, supimos que habíamos hecho el mejor trabajo posible, y nos sentíamos confiados de nuestra decisión final. También le pedimos que trabajara con nuestro administrador de base de datos para el desarrollo de un plan de implementación."

El interés de este capítulo es el proceso de selección e implementación de un SGBD. Este tema podría él sólo abarcar un libro, pero limitaremos nuestra presentación al delineamiento de sus aspectos principales. Primero consideraremos la relación entre las necesidades de información para la gestión y el sistema de bases de datos. Después trataremos las funciones básicas de los SGBD, analizando algunas de las propiedades principales que son importantes cuando se evalúan los SGBD alternativos. A continuación estudiaremos un método para clasificar estas propiedades, seguido de otro estudio de dos métodos que permiten una comparación y evaluación general. Finalmente describiremos los componentes claves de una implementación exitosa de un SGBD.

Después de leer este capítulo, debe ser capaz de:

- Analizar la relación entre las necesidades de información para la gestión y un sistema de bases de datos.
- Explicar cómo una evaluación de un SGBD puede mejorarse mediante la identificación de las propiedades que le permiten a cada función satisfacer las necesidades estratégicas de información.
- Clasificar las propiedades de acuerdo a su importancia.
- Aplicar métodos útiles que combinen evaluaciones de múltiples propiedades con un modelo general de decisión.
- Analizar y conformar la lista de las tareas que deben realizarse para implementar con éxito un SGBD.

▼ Análisis de las necesidades de información para la gestión

información para la gestión. Información de apoyo a los que toman decisiones y a las operaciones dentro de una compañía.

Usamos el término **información para la gestión** en el sentido general de la información requerida como apoyo a la toma de decisiones y a las operaciones de una empresa en busca de ciertas metas. La información para la gestión requerida por una empresa, que ofrece un producto que cambia lentamente en un mercado estable, puede de alguna manera ser diferente de la información que precisa una empresa que se encuentra en un mercado volátil, o que produce una variedad de productos sujetos a una rápida obsolescencia.

Considere la necesidad de información de una empresa de manufactura que usa la Planificación de Requisitos de Materiales (PRM) para administrar sus procesos de producción. La PRM requiere una base de datos extensa que contenga información sobre la planificación final de los productos, inventarios, las facturas de los materiales, enrutamientos, y los tiempos de coordinación de las fases de manufactura. Si se añade la información sobre los costos y la necesidad de recursos para cada paso de manufactura, los datos del PRM pueden usarse para el registro de costos, control de taller y administración y planificación de la capacidad. Tal base de datos proporciona casi toda la información de planificación y control que se necesita para una planta de manufactura.

Esta base de datos proporciona mucha información de gestión requerida por la empresa. Qué tipo de sistema de base de datos podría satisfacer mejor las necesidades de información. La base de datos que precisa PRM proporciona una vasta información para la gestión, pero es muy difícil de construir y mantener. Cada entorno de manufactura es dinámico: nuevos productos se añaden, los viejos se eliminan de la línea de productos; los métodos de diseño y manufactura cambian regularmente; los tiempos varían; los problemas y ajuste de inventario puedeu ocurrir frecuentemente. Tales requisitos juegan a favor de un SGBD relacional.

Contrástelo con la empresa de manufactura que opera un sistema de inventario a la orden JIT (en tiempo preciso) [1]. El intento de un sistema JIT es que las materias primas y las piezas de ensamblaje se mueven inmediatamente de la entrega, pasando por la manufactura, hacia el centro de consumo. Las piezas se hacen en pequeños lotes, el sistema de producción hace sólo unas pocas piezas malas antes que se descubran los errores, y los cortos tiempos de fase hacen que sea más fácil determinar los problemas en su mismo origen.

Los sistemas JIT funcionan mejor cuando la demanda es alta y los requisitos de producción resultan en una producción casi continua. Un SGBD jerárquico o en red (ver Capítulos 15 y 16) puede resultar apropiado en este caso.

Claro, es inapropiado hacer recomendaciones tan ligeras basándose en estas situaciones tan simplificadas. Lo que nos interesa es que las necesidades de información para la gestión que pueden afectar a la selección de un SGBD pueden incluir lo siguiente:

1. La necesidad potencial de información, que puede requerir datos de más de una aplicación.
2. El número de aplicaciones donde las relaciones entre los datos están bien establecidas y sujetas a pequeños cambios.
3. El volumen actual y esperado de inserciones y eliminaciones que pertenecen a estructuras de datos nuevas y existentes.
4. La forma en la que se clasifican los datos para la toma de decisiones. Por ejemplo, un vendedor de comida preparada puede tomar decisiones de promoción basándose en las ventas de sus productos en todo el país. Otra empresa del mismo tipo puede tomar las mismas decisiones basándose en las ventas en varios

[1] En inglés, conocido por las siglas JIT (just-in-time).

lugares dentro de las grandes ciudades. La primera compañía requiere un conjunto de datos más simple que la última. Si la última quiere experimentar con vistas diferentes de la información para probar varias estrategias, se necesitará un gran grado de flexibilidad.

Habiendo presentado el problema de la necesidad de información a nivel general, procedemos ahora a un examen de los requisitos específicos de información.

Determinar los requisitos de la aplicación

La determinación de los requisitos de información que tienen que incluirse en un sistema de base de datos es un proceso complejo que es esencial en la selección de un SGBD. Sin embargo, el proceso de análisis puede simplificarse si se da cuenta que los usuarios típicamente caen en tres tipos de clases: usuarios regulares, usuarios repetitivos y usuarios ocasionales.

El usuario repetitivo es aquel cuyas aplicaciones pueden ser descritas como sistemas de producción. Los requisitos para estos sistemas orientados a transacciones se planifican de antemano y generalmente incluyen operaciones de rutina de la compañía, tales como registro de ventas, mantener el inventario, etc. Idealmente, el SGBD, debe ser capaz de servir a estas aplicaciones de una forma eficiente.

Los usuarios ocasionales pueden requerir de la mayor flexibilidad, ya que sus exigencias son a menudo imprevistas. Tales usuarios tienden a necesitar información para el análisis y las tomas de decisión opuestamente a las operaciones de rutina. Este tipo de usuario necesita una poderosa capacidad de clasificación y combinación de datos, así como un lenguaje de consulta fácil de usar.

Los que proponen los modelos de datos jerárquicos y en red afirman que estos modelos tienen una capacidad superior para aplicaciones de sistemas de producción (ver la discusión en los Capítulos 15 y 16). Los defensores de los SGBD relacionales dicen que estos sistemas actualmente proporcionan un rendimiento mejorado en apoyo a los sistemas de producción, además de capacidades poderosas para satisfacer las necesidades de información *ad hoc* que otros modelos no brindan. El desarrollo de IDMS/R, que ofrece propiedades relacionales en sistemas basados en red, intenta satisfacer ambas necesidades.

La mayoría de las empresas van a tener una mezcla de necesidades y pueden funcionar con el sistema que mejor soporte el tipo de necesidad principal. Los productos tales como IDMS/R intentan minimizar el esfuerzo para satisfacer la necesidad menos importante. Al mismo tiempo, los SGBD relacionales están mejorando su eficacia para satisfacer ambos tipos de requisitos de información.

Mantener la consistencia de los datos

La necesidad de compartir los datos a través de múltiples aplicaciones es la razón primaria para la implementación de la base de datos. Como se discutió en el Capítulo 1, los datos redundantes pueden generar una gran cantidad de problemas, la mayoría de los cuales se producen por inconsistencia entre las copias duplicadas de los registros de datos. Aún en instalaciones bien hechas, es común que los registros duplicados se actualicen en ciclos diferentes de tiempo. Así, en tales instalaciones, la inconsistencia de los datos es inevitable.

Un buen SGBD no puede garantizar que nunca ocurran inconsistencia de los datos, pero puede proporcionar facilidades para minimizar sus ocurrencias. Consecuentemente, cualquier evaluación de un SGBD debe incluir una consideración de las características que asegurarán la consistencia entre las copias duplicadas de los mismos datos.

Requisitos de los tiempos de respuesta

Un SGBD se debe desempeñar a un cierto nivel para que sea válido a los usuarios. Tiempos inaceptables de respuesta a las solicitudes de los usuarios los llevará a la frustración; la frustración los llevará a la búsqueda de otros medios para satisfacer sus necesidades de información. Hacer que los usuarios especifiquen tiempos de respuestas realistas puede ser un reto; sin embargo, un diálogo con los usuarios concentrados en la especificación de los tiempos de respuesta ideales y mínimo puede ser productivo. El resultado puede ser usado para evaluar el rendimiento de un SGBD bajo diferentes aplicaciones y volúmenes.

▼ Funciones y capacidades de los SGBD

Para evaluar la habilidad de un SGBD de satisfacer los requisitos de información de una empresa, necesitamos considerar las funciones que proporciona y sus propiedades fundamentales.

El diccionario/directorio de datos

Un sistema de base de datos efectivo permitirá el crecimiento y la modificación en la base de datos sin comprometer la integridad de los datos. El Diccionario/Directorio de datos (D/DD) ayuda a cumplir los objetivos permitiendo que las definiciones de datos se mantengan separadas de los datos. Esto permite que se hagan cambios en las definiciones de datos sin que tenga efecto en los datos almacenados. Por ejemplo, el subdiagrama usado por un programa en particular puede modificarse sin afectar de ninguna manera los datos almacenados. Otros beneficios almacenados por el D/DD incluyen:

- Las estructuras físicas de almacenamiento pneden cambiarse sin afectar al programa que usa los datos.
- Las contraseñas y otras medidas de seguridad pueden estar almacenadas en el D/DD para facilitar el control sobre el acceso a los datos.
- La definición centralizada de los datos permite fáciles informes sobre el estado de la base de datos: Quién es responsable de varios elementos de datos, qué controles se les aplica, y qué programas y nsuarios están accediendo a los datos.

Para producir estos beneficios, el D/DD incluye usualmente las siguientes características:

- Un lenguaje para definir las entradas en el D/DD.
- Un lenguaje de manipulación para añadir, eliminar y modificar las entradas en el D/DD.
- Métodos para validar entradas en el D/DD.
- Medios para producir informes en relación a los datos contenidos en el D/DD.

Un desarrollo importante en los SGBD relacionales es la práctica común de almacenar el directorio como un conjunto de relaciones. Esto permite el uso de los lenguajes de manipulación de datos de los SGBD para consnltar, actualizar y mantener el diccionario de datos. La Figura 13.1 muestra un fragmento de un catálogo (el nombre para un diccionario de datos usados por DB2, un SGBD relacional de IBM). Este fragmento de catálogo contiene información sobre las relaciones qne se muestran en la Figura 13.2.

TABLAS_SISTEMA	*NOMBRE*	*CREADOR*	*CANTIDADCOL*
	PRODUCTO	JHANSEN	3
	MFR	JHANSEN	4
COLUMNAS_SISTEMA	*NOMBRE*	*NOMBRETB*	*TIPOCOL*
	IDPROD	PRODUCTO	ENTERO
	DESCPROD	PRODUCTO	CARÁCTER
	IDMFR	PRODUCTO	ENTERO
	IDMFR	MFR	ENTERO
	NOMBREMFR	MFR	CARÁCTER
	DIRECCIÓN	MFR	CARÁCTER
	PAÍS	MFR	CARÁCTER

PRODUCTO

IDPROD	*DESCPROD*	*IDMFR*
1035	Abrigo	210
2241	Lámpara de mesa	317
2518	Escultura de cobre y cinc	253

MFR

IDMFR	*NOMBREMFR*	*DIRECCIÓN*	*PAÍS*
210	Kiwi Klothes	Auckland	Nueva Zelanda
253	Brass Works	Lagos	Nigeria
317	Llama Lamps	Lima	Perú

Seguridad e integridad de los datos

controles de acceso. Controles que limitan el acceso del usuario a los programas y a los datos.

controles de concurrencia. Controles que mantienen la integridad de la base de datos cuando dos o más usuarios simultáneamente solicitan un registro de la base de datos.

controles de vista. Aquellos controles que restringen el acceso a las vistas (subconjuntos de las relaciones de la base).

La seguridad e integridad de los datos son esenciales para lograr operaciones efectivas sobre las bases de datos y fueron tratadas con alguna profundidad en el Capítulo 11. Estas son también consideraciones importantes en la selección de un SGBD. Específicamente, debemos estar alerta a las siguientes capacidades:

- Los **controles de acceso** son un factor importante porque son un medio de evitar accesos no autorizados a los datos. En los entornos de bases de datos donde se comparten los datos, los buenos controles de acceso son esenciales.
- Los **controles de concurrencia** son un medio de mantener la integridad en un entorno multiusuario. Suponga que el usuario a y el b accedan ambos a un registro al mismo tiempo (esencialmente) para procesar una transacción en ese registro. El SGBD debe de alguna forma limitar el acceso de uno de los usuarios hasta que la transacción del otro termine. Sin este tipo de facilidad, la exactitud y consistencia de la base de datos puede rápidamente erosionarse.
- Los **controles de vista** proporcionan un medio automatizado de limitar que se le permita a un usuario acceder a una relación dada. Esto es una característica poderosa que se brinda comúnmente en los SGBD relacionales. La comodidad de crear vistas y la capacidad de la facilidad de vistas puede ser un factor útil de distinción entre SGBD. El comprador del SGBD puede estar interesado también en saber si las vistas pueden actualizarse y con qué limitaciones se puede hacer.

cifrado. Codificación de los datos para hacerlos ininteligibles para personas no autorizadas.

controles de seguridad y recuperación. Aquellos controles que proporcionan medios para restaurar la base de datos en caso de fallas del sistema.

- Las facilidades de **cifrado** pueden ser importantes para aquellas instituciones cuya base de datos contiene datos muy sensibles. El cifrado puede ser también importante para el mantenimiento de un directorio seguro de contraseñas.
- **Controles de seguridad y Recuperación** efectivos son absolutamente esenciales para la operación eficiente de la base de datos. La comodidad del uso de los controles de seguridad y recuperación, su completitud y su confiabilidad deben ser factores importantes en la decisión de selección de un SGBD.

Capacidades de consulta, de manipulación de datos y de informes

La habilidad del SGBD para satisfacer los requisitos de informes, junto con las consultas del usuario y las necesidades de manipulación de los datos, es la piedra angular de los sistemas de información de administración de hoy. Un SGDB que quiera "sonar" deberá proporcionar la capacidad de generar informes estructurados en una variedad de formatos. Además, el SGBD brindará un lenguaje de consulta que sea poderoso, pero a la vez fácil de aprender y usar. El lenguaje debe ser capaz de satisfacer los requisitos de consultas planeados y no planeados con tiempo de respuesta cortos.

Soporte a los requisitos de programación especializada

Desarrollar programas especializados para interactuar con los SGBD requiere de facilidades para satisfacer el desarrollo de aplicaciones y sus pruebas. Un SGBD que valga la pena proporcionará un lenguaje para expresar las estructuras estándares de programas procedimentales o brindará una interfaz con uno o más lenguajes procedimentales. Algunos SGBD pueden brindar capacidades adicionales para el desarrollo rápido de prototipos.

Opciones de organización física de los datos

La empresa que va a adquirir un SGBD puede no desear involucrarse con asuntos de la organización física de los datos. En su lugar, puede medir la eficacia de tal organización, ejecutando varias aplicaciones, ejemplos.

Sin embargo, para aquellos que están interesados, explorar la organización física puede ser valioso. Por ejemplo, es conocido que la lista invertida es más eficiente en recuperación de claves múltiples y que la lista encadenada es superior en la actualización de archivos, ya que no hay necesidad de actualizar en un archivo separado. La información sobre otras arquitecturas puede ser deseable en el proceso de considerar la capacidad del SGBD para satisfacer el tipo de aplicaciones comunes en la empresa.

▼ Clasificación de los requisitos de las propiedades de un SGBD

Las funciones y capacidades descritas en la sección anterior pueden dar una guía para determinar el criterio clave de evaluación para seleccionar un SGBD. Idealmente quisiéramos dirigir la función en sus componentes a un nivel que permita una medición objetiva. Por ejemplo, suponga que definimos las funciones necesarias de seguridad y control que son satisfechas por las siguientes características de un SGBD:

- Control de acceso para programas y usuarios.
- Tablas de seguridad protegidas por contraseñas.
- Habilidad de abortar las transacciones en progreso.
- Reconexión automática de intentos fallidos de acceder a los datos.
- Utilidades para crear y mantener tablas de contraseña.

Mediante la definición de las propiedades requeridas que puedan verificarse, el analista evita las ambigüedades que se producen al intentar comprobar si la capacidad de seguridad es buena, mala, o regular, ya que esos términos pueden tener diferentes significados para diferentes analistas. Claro, esto coloca un peso sobre el analista, que tiene que definir exactamente aquellas propiedades que constituyen la capacidad de seguridad.

CODASYL (CODASYL, 1976) recomendaba que tales propiedades fueran clasificadas en las siguientes categorías para un análisis más profundo:

- Obligatorios
- Importantes
- Opcionales
- Innecesarios
- No deseables

propiedad obligatoria. Una propiedad de un SGBD que tiene que ser proporcionada.

propiedad importante. Una propiedad de un SGBD que no es obligatoria, pero hace al SGBD más atractivo.

propiedad opcional. Una propiedad de un SGBD que tiene una importancia secundaria; puede ayudar a distinguir entre otros SGBD igualmente atractivos.

propiedad innecesaria. Una propiedad de un SGBD que no contribuye en nada al valor del SGBD para la empresa.

propiedad no deseable Una propiedad de un SGBD que le resta valor al SGBD para la empresa.

Una **propiedad obligatoria** se define como una que debe ser brindada por el SGBD. Sin ella, el SGBD candidato se desecha de una consideración posterior. Una **propiedad importante** es aquella que hace más atractivo al SGBD. Sin ella el sistema responde con menos eficacia y la implementación puede ser más difícil. Las **propiedades opcionales** tienen una importancia secundaria y son de valor, principalmente, para discriminar entre sistemas que son similares sin ellas. Una **propiedad innecesaria** no es relevante en la evaluación. Las **propiedades indeseables** son a menudo un factor en contra. Esta categoría se refiere a una propiedad que no se necesita y que degrada el rendimiento.

▼ Reunir datos sobre la disponibilidad de propiedades y sobre el rendimiento

El proceso de obtener datos para la evaluación de un SGBD requiere que se contacte al vendedor del SGBD para obtener información y para probar el SGBD en un entorno cercano al de la aplicación.

Adquirir datos de los vendedores

La información que concierne a las propiedades de un SGBD y cuán bien se comporta típicamente se adquiere (1) preguntándole a los usuarios, (2) por prueba directa, o (3) preguntándole al vendedor. Preguntarle a otros usuarios puede ser muy útil porque ellos han tenido experiencia real y no están tratando de vendernos el SGBD. Sin embargo, cuando se evalúan las opiniones de estos usuarios, la empresa debe tener cuidado, ya que otros usuarios pueden tener necesidades distintas de información y, por lo tanto, su evaluación puede no reflejar exactamente cómo el SGBD se computaría en otros entornos. Además, esta información es usualmente de naturaleza general, no proporciona evidencia detallada de los tipos de aplicaciones y su rendimiento respectivo. Cuando menos, sin embargo, tal información puede producir una lista de preguntas para hacerle al vendedor.

Las pruebas directas exhaustivas pueden ser caras y consumidoras de tiempo cuando se consideran varios SGBD. El análisis de requisitos obligatorios puede ayudar a reducir el número de SGBD a probar. Las pruebas directas producen los mejores resultados cuando imitan lo más fielmente posible las condiciones y requisitos operacionales reales.

La información dada por los vendedores puede ser falsa. Sin embargo, pueden a menudo proporcionar datos estadísticos sobre fiabilidad y rendimiento, y estar dispuestos a ejecutar pruebas demostrativas.

solicitud de propuesta. (SDP) Un documento formal que delinea los requisitos de rendimiento y le pide a los vendedores que respondan con una propuesta que satisfaga aquellos requisitos.

La vía más común de adquirir información del vendedor es emitir una **solicitud de propuesta (SDP)**. La SDP es útil especificando los requisitos que el SGBD debe satisfacer y solicita información de los vendedores para mostrar cómo sus SGBD podrán satisfacer aquellos requisitos. La respuesta a la SDP puede ser una fuente de información a comparar con la realimentación recibida de los usuarios del producto de un vendedor. Las discrepancias o inconsistencias pueden consultarse con el vendedor para su aclaración. Este proceso puede sugerir también áreas que pudieran probarse directamente. En la Figura 13.3 se muestra una lista de requisitos sugeridos en cuatro categorías.

Pruebas de evaluación (*Benchmark*)

pruebas de evaluación. Un método de comparación del rendimiento de un SGBD mediante pruebas sobre aplicaciones reales.

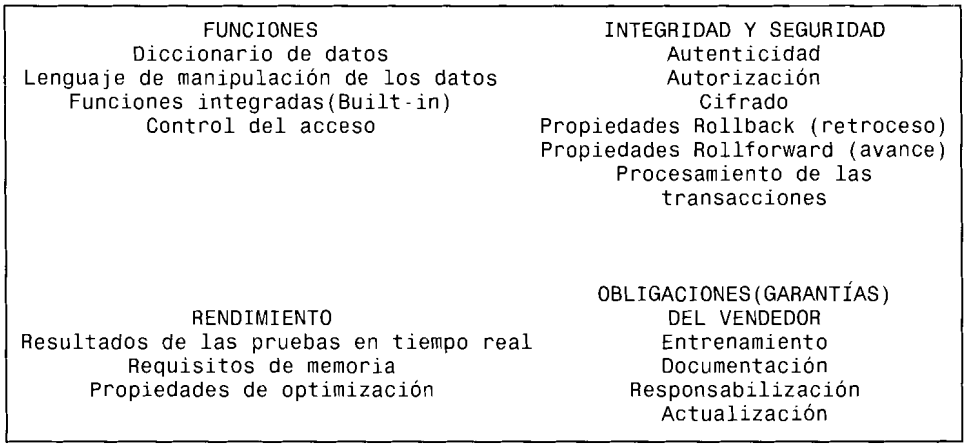
Las **pruebas de evaluación** es un método convencional para generar información sobre el rendimiento de un SGBD que luego se usa en el proceso de su evaluación. Lo que se pretende es estimular el entorno de la aplicación para generar datos de rendimiento realistas. La modelación del entorno real puede ser un reto, ya que es difícil mantener constantes todos los parámetros del entorno operativo: el sistema operativo, la secuencia de operaciones, la multiprogramación, etc. La invarianza deseada usualmente puede alcanzarse solamente mediante el uso dedicado de un computador en la prueba de evaluación. También puede ser costoso el entrenamiento de los usuarios hasta el nivel deseado con respecto a cada SGBD.

Aunque los vendedores de SGBD proporcionan a menudo pruebas de evaluación y las muestran como un servicio al posible comprador, queda como responsabilidad de éste determinar las características del sistema que son importantes y asegurar que esas características se prueben suficientemente.

Una costumbre útil es incluir como requisito en la SDP especificaciones para las pruebas de evaluación, para eliminar de antemano aquellos vendedores cuyos productos no cumplan esas especificaciones.

Cuando se hace la prueba de evaluación, hay que considerar los siguientes elementos:

- Las pruebas deben ser representativas del entorno de aplicación de la empresa.
- La naturaleza de la prueba de evaluación debe establecerse antes de hacer la prueba: requisitos a satisfacer, asignación de pesos relativos a varios componentes de la prueba y los procedimientos de evaluación que se deben seguir.



Mientras la naturaleza de las pruebas de evaluación pueden variar de una empresa a otra, hay ciertos aspectos de un SGBD que son necesarios en toda empresa: requisitos de memoria principal, de almacenamiento de la base de datos, de servicio a múltiples usuarios, que acceden al sistema concurrentemente, de entrada y salida y facilidades de seguridad y recuperación.

▼ Modelos de evaluación

La adquisición de un SGBD refleja un gran compromiso por parte de la empresa. Los métodos de recopilación y almacenamiento de los datos pueden afectarse. A un nivel más alto, esto implica un compromiso de proporcionar una mejor información de administración. Lo más importante es que la adquisición de un SGBD representa un compromiso de usar la información para mejorar la forma en que la empresa hace negocios. Estos compromisos requieren que el proceso de selección de un SGBD incluya la consideración de propiedades importantes y consistente de éstos de una manera racional. La conquista de estos objetivos puede auxiliarse con el uso de una metodología formal. En el siguiente estudio se presentan dos métodos formales: un modelo de puntuación y un modelo de envoltura de datos. El modelo de puntuación es fácil de aplicar y ha sido muy usado para la ayuda en el proceso de selección de un SGBD. El modelo de envoltura de datos es más poderoso; no obstante, es fácil de aplicar.

Modelo de puntuación

El modelo de puntuación que se presenta aquí ha sido muy usado en la práctica. Muchas empresas limitan la clasificación de requisitos recomendada a solamente dos: obligatorios y deseables. Tanto como se pueda, la verificación de requisitos obligatorios no debe estar sujeta a juicio u opinión. Los requisitos deseables pueden, y a menudo lo hacen, incluir características que son más difíciles de medir.

Conceptualmente, el modelo de puntuación es fácil. La proporción de buenos datos de entradas es la parte más difícil. Esto se ilustra aquí con un ejemplo simple, pero representativo.

Ejemplo: Divida las propiedades requeridas en dos categorías, obligatorias y deseables, como sigue:

PROPIEDAD/VENDEDOR	*A*	*B*	*C*
Obligatorios			
Vistas			
Control de contraseñas			
Facilidad de seguridad			
Deseables			
Lenguaje de consulta relacional			
Entrenamiento a los usuarios			

La selección del esquema de pesos es un poco arbitraria, pero funcionará si es consistente. Un esquema simple es asignar un peso de 10 a cada uno de los requisitos obligatorios y un peso de 1 a 9 a cada propiedad deseable, dando los números mayores en dependencia del grado mayor de deseo. Cuando esto se haya determinado, se puede asignar un valor entre 1 y 10 a cada propiedad para cada vendedor, dependiendo de cómo el SGBD del vendedor satisfaga el requisito. Una tabla de puntuación completa es así:

PROPIEDAD/VENDEDOR		*A*	*B*	*C*
Obligatorios	*Peso*			
Vistas	10	5	7	9
Control de contraseñas	10	7	7	9
Facilidad de seguridad	10	9	8	7
Deseables	*Peso*			
Lenguaje de consulta Relacional	7	10	0	10
Entrenamiento a los usuarios	9	6	7	4

Las puntuaciones finales para los vendedores se determinan multiplicando el peso de la propiedad por el valor asignado a éste por esa propiedad. Los resultados se suman para cada vendedor para calcular la puntuación final, como sigue:

PROPIEDAD/VENDEDOR		*A*	*B*	*C*
Obligatorios	*Peso*			
Vistas	10	50	70	90
Control de Contraseñas	10	70	70	90
Facilidad de Seguridad	10	90	80	70
Deseables	*Peso*			
Lenguaje de Consulta Relacional	7	70	0	70
Entrenamiento a los Usuarios	9	54	63	36
PUNTUACIÓN FINAL		334	283	356

Basándose en los resultados de este ejemplo, la empresa puede eliminar al vendedor *B* de una consideración posterior. Sin embargo, las puntuaciones de *A* y *C* son suficientemente cercanas como para realizar una consideración adicional. La diferencia pudo haberse producido por la incertidumbre al asignarle los valores al vendedor. En cualquier caso, los resultados de tales modelos normalmente tienen las dos siguientes funciones:

- Presentar alternativas para ulterior consideración.
- Brindar información para el proceso de decisión, incluyendo la necesidad de pruebas o estudios adicionales.

En la próxima sección presentamos un modelo de evaluación que proporciona una gran cantidad de información comparativa. Es un modelo fácil de usar y matemáticamente potente.

Análisis de la envoltura de datos

El análisis de la envoltura de datos (AED) es una metodología poderosa para la evaluación de alternativas de decisión tales como la selección de un SGBD, especialmente cuando hay varios criterios para la selección. Todo lo que se necesita es una comprensión básica de los conceptos y acceso al software de programación lineal, ampliamente disponible para el uso en microcomputadores.

Fundamentos. Podemos presentar los conceptos básicos del AED con un ejemplo numérico. La Figura 13.4 contiene datos que pertenecen a la graduación del rendimiento comparativo de los SGBD, desde la *A* hasta la *F* usando sólo dos criterios: control de acceso y seguridad y recuperación. Las filas de la figura identifican los seis SGBD a evaluar, y las

columnas identifican los criterios de selección. Cada criterio es medido en una escala de 1 a 10, donde 1 es el nivel más bajo de rendimiento y 10 es el más alto.

La Figura 13.5 dibuja los seis SGBD y proporciona una base gráfica para discutir cómo medir el rendimiento relativo de un SGBD. El SGBD cuyo rendimiento es bueno está lejos del origen (0) en ambas direcciones. La frontera noreste definida por *A*, *C* y *B* identifica a los SGBD dominantes en el conjunto. Esta frontera exterior es llamada la **frontera de la eficacia**.

frontera de la eficacia. La frontera exterior en el análisis AED.

El AED mide la eficacia relativa de cada SGBD por su posición relativa a la frontera. Un punto en la frontera tiene una eficacia de 1,0, y el origen tiene un valor de 0,0. Los puntos en el interior de la frontera representan valores entre 0 y 1, dependiendo de su cercanía a la misma. En la Figura 13.5(a) la capacidad de control de acceso se representa en el eje horizontal, y la capacidad de seguridad y recuperación se representa en la vertical.

Para explicar la medida de capacidad del AED, primero introduzcamos algunas definiciones adicionales. El SGBD que está siendo evaluado se le llama **unidad de referencia**, y el camino desde el origen a la unidad de referencia se le llama **camino de expansión** para esa unidad. Cualquier punto en el camino de expansión se le llama **punto de expansión**, y cualquier punto sobre la frontera se le denomina **punto de frontera**.

unidad de referencia. El SGBD que está siendo evaluado en el análisis AED.

camino de expansión. El camino desde el origen a la unidad de referencia en el análisis AED.

punto de expansión. Cualquier punto en el camino de expansión en el análisis AED.

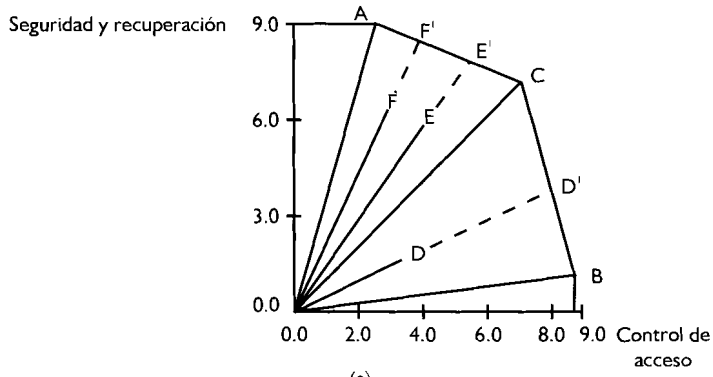
punto de frontera. Cualquier punto sobre la frontera en el análisis AED.

Considerando de nuevo la Figura 13.5(a), se puede ver que el camino de expansión para *D* comienza en el origen (O), pasa a través de *D* e intercepta la frontera en *D'*. La significación de un camino de expansión es que todos los puntos sobre él tienen la misma razón de capacidad de control de acceso a capacidad de seguridad y recuperación y, por lo tanto, pueden ser comparados directamente entre ellos. Por ejemplo, *D* tiene una razón de control de acceso de 3 y una razón de seguridad y recuperación de 2, luego (3,2) está en el camino de expansión. El punto (6,4) está también en el camino de expansión para (3,2), y un SGBD en (6,4) es doblemente mejor que uno en (3,2), ya que ambos, el control de acceso y la seguridad y recuperación son el doble de (3,2).

El punto *D'* donde el camino de expansión se cruza con la frontera tiene un factor de 2,58, lo que quiere decir que el control de acceso y la seguridad y recuperación en *D'* son 2,58 veces los valores correspondientes en *D*. La eficacia relativa de un SGBD dado es el inverso del factor de expansión en la frontera. Luego la eficacia relativa de *D* es 1 / 2,58 = = 0,39, lo que quiere decir que *D* tiene 0,39 de la capacidad de control de acceso y de seguridad y recuperación de un SGBD comparable en la frontera.

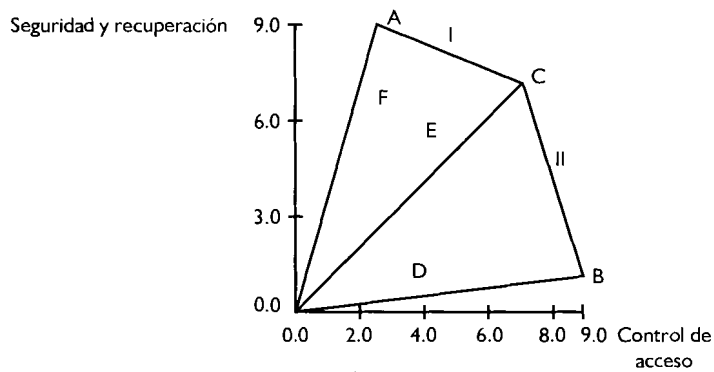
La Figura 13.5(b) muestra que el proceso de AED de medir la capacidad relativa divide en unidades similares a los SGBD que se están evaluando. La capacidad relativa de *D* se evalúa comparando el SGBD real con el *D'* hipotético que está en el segmento frontera uniendo *B* y *C*. *D*, *B* y *C* son comparativamente fuertes en la capacidad de control-del-acceso en comparación con la capacidad de seguridad-y-recuperación. Sin embargo, *E* y *F* son comparativamente fuertes en la capacidad de seguridad-y-recuperación y sus capacidades relativas se calculan comparándolos con *A* y *C*. Por lo tanto, el cono etiquetado

SGBD	CONTROL DEL ACCESO	SEGURIDAD Y RECUPERACIÓN
A	3	9
B	9	2
C	7	7
D	3	2
E	4	5
F	3	6



(a)

FRONTERA DE LA EFICACIA Y LOS CAMINOS DE EXPANSIÓN PARA LOS SEIS SGBDs



(b)

CONOS DE UNIDADES SIMILARES

como I en la Figura 13.5(b) contiene los SGBDs que son fuertes en la seguridad-y-recuperación, mientras que el cono etiquetado como II contiene aquellos que son fuertes en el control del acceso. El SGBD *C*, que es fuerte en ambas categorías, está en ambos conos.

Este método gráfico de medición de la capacidad relativa es una herramienta valiosa para ilustrar los conceptos del AED, pero está limitado por dos criterios. Para medir la capacidad relativa de los SGBDs caracterizados por más de dos criterios debemos usar modelos de programación lineal (PL). La formulación más real y práctica se basa en sopesar los valores relativos de los criterios.

Aplicación. Ser capaz de incorporar opiniones acerca de la importancia relativa de los criterios constituye el corazón para la toma de decisión sobre qué SGBD escoger. Continuando con nuestro ejemplo, *A* es el mejor en la seguridad y recuperación, pero es débil en el control del acceso. *B* es el mejor en el control del acceso, pero débil en la seguridad y recuperación. Ambos están en la frontera y tienen una capacidad relativa de 1,0 en la medición del AED. Sin embargo, para comparar la capacidad de rendimiento total de *A* y *B*, necesitamos emitir algunos criterios sobre la importancia relativa de la capacidad de control del acceso y la capacidad de seguridad y recuperación.

Una prueba basada en el modelo de PL nos permite centrar la atención en la importancia relativa del criterio. Las variables de decisión descansan sobre los atributos, que expresan la importancia relativa de los mismos. El objetivo consiste en usar el modelo de

			PESO (CONTROL DEL ACCESO) 1 (SEGURIDAD Y RECUPERACIÓN) 2		PESO (CONTROL DEL ACCESO) 3 (SEGURIDAD Y RECUPERACIÓN) 2	
DBMS	CONTROL DEL ACCESO	SEGURIDAD Y RECUPERACIÓN	PUNTUACIÓN TOTAL	EFICACIA RELATIVA	PUNTUACIÓN TOTAL	EFICACIA RELATIVA
A	3	9	21*	1,000	18	0,621
B	9	2	13	0,619	29*	1,000
C	7	7	21*	1,000	28	0,966
D	3	2	7	0,333	11	0,379
E	4	5	14	0,667	17	0,586
F	3	6	15	0,714	15	0,517
			MAX = 21		MAX = 29	

PL (programación lineal) para escoger el criterio con pesos que maximicen la eficacia relativa de la unidad de referencia. La capacidad relativa se mide como un radio del valor total de la unidad de referencia comparándolo con el valor completo de lo mejor en el conjunto de los SGBDs. La Figura 13.6 muestra cómo la selección del peso de los criterios afecta a la capacidad relativa. Cuando la capacidad de seguridad-y-recuperación es dos veces más importante que la capacidad de control-del-acceso, *A* y *C* poseen la mejor capacidad total, cada uno con una razón total de 21[= 1x(razón de control-del-acceso) + 2x(razón de seguridad-y-recuperación)]. La capacidad relativa de cada SGBD se mide como el radio de la razón total frente a la razón total más alta. Con estos valores (pesos), *B* se considera en un 61,9 por 100 (=13/21) tan bueno como *A* y *C* en términos de capacidad.

Las dos últimas columnas nos guían hacia un resultado bastante diferente, ya que se supone que la capacidad de control-del-acceso es tres veces más importante que la capacidad de seguridad-y-recuperación. Con estos valores (pesos), *B* tiene la puntuación total más alta, 29. *A* tiene ahora una capacidad relativa de un 62,1 por 100 (18/29), mientras que *C* se incrementó hasta un 96.6 por 100 (28/29).

D, E, y *F* se mantienen con razones de capacidad total bajas para ambos conjuntos de valores (pesos). En realidad, las unidades que no se encuentran en la frontera nunca serán las mejores para ningún conjunto de valores (pesos) y siempre tienen una capacidad relativa menor que 1,0. Por el contrario, para cualquiera de los SGBDs en la frontera, existe al menos un conjunto de valores (pesos) para el cual el SGBD frontera tiene la mejor puntuación total, y por lo tanto, una eficacia relativa de 1,0.

Esta observación nos guía directamente a la utilización del modelo PL para medir la eficacia relativa. Las variables de decisión son los pesos criterios y el objetivo es maximizar la capacidad relativa de una determinada unidad de referencia. Las limitantes son que cada SGBD tiene una capacidad relativa que no excede el 100 por 100. La expresión verbal es:

MAXIMIZAR EL VALOR DE LA UNIDAD DE REFERENCIA
DE ACUERDO CON EL VALOR COMPLETO DE CADA UNA
DE LAS UNIDADES MENORES O IGUALES QUE 1.

El modelo completo basado en los criterios para medir la capacidad relativa de *D* es

MAX $3P + 2Q$	(Razón total de *D*; *P* = razón del control del acceso; *Q* = razón de seguridad y recuperación)
ST $3P + 9Q <= 1$	(Límite de la razón total de *A*)

$9P + 2Q <= 1$	(Límite de la razón total de B)
$7P + 7Q <= 1$	(Límite de la razón total de C)
$3P + 2Q <= 1$	(Límite de la razón total de D)
$4P + 5Q <= 1$	(Límite de la razón total de E)
$3P + 6Q <= 1$	(Límite de la razón total de F)
$P,Q >=0$	(no negativos)

▼ Cuestiones de implementación

La planificación y la administración de la implementación es tan importante para un sistema de base de datos como lo es una implementación efectiva para cualquier tecnología nueva. En esta sección definiremos algunas consideraciones importantes asociadas con la implementación de un SGBD.

Administración de la base de datos

La responsabilidad de la administración de una base de datos normalmente se asigna a una persona llamada administrador de la base de datos (ABD). El ABD es el encargado de asegurar que el sistema de base de datos opere de forma efectiva y eficiente. Con este objetivo, las actividades diarias de los ABDs se concentran en las tareas siguientes:

1. Requisitos del servicio al usuario final.
2. Garantizar la seguridad e integridad de la base de datos.
3. Establecer los procedimientos de seguridad-y-recuperación.

Requisitos del servicio al usuario final. El ABD es el responsable de seguir la frecuencia con que los usuarios finales utilizan la base de datos y los tiempos de respuestas que necesitan sus aplicaciones de base de datos. Esto se realiza para asegurarse que las necesidades del usuario estén satisfechas. El ABD también está involucrado en la satisfacción de las necesidades de entrenamiento de los usuarios. Este entrenamiento brinda a los usuarios finales las herramientas apropiadas que les permiten ser más productivos. Un ejemplo podrían ser las utilidades especiales o las capacidades del lenguaje.

Garantizando la seguridad e integridad de las bases de datos. Siguiendo la frecuencia con que los usuarios usan la base de datos, el ABD obtiene información necesaria para el control de la base de datos. Por ejemplo, una consulta inesperada de un usuario puede provocar una reevaluación de la correción del control del acceso. Por ejemplo, suponga que el usuario U1 no tiene permiso para acceder a los registros individuales en el archivo de nómina. Recientemente, U1 ha estado consultando datos adicionales, como son cuentas y sumas de varios campos en la tabla. U1 puede obtener datos desde los cuales él mismo puede deducir valores de atributos individuales que no pueden ser accedidos directamente.

En términos de la integridad de la base de datos, el ABD debe asegurarse que existe una correspondencia entre la base de datos y su definición reflejada en el diccionario de datos. El ABD debe también mantener el control para restringir la actualización de la base de datos por usuarios autorizados. El mantenimiento de los niveles de restricción coloca por encima de cada usuario la responsabilidad del ABD.

El ABD debe ayudar a garantizar que el control de los datos de un usuario-específico esté implementado y supervisado. Por ejemplo, los usuarios pueden tener un mejor conocimiento que el ABD acerca de qué datos son sensibles y se debieran restringir y a qué nivel. Por ejemplo, el acceso a datos de costo pertenecientes a operaciones de manufactura puede no estar restringido al departamento de contabilidad, pero a su vez no debe ser accedido por otros grupos funcionales sin permiso previo.

Seguridad(*backup*)-y-recuperación. Los SGBDs brindan facilidades para el mantenimiento de la integridad de los datos durante las operaciones. Sin embargo, por otro lado, no pueden facilitar la integridad de la base de datos. Por ejemplo, la corrupción de los datos debido a un fallo en el disco puede estar fuera del control del SGBD. Estas posibilidades hacen que el ABD disponga de métodos para restaurar la base de datos completa y consistentemente después de un fallo.

Los procedimientos de restauración involucran típicamente utilidades suministradas por el SGBD como parte de un largo plan de recuperación diseñado por el ABD. Esto significa que el ABD debe familiarizarse con todas las funciones y capacidades del SGBD. El plan de recuperación generalmente incluye los procedimientos siguientes:

1. Identificar las utilidades de seguridad y recuperación que están disponibles para manejar los fallos del sistema.
2. Establecer los procedimientos a seguir por todos aquellos que participan en el plan de recuperación.
3. Crear un plan para informar a los usuarios que están afectados por un fallo en la base de datos debido al impacto potencial que el mismo puede tener sobre sus operaciones, así como estimar el tiempo para restaurar la base de datos.
4. Crear y analizar una planificación de pruebas para el plan de recuperación y normalizarlo para poder medir los resultados. Esto garantiza que el plan de recuperación cumpla sus objetivos en caso que ocurra un desastre real.

Probar la base de datos

Para garantizar que el sistema de base de datos cumpla con los requisitos de los usuarios y opere sin mayores problemas se necesitan buenos procedimientos de pruebas. En el desarrollo de aplicaciones nuevas, las pruebas de ejecución sólo garantizan la programación en términos de tiempo necesario. Las pruebas de validación normalmente se centran en las operaciones siguientes:

1. La carga de la base de datos se realiza sin violar la integridad de los datos.
2. La correcta interfaz de las aplicaciones con la base de datos.
3. El rendimiento del sistema satisface las necesidades para las que se adquirió el SGBD.

Un objetivo de las pruebas, que es muchas veces pasado por alto, aparece cuando el sistema de base de datos no funciona como se esperaba. La mayoría de los problemas surgen cuando los valores de entrada son inusuales o erróneos, cuando las combinaciones de condiciones obtienen un resultado inesperado, o cuando los usuarios toman decisiones anticipadas. Por lo que se deben generar datos de prueba para determinar cómo el sistema manipula los tipos de entradas siguientes:

1. Valores que tienen signos negativos erróneos.
2. Códigos o claves inválidas.
3. Datos que amenazan la integridad referencial.
4. Valores nulos.
5. Un intento de cambiar los datos por un usuario no autorizado.
6. Valores extremos, como son una inusual gran cantidad de horas trabajadas o grandes recibos de compra inusuales.
7. Unidades de medición incorrectas.
8. Intento no autorizado para cambiar la información que se encuentra en el diccionario de datos.

Acerca del nivel de control operacional, la empresa debe preocuparse por la supervisión del uso de los recursos de computación. La admiuistración debe interesarse en la información siguiente:

1. ¿Quién está usando el sistema de base de datos, qué datos están siendo usados y durante qué tiempo serán usados?
2. ¿Qné recursos del hardware está usaudo el sistema de base de datos, cuándo son usados y si existen cuellos de botella?
3. ¿Cuánto tiempo se usa por una rutina de procesamiento en comparación con una consulta *ad hoc*, y si existen tendencias en este uso?

Para resumir estas ideas ilnstramos algunas de las medidas de rendimiento incorporadas en el SGBD relacional de IBM DB2. El DB2 Performance Monitor (Supervisión de Rendimiento de DB2) se usa en conjunto con la DB2 Instrumentation Facility (Facilidad de Instrumentación de DB2) para generar información sobre el rendimiento del sistema. La DB2 Instrumentation Facility colecciona los datos de rendimiento del sistema de base de datos, que se analizan por el DB2 Performance Monitor con el objetivo de generar información de interés para el ABD y otro personal de administración. Ejemplos de dicha iuformación son el tiempo de procesamiento de una consulta SQL, seguimientos de la ejecución de una consulta SQL, estadísticas de los datos de entrada y salida, e información resumen sobre los tiempos de espera para la ejecución de consultas y programas.

La generación de datos de prueba que verifiquen todas las posibles condiciones no deseadas se muestra como un problema intratable (Garey y Johnson, 1979). Por consiguiente, una prueba robusta (completa) de un sistema de base de datos necesita un diseño bien pensado de los datos de prueba que sean a la vez razonablemente completos y que se puedan aplicar a un costo y tiempo razonable.

Las tablas de decisión se usan con éxito en la generación de datos de prueba.

Muchas soluciones sofisticadas realizan una generación automática de los datos de prueba.

Preparar a los usuarios para el cambio

Puesto que la aceptación del usuario es esencial para una implementación exitosa del sistema de base de datos, la preparación de los mismos para los cambios que pudieran afectarlos es también esencial. ¿Cómo se cumplimenta esto mejor?

Conceptualmente, la respuesta es simple: Involucrar a los usuarios en el desarrollo de nuevos sistemas, entrenarlos cabalmente, e incluirlas pruebas de aceptación como parte del esfuerzo de implementación. Llevar a cabo estas actividades sería algo más complejo.

Una gran parte de la incorporación de los usuarios en el desarrollo del sistema de base de datos se facilita si los usuarios están convencidos de esta necesidad para el sistema. El administrador jefe, junto con los analistas funcionales y los analistas de sistema, deben comunicarle claramente a los usuarios cuáles serían los beneficios esperados, así como cualquier costo de puesta en marcha que pudiera ser asumido por el usuario. Esto se debe realizar lo más temprano posible en el ciclo de desarrollo.

La experiencia ha mostrado que los usuarios se identifican con el sistema cuando participan en su desarrollo e implementación. Se seleccionan usuarios claves como parte del equipo de desarrollo, y otros se consultan a medida que avanza el proyecto. Los usuarios necesitan participar y contribuir. El objetivo no es simplemente político. Algunos usuarios contribuyen con ideas y observaciones que han sido pasadas por alto y que pudieran mejorar el funcionamiento del sistema. Enfatizamos en que el desenvolvimiento del usuario es más efectivo cuando participa en el desarrollo e implementación. Una consulta de cortesía al comienzo y al final del proyecto raramente funciona.

Involucrar a los usuarios en el desarrollo e implementación del sistema de base de datos brinda otro beneficio: los prepara para el entrenamiento en el uso del sistema. Posteriormente los usuarios pueden recibir el entrenamiento en el uso del sistema mediante (1) la lectura de los manuales de operación y procedimientos, (2) sesiones de entrenamiento formales dirigidas por el vendedor y la empresa, y (3) entrenamientos informales a cargo de un coordinador interno para asistir a los usuarios a medida que surgen interrogantes y problemas. Si un mismo tipo de interrogantes surgen en varias ocasiones, el coordinador debe organizar una sesión de entrenamiento formal para establecer los procedimientos del problema.

Entre otras cosas, los usuarios necesitan conocer cómo procesar los documentos, entrar los datos, producir informes, corregir errores, generar consultas, entre otras cosas. En la medida en que los usuarios se entrenan, pueden utilizarse como recurso para entrenar otros usuarios en su área funcional.

Cargar la base de datos

Frecuentemente, los datos que se almacenan en una base de datos ya existen sobre algún medio en la computadora, como es una cinta magnética. En el mejor de los casos, todos los datos necesarios existen, y la carga de la base de datos consiste simplemente en la reestructuración de los datos existentes. Es decir, un programa debe leer los archivos viejos y crear la estructura que necesitan los nuevos.

Es más frecuente que los archivos existentes no contengan todos los datos necesarios para la nueva base de datos. Una de las soluciones usada comúnmente para este caso consiste en transformar los archivos existentes, de tal forma que contengan los datos necesarios antes de realizar el cambio en el sistema de base de datos. Esta solución posee una limitación digna de mencionar y es que el viejo sistema debe continuar procesando las transacciones diarias mientras que se realiza la modificación.

Mantenimiento de la base de datos

Una vez que el SGBD está instalado y en operación, se necesita organizar y ejecutar las actividades de mantenimiento para garantizar los servicios y operaciones del mismo. En esta sección se describen algunas de las funciones de mantenimiento necesarias.

Administrando recursos. Todo el tiempo se añaden datos nuevos a la base de datos, algunos se eliminan, y las aplicaciones se actualizan o modifican. La mayoría de las empresas experimentan que sus bases de datos crecen. Nuevos requisitos pueden cambiar la mezcla de las aplicaciones de bases de datos, y es importante supervisar el efecto de tales cambios. Los recursos de los sistemas de bases de datos, como son los dispositivos de almacenamiento, buffers, índices y tablas, se chequean para determinar su adecuación a medida que cambian los requisitos del sistema. Esta información nos ayuda a la hora de tomar decisiones acerca de la adición o actualización de recursos, así como para prever necesidades futuras.

Seguridad y recuperación. Salvar una copia de la base de datos cada ciertos intervalos de tiempo debe ser una práctica estándar de seguridad. De esta forma, si la base de datos que se está usando se daña o se destruye, se puede reconstruir cargando la copia de seguridad y reevaluando todas las transacciones que se procesaron después que se hizo la copia de seguridad.

Las salvas de la base de datos se realizan usualmente mediante un utilitario del sistema operativo o un utilitario del SGBD. El primero hace una copia de seguridad del archivo de disco completo, incluyendo las estructuras de datos físicas. Una copia de seguridad similar es posible a través del SGBD. Sin embargo, las estructuras de los datos no se representan realmente en los datos salvados, sino que se reconstruyen durante la carga de los mismos. Este proceso puede ser lento para archivos largos.

Administrando los cambios al sistema de bases de datos. Los cambios en el sistema de bases de datos son inevitables debido a que las necesidades del usuario cambian todo el tiempo. Los usuarios se pueden acostumbrar al lenguaje de consulta y por lo tanto extender el rango y la complejidad de las aplicaciones requeridas por el sistema. A medida que pasa el tiempo, los usuarios ven nuevos caminos mediante los cuales se puede organizar y actualizar la información de la base de datos para satisfacer las necesidades de sus trabajos. En la medida en que estos cambios se acumulan, se necesitan mejoras en el sistema de base de datos.

Los cambios en las necesidades de los usuarios que lleguen a organizar los datos en formas más complejas pueden degradar el rendimiento del SGBD si están dispersos, pero pueden no requerir que se cambie el contenido y la estructura de la base de datos.

Sin embargo, las aplicaciones nuevas pueden necesitar que se añadan atributos a las tablas, se creen nuevas tablas, se formulen nuevas vistas, se construyan índices adicionales, entre otros.

Cambios como éstos pueden afectar las operaciones y degradar el rendimiento del sistema. No existe una solución de golpe a esta situación, pero una exitosa implementación de la base de datos contribuye a realizar el proceso de cambio ordenadamente y sujeto al análisis y la gestión involucradas. La administración se invoca cuando una consulta de cambio del usuario degrada la capacidad y rendimiento del sistema de base de datos para otro usuario.

Ya que las decisiones de cambios entrañan efectos secundarios, es útil supervisar las nuevas implementaciones con el objetivo de medir su impacto real. La supervisión nos ayuda a acumular experiencia en la predicción de los efectos de los cambios necesarios en el futuro.

Facilidades para supervisar los SGBD. Los SGBDs presentan capacidades de incompatibilidad. En esta sección daremos una visión panorámica de las facilidades de supervisión más representativas que puede contener un SGBD genérico que llamamos *G*.

G brinda capacidades en línea para la comprobación de las actividades actuales del SGBD. Un menú permite la selección de las funciones de supervisión, como son:

1. Uso de la entrada/salida por el usuario.
2. Cantidad de operaciones de lectura y escritura ejecutadas en un período de tiempo prefijado.
3. Cantidad de transacciones completadas por un período de tiempo prefijado.
4. Brindar el seguimiento de actividades de interés. Por ejemplo, puede desearse comprobar quién está modificando los datos en la TABLA NÓMINA. Esto se puede realizar con la orden siguiente:

```
AUDIT UPDATE ON NÓMINA BY SESSION
(INSPECCIONA LA ACTUALIZACIÓN SOBRE NÓMINA POR SESIÓN)
```

La cláusula ON (SOBRE) identifica la tabla que será inspeccionada, y la cláusula BY (POR) cuan fino es el nivel de inspección. BY SESSION, por ejemplo, escribe una entrada hacia el camino a inspeccionar para cada *sesión* del usuario, accediendo a la relación inspeccionada. Como puede verse, los SGBDs necesitan tener un rango razonable de capacidades de supervisión.

En este capítulo hemos establecido las tareas de selección e implementación de los SGBD. Nuestro enfoque ha sido comenzar con la gestión de las necesidades de información, seguido por los tipos de requisitos del sistema que estas necesidades implican. Entonces caracterizamos las funciones básicas de los SGBD que están disponibles para mantener dichos

requisitos y ejemplificamos cómo subdividirlos en características fundamentales a un nivel que nos permita una medida de evaluación objetiva. Después se estudió una simple forma de hacer un análisis inicial del SGBD, en el que se clasifican sus características de acuerdo con su importancia. Continuamos con un análisis de las fuentes de información sobre los SGBDs comerciales.

Luego se consideraron dos métodos de evaluación total del SGBD: el modelo de puntuación multiatributo, que es simple y bien conocido y el análisis del recubrimiento de datos, que es menos conocido, pero que es útil por su facilidad de uso y porque brinda capacidades poderosas para manejar la entrada de toma de decisiones y dar como salida información comparativamente valiosa. El mismo es particularmente efectivo en la categorización de los SGBDs, cuyas capacidades totales son similares.

Después discutimos los factores principales que deben formar parte de la estrategia de implementación del SGBD. Muchos de estos factores tienen que ver con la gestión de SGBD a largo plazo también. Vimos que la función de administración de la base de datos juega un papel importante en la implementación y que es esencial una prueba robusta del mismo. También consideramos la necesidad de preparar al usuario para participar en los cambios, a través de su participación en el proyecto, así como en los entrenamientos.

Se hizo un resumen de la necesidad de mecanismos para acomodar las necesidades de cambios en la información de la base de datos, junto con los requisitos de seguridad y recuperación. Finalmente, se destacó el papel que juega la supervisión de la ejecución.

1. Defina cada uno de los términos siguientes con sus propias palabras:
   a. gestión de información
   b. controles de acceso
   c. encriptado
   d. controles de seguridad-y-recuperación
   e. propiedades importantes
   f. propiedades opcionales
   g. propiedades innecesarias
   h. solicitud de propuesta (SDP)
   i. frontera de la eficacia
   j. punto de expansión
   k. punto frontera

2. Describa cada una de las siguientes funciones y capacidades del SGBD:
   a. directorio/diccionario de datos
   b. seguridad e integridad de los datos
   c. consulta, manipulación de datos, y capacidades de informe
   d. soporte de los requisitos de programación especializada
   e. opciones de organizaciones físicas de datos

3. Identifique los rasgos de importancia para cada una de las funciones en (2).

4. ¿Cómo debe usarse la salida de un modelo analítico para seleccionar un SGBD?

5. Describa cómo influye el modelo de puntuación en la selección de un SGBD.

6. ¿Qué valor hay en incluir clasificaciones tales como INNECESARIA e NO DESEABLE?

7. Estudie las cuestiones de mayor importancia en la implementación de un SGBD.

1. Marque cada término con su selección:

—*controles de concurrencia*	a. Controles que limitan el acceso del usuario a un subconjunto de una tabla base.
—*controles de vista*	b. Probando el rendimiento de un SGBD ejecutando una aplicación sobre él.
—*propiedad obligatoria*	c. En el análisis AED, el SGBD que se está evaluando.
—*propiedad indeseable*	d. La propiedad de un SGBD que aleja su valor hasta la empresa.
—*evaluación (benchmarking)*	e. La frontera externa en el análisis AED.
—*unidad referencia*	f. Controles que mantienen la integridad de la base de datos cuando dos o más usuarios consultan simultáneamente un registro de la base de datos.
—*camino de expansión*	g. En el análisis AED, el camino desde el inicio hasta la unidad referencia
	h. Una propiedad de los SGBD que se debe proporcionar.

2. Obtenga información sobre SGBDs en red. (Esto puede encontrarse en la biblioteca.) ¿Estos SGBDs abarcan las funciones descritas en este capítulo? ¿Existe información sobre las propiedades que habilitan esas funciones? ¿Puede crear una clasificación genérica de las propiedades para las empresas de una industria en particular?

3. Obtenga información acerca de dos SGBDs relacionales y repita el proceso descrito en el ejercicio 2. Entonces busque información similar en un modelo jerárquico o un modelo en red y compárelo con uno de los sistemas relacionales. ¿Es más fácil o más difícil encontrar información similar? ¿Por qué?

4. Usando los datos de dos o tres de los SGBDs anteriores, cree un modelo de puntuación. Justifique su selección de los criterios de pesos y razones del SGBD.

5. Los programas de programación lineal (LP) están ampliamente difundidos en las escuelas de negocios y departamentos de ciencia de la computación, a menudo sobre un microcomputador. Use un LP para analizar los datos de los SGBDs A-F en el capítulo. Podría usar los datos tal y como están, sin los pesos. Después repita el análisis usando los pesos. Finalmente experimente con algunos pesos propios.

6. Después de completar el ejercicio 5, trate de aplicar el AED al análisis desarrollado en el ejercicio 4.

7. Basado en este capítulo, escriba una guía pequeña que se pueda usar en la selección de un SGBD en una compañía hipotética.

8. Escriba una guía breve acerca de los procedimientos de implementación de los SGBD.

1. Contacte con una o dos instituciones que usen SGBDs. Encuentre qué enfoque usan en la selección de sus SGBD. ¿Están satisfechos con el resultado?

2. Contacte con un vendedor para adquirir datos sobre su SGBD. ¿Qué información contienen estos datos que le resultaría útil en el proceso de seleccionar un SGBD?

PARTE

CINCO

TEMAS AVANZADOS

La Parte 5 está compuesta por un solo capítulo que trata los avances recientes en el desarrollo de los sistemas de bases de datos. Se tratarán las bases de datos orientadas a objeto y los sistemas basados en el conocimiento, ambos ofrecen importantes capacidades que deben elevar el nivel de las bases de datos hacia niveles más altos. Las bases de datos orientadas a objetos son capaces de *incorporar complejidades que sobrepasan a las de los sistemas de bases de datos relacionales*. Los sistemas basados en el conocimiento aplican la lógica de predicados a los sistemas de bases de datos relacionales para poder inferir información adicional de alto nivel a partir de los datos que están realmente almacenados en la base de datos.

Este es un trabajo excitante que nos permitirá poner al lector en el estado del arte del desarrollo de los sistemas de bases de datos.

CAPÍTULO

14

SISTEMAS AVANZADOS: SISTEMAS DE BASES DE DATOS ORIENTADOS A OBJETOS Y SISTEMAS BASADOS EN EL CONOCIMIENTO

Una evolución hacia los sistemas de bases de datos orientados a objeto
- Intento de proporcionar una representación de los datos más poderosa

La contribución de la programación orientada a objetos
- Abstracción de clases
- Clases derivadas y herencia simple
- Agregación
- Un ejemplo extendido que demuestra la herencia múltiple

Desarrollo de los sistemas de bases de datos orientadas a objetos
- GemStone
- Vbase
- Orion
- PDM
- IRIS
- O_2

Un lenguaje genérico orientado a objetos
- El modelo de datos del Lerner College

Formular consultas con TextQuery
- El formato básico de una consulta
- Soluciones en TextQuery
- Definir direcciones de las interrelaciones
- Navegar sobre los atributos y las interrelaciones
- Asignar nombres a conjuntos derivados
- Expresiones calificadas de comparación de conjuntos
- Conectores booleanos

Introducción a los sistemas basados en el conocimiento

Conocimiento y bases de datos

Representación del conocimiento con reglas
- Formulación de reglas
- Reglas en PROLOG

Una aplicación simple de base de datos en PROLOG
- Aspectos fundamentales de PROLOG
- La estructura de una aplicación PROLOG
- Aplicación de bases de datos

Datalog

Lenguaje de datos lógico (LDL)

Resumen

Preguntas de repaso

Problemas y ejercicios

Proyectos y cuestiones profesionales

Sanford Mallon, CIO de International Product Distribution, estaba almorzando con Billy Clark, quien fue su compañero de aulas en la Obelisk University. Billy es ahora el administrador de bases de datos de Simpson Technologies, una firma de manufacturas de alta tecnología. Acaba de regresar de una visita a la sede de su vendedor de SGBD, Magicware, y está entusiasmado con algunos de los nuevos desarrollos que ha visto. "Tú sabes, Sandy, que estamos muy contentos con las mejoras en la información de gestión que hemos realizado para nuestro sistema actual de base de datos y nuestra dirección ha estado muy receptiva en su utilización. Pero ahora esa misma dirección nos está preguntando si podemos hacer más."

Sanford le respondió: "¿Qué más puede hacer un sistema de bases de datos? Estamos usando el mismo SGBD y creemos que estamos operando con lo más actualizado."

"Bien", respondió Billy, "tenemos uno o dos 'nuevos iluminados' que están preguntando sobre sistemas de bases de datos orientados a objeto y sobre sistemas basados en el conocimiento. Yo nunca había oído hablar de eso, por lo que llamé a Magicware para preguntar qué es lo que sabían. Resulta que están desarrollando un producto, un sistema basado en el conocimiento, que realizará las funciones usuales de gestión de bases de datos a la vez que le incorpora la tecnología orientada a objetos y "experiencia" en forma de reglas que operan sobre la base de datos. Realmente no entendía esto hasta que no visité las oficinas de Magicware y vi una demostración. Ahora estoy muy entusiasmado en presentarles esas capacidades a nuestra dirección."

"Espera un minuto, "dijo Sandy..."

Los avances en la **programación orientada a objetos** (OOP)[1] está conduciendo los desarrollos actuales de los SGBD de objetos (SGBDO)[2] de modo que se puedan manejar objetos complejos, herencia y otras características que permitan la implementación directa de los modelos conceptuales orientados a objeto. Al mismo tiempo, la tecnología de las bases de datos relacionales está siendo extendida para combinar las capacidades de gestión de los datos con la aplicación de reglas lógicas para brindar información más refinada para la gestión. Tales sistemas se llaman *sistemas basados en el conocimiento* (SBC)[3].

En este capítulo se hará una revisión de algunas de las características que han motivado estos desarrollos y se discutirán algunos ejemplos y cómo extienden las capacidades de los sistemas de bases de datos existentes. Después de leer este capítulo deberá ser capaz de:

- Comprender las características básicas de la OOP y cómo ésta se relaciona con los SGBD.
- Comprender cómo los modelos conceptuales del Capítulo 4 se relacionan con la OOP y los SGBDO.
- Discutir algunos de los principales SGBDO que están en el mercado o que están en desarrollo.
- Familiarizarse con un lenguaje genérico de consulta para un SGBDO.
- Explicar cómo los sistemas basados en el conocimiento pueden extender el poder de los sistemas de bases de datos relacionales para brindar información para la gestión.
- Identificar las formas en que el conocimiento se puede representar mediante reglas.
- Discutir lenguajes como PROLOG y LDL y explicar cómo han influido en el desarrollo de los sistemas basados en el conocimiento.

[1] Conocido en inglés por las siglas OOP (*object oriented programming*) (N. del T.).
[2] Conocido en inglés por las siglas ODBMS (*object database management system*) (N. del T.).
[3] Conocido en inglés por las siglas KBS (*knowledge base systems*) (N. del T.).

▼ Una evolución hacia los sistemas de bases de datos orientados a objeto

Intentos de proporcionar una representación de los datos más poderosa

Con todo la potencialidad y éxito de los sistemas de bases rde datos relacionales, hay algunas aplicaciones cuya complejidad no se puede manejar bien con el modelo relacional. Es típico en ese tipo de aplicaciones involucrar datos que están altamente interrelacionados, tales como definiciones de productos, multimedia o descripciones de materiales. La evolución de los SGBDO es un esfuerzo para tratar esas complejidades. Los desarrollos de los SGBDO se han debido fundamentalmente a la génesis y al refinamiento de la OOP.

La contribución de la programación orientada a objetos (OOP)

La Contribución de la Programación Orientada a Objetos (OOP) al desarrollo de los SGBDO es considerable. Los lenguajes de OOP tienen más expresividad que los típicos lenguajes de bases de datos. Sin embargo, les falta dar mayor soporte a la persistencia de los objetos —objetos que necesitan existir más allá del alcance de una interfaz particular del usuario con la base de datos—. Esto naturalmente ha producido desarrollos, tales como el SGBDO Gemstone (Maier y Stein, 1992), que intenta extender las capacidades de un lenguaje OOP para incluirle las capacidades de gestión de datos de los SGBD.

En el Capítulo 4 se usaron ciertas ideas fundamentales que se derivan del desarrollo de la programación orientada a objetos (OOP). En particular los "conjuntos conceptuales" son análogos a las "clases" de la OOP sin las funciones miembros. Las instancias de los conjuntos conceptuales son como los objetos de la OOP. En la OOP, una clase se define por las propiedades siguientes:

1. Un conjunto de variables (atributos) que contienen los datos para un objeto de la clase.
2. Bloques de código llamados funciones que realizan las operaciones sobre los datos de los objetos de la clase.
3. Un conjunto de mensajes frente a los cuales responden los objetos de la clase.

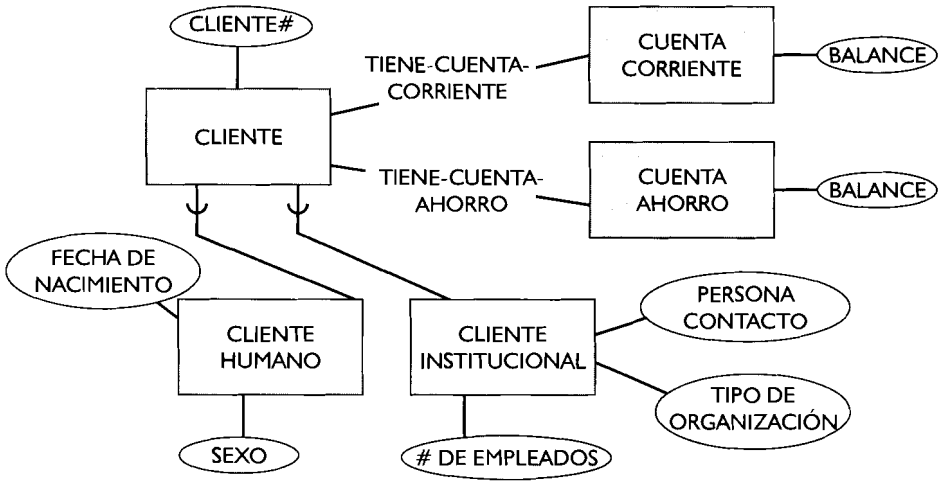
En el modelado conceptual tenemos todas las estructuras de datos de una clase, pero no se definen ninguna de las funciones o mensajes que pertenecen a dichas clases. La implementación requiere añadir tales características.

Los SGBDOs, tales como Gemstone, combinan las capacidades de los lenguajes de OOP con las funciones de almacenamiento de los SGBD tradicionales. En la siguiente sección se destacan algunas de estas características básicas.

Abstracción de clase

La Figura 14.1 es una reproducción de la Figura 4.21 que se estudió al comienzo. Podemos ver que CLIENTE es una clase que tiene el atributo CLIENTE#. Esto se representa de la forma siguiente:

```
class CLIENTE {
protected:
int CLIENTE#;
      }
```



clase. Una representación abstracta (plantilla) de un conjunto de objetos.

Usamos una sintaxis de C++ (C++ se usa ampliamente como lenguaje de OOP). Los elementos del lenguaje que necesitamos conocer son bastante fáciles de seguir. Por ejemplo, en el código anterior no es difícil ver que el término **class** denota una declaración de clase. CLIENTE es la etiqueta que le hemos dado a esa clase y el código encerrado entre claves define los miembros de la clase. En este caso; el único miembro es CLIENTE# que requiere un tipo de datos —se ha escogido entero (int)—. La coma "," indica simplemente el fin de una instrucción.

Las clases son abstracciones en el mismo sentido que los conjuntos en el modelo conceptual: Ellas permiten una representación de aquellas características de la realidad que son importantes en el problema que se está tratando. Estas clases permiten definir un número finito de objetos de dicha clase. El siguiente código es ilustrativo:

```
class CLIENTE {
      protected:
          int CLIENTE#;
      }

main( ) {
       CLIENTE cliente1, cliente2, cliente3;
            }
```

Se ha añadido una función llamada " main" que crea tres objetos de la clase CLIENTE: cliente1, cliente2, cliente3. Hasta ahora esta clase CLIENTE tiene sólo miembros que son datos y no tiene funciones, ésta podría extenderse fácilmente para incorporarle miembros adicionales a la clase. El punto aquí ha sido demostrar que, una vez que se ha definido una clase, es fácil crear objetos de esa clase. Si piensa en esto un momento, puede imaginar a la clase como un esquema relacional y los objetos se parecen a las instancias del esquema. Esto es verdad en un sentido limitado, puesto que una relación no puede contener una función, ni tampoco permite otras posibilidades que analizaremos a continuación.

Hay aquí también un ejemplo simple de envío de mensajes. Cuando se declara un objeto de una clase determinada, se le pasa un mensaje al constructor de la clase para que cree una plantilla (*template*) de la clase en memoria y le dé a ésta el nombre del objeto. (En este ejemplo tenemos tres mensajes.)

Clases derivadas y herencia simple

clase derivada. Una clase que hereda características de otra clase.

A la Figura 14.1 le hemos añadido dos tipos adicionales de objetos, ambos son especializaciones del objeto CLIENTE: CLIENTE-HUMANO (CH) y CLIENTE-INSTITUCIONAL (CI). Esto corresponde a las **clases derivadas** siguientes:

```
class CH: public CLIENTE {
          protected:
          char sexo;
          char fecha_nac[8];
          };

class CI: public CLIENTE {
          protected:
          int NÚM_EMPLEADOS;
          char TIPO_ORG;
          char PERS_CONTACTO[12];
          };
```

La notación "class CI" declara como es usual a una clase de nombre CI. La extensión ":public CLIENTE" declara a CH y CI como clases derivadas de la clase CLIENTE. La notación "public" significa que a la clase derivada se le permite el acceso a todos los miembros de la clase CLIENTE —excepto aquellos que hayan sido declarados "private"—. En los ejemplos anteriores, tanto CI como CH, pueden acceder a todos los miembros de CLIENTE. Por simplicidad hemos omitido ejemplos de funciones de clase y paso de mensajes, pero es fácil pensar en funciones que lean datos, escriban datos, busquen datos y ejecuten operaciones aritméticas y lógicas sobre los datos de la clase.

Podemos ahora declarar objetos de la clase derivada que tengan no sólo sus propios datos o funciones miembro, sino que hereden todos los que tienen sus clases base (padre). Por ejemplo, el fragmento C++

```
.
.
main ( ) {
        CI clientinst;
        CH clienthum;
}
```

crea un objeto (clientinst y clienthum) para cada una de las clases derivadas (CI y CH). Cuando se declare cada objeto de clase se pasa un mensaje para crear la plantilla para la clase base y luego la plantilla [4] para la clase derivada.

El objeto "clientinst" puede aceptar y manipular valores para sus propios miembros como son NÚM_EMPLEADOS, TIPO_ORG y PERS_CONTACTO y también hereda CLIENTE# de la clase base.

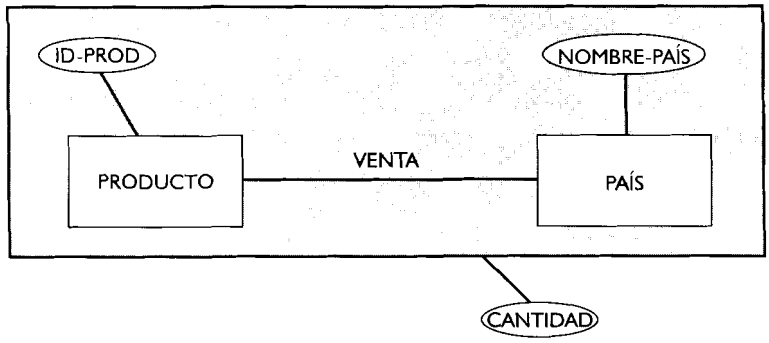
Agregación

objeto complejo. Corresponde a un agregado o interrelación de alto nivel (como se definió en el Capítulo 4).

¿Qué hay con la agregación? Considere el fragmento de un modelo orientado a objetos que se muestra en la Figura 14.2. Aquí tenemos una interrelación de agregación como un **objeto complejo**. Esto se representaría como sigue:

```
class   PRODUCT {
        protected:
        int ID_PROD;
        };
```

[4] C++ usa el término plantilla (*template*) para referirse a clases genéricas, éste no es exactamente el significado con el que se usa aquí, pero se ha respetado la terminología del texto original (N. del T.).




```
class   PAÍS {
        protected:
        char NOMBRE_PAÍS[15];
        };
class   VENTA {
        protected:
        int CANT;
        PAÍS p;
        PRODUCTO prod;
        };
```

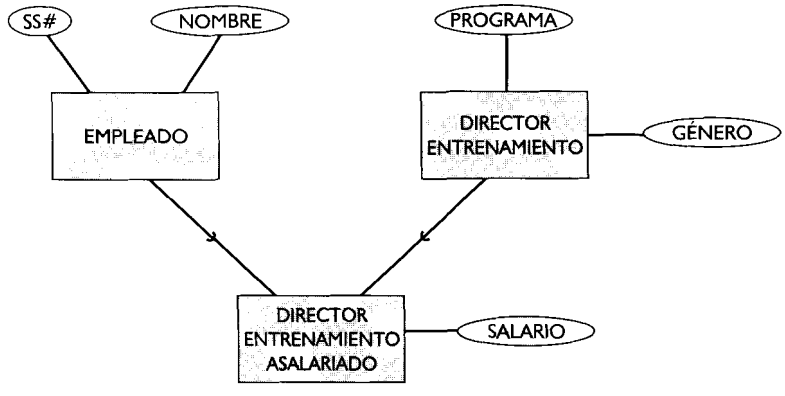
Vemos que PAÍS y PRODUCT se describen de la manera usual. Esas clases devienen en miembros de la clase VENTA. Cuando se declara un objeto VENTA en la forma siguiente:

```
VENTA esta_venta;
```

Se crea una instancia de la clase VENTA con el nombre esta_venta. El objeto esta_venta puede tener los datos de una venta en particular, por ejemplo, " Alemania" , "Jabón", " 25". Tenemos así toda la información agregada que se quería.

Un ejemplo extendido que demuestra la herencia múltiple

Nos referiremos a las Figuras 14.3 y 14.4. Observe que en el fragmento de modelo orientado a objetos de la Figura 14.3 tenemos tres conjuntos de objetos: EMPLEADO,



```
// Este programa demuestra la herencia múltiple

// Primero se define la clase EMPLEADO
class empleado
     {
          char   ss#[11];
          char   nombre[12];
          void   traeDato( );    // Esta es una función miembro para la entrada de datos
                 {
                 cout << "\ Introduzca el número de seguridad social del empleado: ";
                 cin >> ss#;
                 cout << "Introduzca el nombre del empleado: ";
                 cin >> nombre;
                 }
     };
// Ahora se define la clase DIRECTOR DE ENTRENAMIENTO
class dir_entrenamiento
     {
          char   genero;
          char   program[12];
          void   entDato( )
                 {
                 cout << " Introduzca el género de entrenamiento: ";
                 cin >> género;
                 cout << " Introduzca tipo de programa de entrenamiento: ";
                 cin >> programa;
                 }
     };
//Ahora definimos la clase derivada DIRECTOR DE ENTRENAMIENTO ASALARIADO
class asal_dir_entrenamiento: public empleado, public dir_entrenamiento
     {
          float  salario;
          void   entrada( )
          {
            empleado::traeDato( );
            dir_entrenamiento::entDato( );
            cout << " Entre el salario del entrenador: ";
            cin >> salario;
          }
     };
main( )
     {
     asal_dir_entrenamiento ade1;
     ade1.entrada( );
     }
```

DIR_ENTRENAMIENTO y ASALARIADO_DIR_ENTRENAMIENTO. EMPLEADO y DIR_ENTRENAMIENTO son clases bases corrientes, mientras que ASALARIADO_DIR_ENTRENAMIENTO es una especialización tanto de EMPLEADO como de DIR_ENTRENAMIENTO. Esto es, ASALARIADO_DIR_ENTRENAMIENTO hereda todos los atributos de EMPLEADO y DIR_ENTRENAMIENTO y a su vez añade el atributo SALARIO. Esto es consistente con nuestros primeros modelos del Capítulo 4.

Habiendo expuesto algunas representaciones modestas de OOP, vamos a ampliar la representación un poco para dar un cuadro más general. Consideremos ahora el programa C++ de la Figura 14.4, que convierte el fragmento de modelo de la Figura 14.3 a OOP.

Se han incluido varios comentarios en el programa (se denotan por "//"). Se aclaran y extienden estos comentarios. Primero observamos la creación de la clase llamada "empleado". Previamente se usó este nombre con letras mayúsculas conforme al modelo orientado a objetos —pero esto fue por facilidad y no es una exigencia—. Observe que la clase contiene ahora no sólo atributos del conjunto de objetos EMPLEADO, sino que contiene también una función simple. Esta función tiene nombre "traeDatos". Sin profundizar mucho en la sintaxis es fácil ver que esta función permite la entrada de datos.

Vemos una representación similar para el conjunto DIR_ENTRENAMIENTO. El conjunto ASALARIADO_DIR_ENTRENAMIENTO se declara como una clase en la forma usual (con la etiqueta "asalariado_dir_entrenamiento" como nombre de clase), pero también vemos que se ha añadido la extensión "public empleado, public dir_entrenamiento. Esta extensión significa que la clase "asalariado_dir_entrenamiento" hereda todos los atributos y funciones de la clase "empleado" y de la clase "dir_entrenamiento" (a menos que ellos se hayan declarado privados, lo que no es el caso aquí).

La clase "asalariado_dir_entrenamiento" también tiene sus propios atributos y funciones. El atributo que es único para esta clase es "salario". Observe que esta clase contiene una función "entrada". La palabra "void" significa que no se retorna ningún valor. Los "( )" significan que la función no recibe valores (parámetros). Observe que la función "entrada" utiliza funciones de ambas clases base.

"empleado::traeDatos" provoca la ejecuciou de la función traeDatos de la clase "empleado", lo que garantiza que se introduzca el número de seguridad social y el nombre del empleado. dir_entrenamiento::entraDatos" origina que se ejecute la función entraDatos de la clase "dir_entrenamiento" que obtiene el género del entrenamiento y la identificación del programa. Esto va seguido de instrucciones que obtienen el salario de dir_entrenamiento.

Ninguna de estas acciones ocurre hasta que no se invoquen a través de un objeto. Aquí vemos que se declara un objeto de la clase "asalariado_dir_entrenamiento". Este objeto tiene nombre "age1". Entonces "age1.entrada" provoca la ejecución de la función "entrada" de la clase "asalariado_dir_entrenamiento", de la cual age1 es un objeto. De modo que todos los datos que se introduzcan mediante esta función irán a parar al objeto "age1".

Aunque estos son ejemplos muy simples, sirven para demostrar la interrelación entre el modelado conceptual y su implementación en el software orientado a objetos. En particular se requieren las siguientes características en los SGBDOs:

1. **Capacidad de manejar objetos complejos.** El SGBDO debiera ser capaz de almacenar y manipular objetos de alto nivel —aquellos que están compuestos de objetos simples y sus interrelaciones.
2. **Capacidad de incorporar métodos que operen sobre los datos de un objeto.** Las funciones que se aplican a un objeto deben estar encapsuladas en la definición del objeto.
3. **Capacidad de implementar objetos como miembros de las clases.** La idea de una clase como un conjunto de objetos instancias es la base sobre la cual se formulan las consultas. En las bases de datos relacionales, una consulta se aplica a una relación o a una colección de relaciones. De manera similar, en las **bases de datos orientadas a objeto,** una consulta se aplica a una clase o a un conjunto de clases.
4. **Capacidad de representar jerarquías de clases y herencia.** Los SGBDOs permiten al usuario derivar una nueva clase a partir de una clase existente que hereda todos los atributos y métodos de una o más clases base (padres). Una clase puede tener una cantidad cualquiera de clases derivadas, aunque algunos SGBDOs permiten que una clase tenga sólo una clase base. En el primer caso, una clase hereda atributos y métodos de más de una clase base. Veremos esto en el ejemplo de herencia múltiple. En el segundo caso, una clase derivada hereda atributos y métodos de una sola clase base. También se mostrará un ejemplo de este caso.

sistema de base de datos orientado a objetos. Sistemas de bases de datos que pueden implementar modelos conceptuales directamente y que pueden representar complejidades que van más allá de las capacidades de los sistemas relacionales.

▼ Desarrollos de los sistemas de bases de datos orientados a objetos

En esta sección se resumen algunos de los SGBDOs más conocidos. Empezaremos con Genstone (Meier y Stone, 1987).

Gemstone

Gemstone está basado en el lenguaje de OOP Smalltak, con sólo unas pocas extensiones. Es un producto comercial que suministra aspectos de la OOP, tales como identidad de un objeto y encapsulación, junto con herencia simple y una interfaz externa como un conjunto de mensajes. Como sistemas de gestión de bases de datos, Gemstone proporciona control de concurrencia y recuperación, gestión de almacenamiento secundario y autorización. Gemstone tiene un lenguaje de consulta que permite formular expresiones sobre las variables de instancia de un objeto.

Gemstone soporta acceso concurrente y métodos para mantener la seguridad y la integridad de la base de datos.

Vbase

Vbase (Andrews y Harris, 1987) es también un SGDBO comercial. Vbase enfatiza algunas características de la OOP, como una estructura fuerte de tipos de datos y la utilización de los tipos de datos abstractos. Vbase usa un lenguaje de especificación, TDL, para definir objetos, nombrar las variables de la base de datos y para declarar restricciones. Vbase usa un lenguaje de implementación llamado COP.

Vbase permite que se creen objetos de un tipo (clase) para los que sus propiedades y operaciones se determinan por este tipo de creación. Vbase puede crear también objetos agregados y soporta las interrelaciones uno-uno, uno-muchos y muchos-muchos entre los objetos. Vbase soporta la mayoría de las funcionalidades que se esperan de un SGBD. Los objetos pueden compartirse entre múltiples procesos concurrentemente, también suministra facilidades de seguridad y recuperación junto con control de acceso simple.

Orion

Orion es un prototipo de SGBDO que está siendo desarrollado en el MCC (Banerjee y otros, 1987). Al igual que Gemstone, Orion se deriva del lenguaje de OOP Smalltalk, pero posee capacidades de herencia múltiple. Incluye recursos adicionales para bases de datos, como la posibilidad de hacer una amplia variedad de cambios en el esquema de la base de datos —tales como la definición de clases y la definición de una jerarquía de clases, así como la capacidad de crear y manipular objetos complejos.

Orion está diseñado para soportar de modo natural necesidades de orientación a objetos tales como las de los sistemas CAD/CAM o la de los sistemas basados en el conocimiento.

PDM

PDM (Manola y Dayal, 1986) está basado en el modelo funcional de datos DAPLEX. Como consecuencia, el álgebra de PDM está muy cercana al álgebra relacional, con algunas diferencias importantes. La más importante entre éstas es la capacidad de aplicarle funciones multiargumentos a los objetos. PDM incluye también un operador APPLY_APPEND que permite realizar la composición de funciones. En PDM, al igual que en DAPLEX, todo es función.

IRIS

IRIS (Fishman y otros) está siendo desarrollado por Hewlett Packard. Tiene una semejanza cercana a los sistemas relacionales. IRIS utiliza un administrador de almacenamiento similar al que usa System R y soporta un lenguaje de consulta basado en SQL (OSQL). El modelo de IRIS permite la representación de objetos y de colecciones de objetos, así como de operaciones.

El SGBD consiste en un procesador de consultas que implementa el modelo de datos orientado a objetos y que soporta abstracciones estructurales de alto nivel, tales como la clasificación, la generalización y la agregación, así como abstracciones de comportamiento. Las interfaces con IRIS incluyen una extensión a objetos de SQL.

O_2

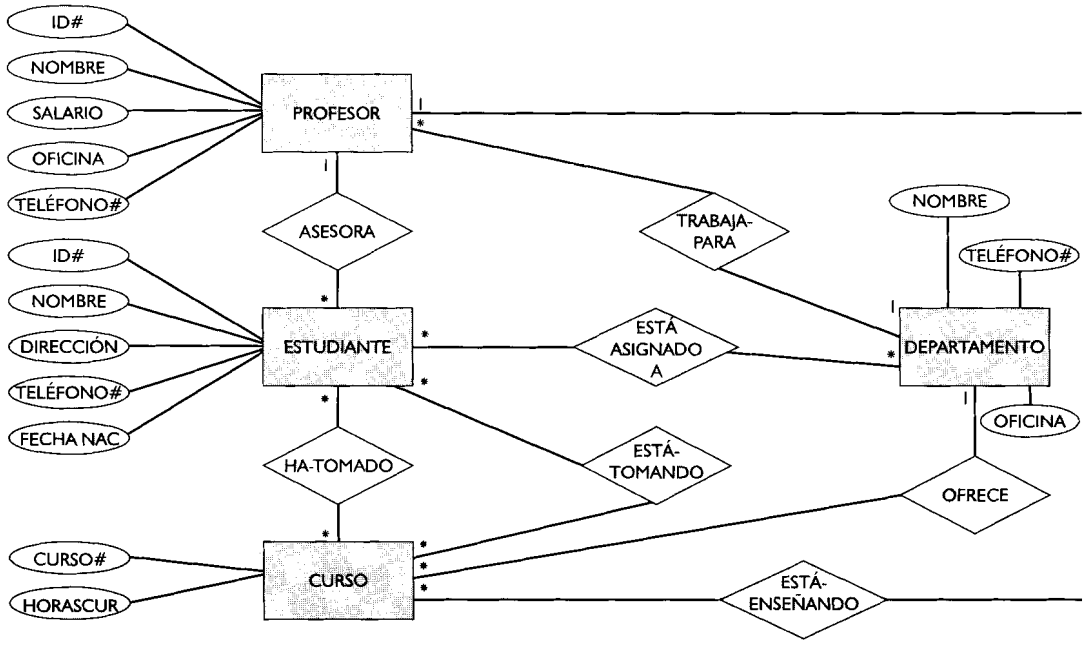
O_2 (Leehse y otros, 1988) soporta tipos de objetos, encapsulación y herencia. O_2 soporta también múltiples lenguajes, tales como BASIC, C y LISP. Proporciona recursos para la encapsulación, la herencia y la sobrecarga de operadores. En contraste con Gemstone, en el que todo es un objeto, O_2 no define las clases y los métodos como objetos.

▼ Un lenguaje genérico de base de datos de objetos

En esta sección se presenta un lenguaje genérico de base de datos de objetos llamado TEXT-QUERY. Como el nombre sugiere, es un lenguaje de consulta textual que opera sobre bases de datos de objetos. Este lenguaje no está realmente implementado, pero demuestra muchas de las características comunes en los lenguajes de bases de datos de objetos.

El modelo de datos del Lerner College

En la Figura 14.5 se muestra una parte del modelo de datos del Lerner College. Este modelo contiene cuatro conjuntos de objetos (PROFESOR, ESTUDIANTE, CURSO, DEPAR-



TAMENTO) y siete interrelaciones. Cada uno de los conjuntos de objetos tiene nn número de atributos. Para hacer el ejemplo más concreto se han tomado algnnos ejemplos de datos del modelo de datos del Lerner College en la Fignra 14.6. Esta figura muestra valores simples de atributos para instancias de cada uno de los conjuntos de objetos y uno de los conjuntos interrelaciones (HA-TOMADO). La Figura 14.7 muestra cómo las instancias de los conjuntos de objetos se relacionan con otras vías de las interrelaciones dadas.

Antes de seguir adelante, pudiera ser útil revisar la distinción entre una instancia de objeto y los valores de sus atributos. De modo que "John" es el nombre de una persona, pero la persona a la que llamamos "John" no es la misma cosa que su nombre. Como un ejemplo adicional, podemos identificar una persona por su número de seguridad social, pero la persona no es un número de seguridad social. Por lo tanto, la Figura 14.5 representa conjunto de objetos separadamente de sus atributos. La ventaja de esto es que podemos cambiar los valores de los atributos, añadir atributos y borrar atributos sin preocuparnos de perder ninguna de las instancias de objetos. También podemos manipular

PROFESOR

ID#	NOMBRE	SALARIO	OFICINA	TELÉFONO#
821	Adams	38000	281 CB	4822
911	Clyde	27000	48 TMB	3085
237	Brown	38000	521 MCK	7324
113	Parker	33000	492 CB	6122
544	Hinman	42000	213 TMB	4188
145	Stevens	45000	312 CB	1203

DEPARTAMENTO

NOMBRE	TELÉFONO#	OFICINA#
MATEMÁTICAS	8111	411 TMB
INGLÉS	4980	512 MCK
HISTORIA	5233	313 CB

ESTUDIANTE

ID#	NOMBRE	DIRECCIÓN	TELÉFONO#	FECHA-NAC
3825	Mary	214 HH	2112	3/12
4913	John	3ª DG	3114	14/9
6255	Ann	4117 RCK	5311	5/3
4118	Kelly	311 ST	6622	24/4
3223	Roger	4214 RCK	1383	17/7

CURSO

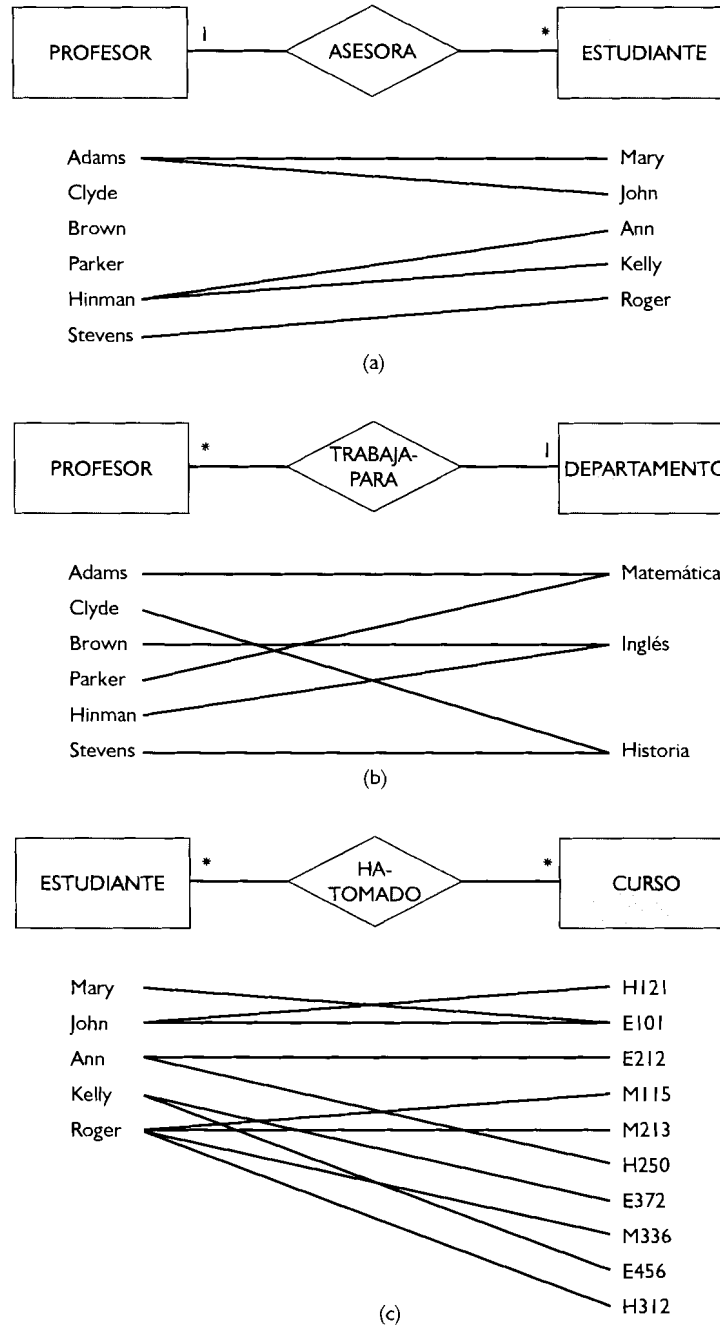
CURSO#	HORASCR	CURSO#	HORASCR
H121	3	H250	2
I101	3	E372	3
I212	2	M336	3
M115	4	E456	3
M213	3	H312	3

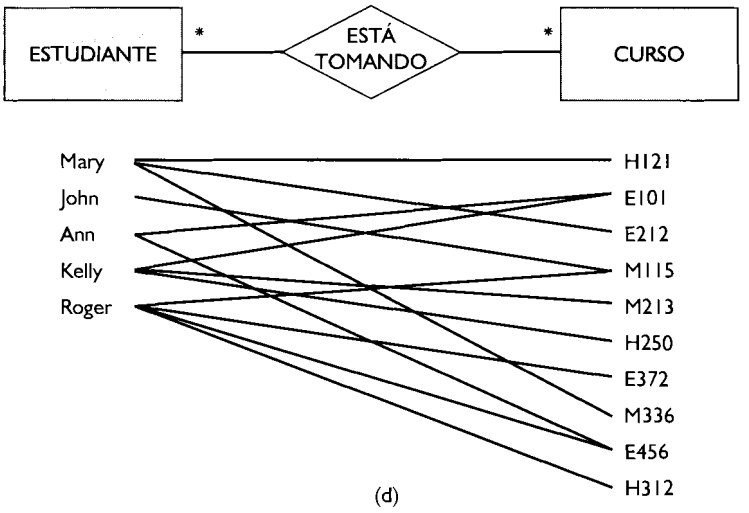
HA-TOMADO

NOMBRE(ESTUD)	CURSO#	NOMBRE (ESTUD)	CURSO
Mary	I101	Kelly	I456
John	H121	Roger	M115
John	I101	Roger	M213
Ann	I212	Roger	M336
Ann	H250	Roger	H312
Kelly	I372		

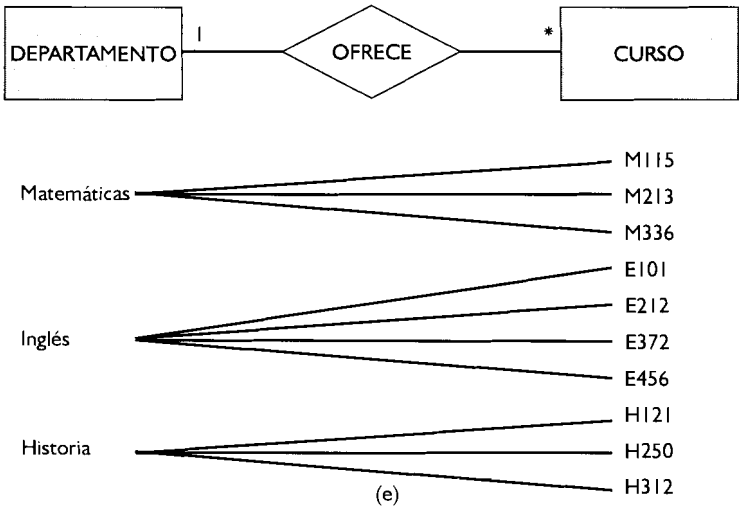
instancias de objetos sin preocuparnos sobre atributos específicos o sobre los valores de los atributos.

La Figura 14.8 puede ayudarnos a comprender la separación entre una instancia de objeto de un valor objeto. La Figura 14.8 (a) muestra una representación típica del conjunto de objetos ESTUDIANTE y del atributo NOMBRE, mientras que la Figura 14.8(b)

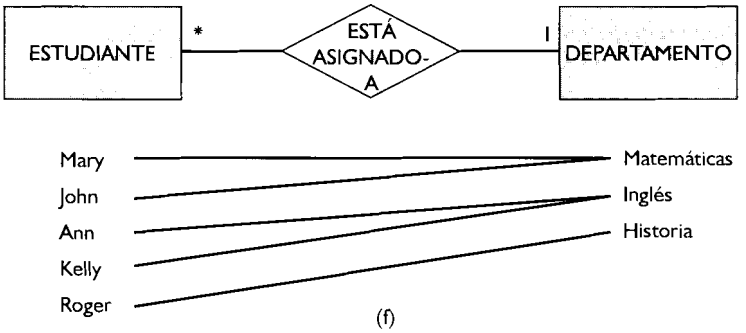



ESTUDIANTE
*
ESTÁ TOMANDO
*
CURSO
Mary
John
Ann
Kelly
Roger
H121
E101
E212
M115
M213
H250
E372
M336
E456
H312

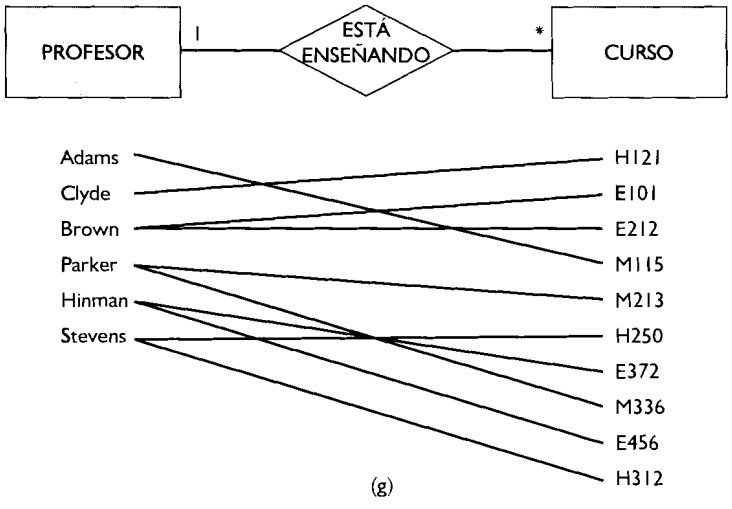
(d)


DEPARTAMENTO
1
OFRECE
*
CURSO
Matemáticas
Inglés
Historia
M115
M213
M336
E101
E212
E372
E456
H121
H250
H312

(e)


ESTUDIANTE
*
ESTÁ ASIGNADO-A
1
DEPARTAMENTO
Mary
John
Ann
Kelly
Roger
Matemáticas
Inglés
Historia

(f)

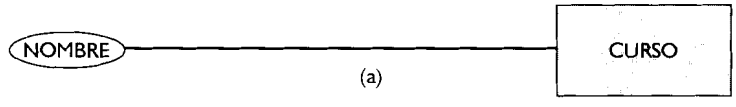


(g)

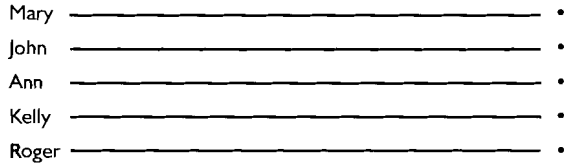
muestra las instancias reales de objetos y los valores de los atributos. En este caso, el conjunto de objetos ESTUDIANTE contiene cinco instancias, o estudiantes, representados por los cinco puntos que se muestran en la caja ESTUDIANTE. Tenemos un valor —un nombre específico— para el atributo NOMBRE para cada estudiante. Este valor se indica por una línea que se dibuja desde el valor del atributo —el nombre— hasta el punto que representa al estudiante. De modo que tenemos una línea desde "Mary" hasta el punto que representa al estudiante de nombre "Mary" en el conjunto de objetos ESTUDIANTE.

Los atributos son importantes porque identifican y ayudan a distinguir entre las diferentes instancias de objetos. De hecho, una *clave* es un atributo o un conjunto de atributos que identifica únicamente cada instancia del objeto. Sin embargo, para algunos propósitos no es necesario usar valores de atributos para distinguir entre instancias de objetos. Por tanto, tratar una instancia de un objeto separadamente de los valores de sus atributos facilitará ciertos tipos de manipulación de datos. Esto se ilustrará cuando posteriormente en el capítulo se realicen varias consultas.

Observe que estamos usando atributos —NOMBRE y CURSO#— para identificar instancias de objetos en nuestro ejemplo de base de datos. La Figura 14.7 a-g hace esto más que mostrar las instancias como puntos que están conectados con los valores de los atributos. Esto se realiza sólo para simplificar los diagramas.



(a)



(b)

▼ Formular consultas con TextQuery

Durante sus entrevistas con el personal en Administración, Registro y Admisión y dondequiera en el campus del colegio, Roberta y Juan han coleccionado una larga lista de preguntas que los usuarios potenciales de la base de datos quisieran ver respondidas. Los siguientes son algunos ejemplos representativos:

¿Quiénes son los profesores que ganan por encima de los $40.000? ¿Qué profesores trabajan para el departamento de Inglés? ¿Cómo se llaman los profesores con los que toma clases Kelly? ¿Qué estudiantes han tomado todos los cursos ofrecidos por el departamentos de Matemáticas? ¿Qué estudiantes han tomado todos los cursos ofrecidos por el departamento básico? ¿Qué estudiantes están repitiendo? ¿Qué estudiantes han pasado, o están pasando, un curso de historia?

Las respuestas a estas pregnntas deberían estar contenidas en los conjuntos de objetos e interrelaciones de nuestra base de datos. Si pudiera usarse un computador para acceder a la base de datos, podrían responderse a estas preguntas. La clave es formular esas consultas en un lenguaje orientado a objetos, tal como TextQuery, que podría traducirse en operaciones que el computador pueda ejecutar, permitiendo a la computadora identificar y acceder los datos apropiados.

El formato básico de las soluciones de consulta en TextQuery

Empezamos la descripción de TextQuery con un ejemplo.

```
¿Cuáles son los nombres y salarios de los profesores que ganan por en-
cima de los $40.000?
```

En la Figura 14.5 se puede ver que los datos necesarios para responder a esta consulta están contenidos en el conjunto de objetos PROFESOR y sus atributos. La solución a esta consulta puede formularse en TextQuery en la forma siguiente:

```
{ (nombre, salario) :   nombre ES-NOMBRE-DE profesor AND
                        salario ES-SALARIO-DE profesor AND
                        salario > 40000 }
```

Esta solución en TextQuery define un conjunto cuya definición se da entre claves. La definición debe leerse como: "El conjunto de los pares de nombres y salarios tales que el nombre es el nombre de algún miembro del profesorado, el salario es el salario del mismo miembro del profesorado y el salario es mayor que 40.000."

Conjunto solución. Un conjunto de valores de datos de la base de datos que satisface las condiciones de una consulta.

Si esta solución TextQuery se aplica a la base de datos nos da un **conjunto solución** —un conjunto de valores de datos de la base de datos consistente de pares de nombres y salarios que satisface las condiciones de que el nombre y el salario del par se aplican a un mismo profesor y que el salario es mayor que 40.000—. Usando los datos de la Figura 14.6, el conjunto solución para esta consulta es un conjunto que consiste en los dos pares:

```
{ (HINMAN, 42000), (STEPHENS, 45000) }
```

Tomemos aparte la solución TextQuery y examinémosla por partes.

variable. Un nombre simbólico que representa una instancia no especificada en un conjunto de objetos.

Términos. En la solución TextQuery, las palabras *nombre*, *salario* y *profesor* representan instancias de los conjuntos de objetos NOMBRE, SALARIO y PROFESOR, respectivamente. Esto es, estas palabras son **variables** que pueden tomar cualquiera de los valores

en estos tres conjuntos. ES-NOMBRE-DE y ES-SALARIO-DE son nombres de interrelaciones correspondientes a los atributos NOMBRE y SALARIO que relacionan a los miembros del profesorado con su nombre y salario. De modo que si la variable *profesor* representa al profesor cuyo nombre es BROWN, se tiene entonces que

```
nombre = BROWN, y
salario = 38000.
```

Partes de definición de conjuntos. Los dos puntos (:) separan las dos partes principales de la definición de conjunto —la lista de resultado y la expresión de calificación (Cood, 1971)—. La **lista resultado** (*target list*), una lista parentizada de variables, da el formato deseado de un miembro típico del conjunto de soluciones —en este caso, un par de elementos consistente en un nombre y un salario—. La **expresión de calificación** da una condición que se refiere a la lista resultado y debe ser verdadero para los elementos en el conjunto solución.

lista resultado. Una lista parametrizada de variables que representa el formato deseado de un miembro típico de un conjunto de solución de una consulta.

expresión de calificación. Una condición verdadera o falsa que se refiere a la lista resultado y que debe cumplirse para los elementos en el conjunto solución.

La lista de resultado del ejemplo es

```
(nombre, salario)
```

y la expresión de calificación consta de tres subexpresiones:

```
nombre ES-NOMBRE-DE profesor
salario ES-SALARIO-DE profesor
salario > 40000
```

que están conectadas por AND lógicos.

Aplicando la solución TextQuery a la base de datos se genera el conjunto de solución. Esto se hace permitiendo que la lista resultado vaya asumiendo todas las posibles combinaciones de valores nombre y salario. Para cada combinación, se evalúa la expresión de calificación. Si la expresión es verdadera, entonces el par nombre-salario se pone en el conjunto solución. Si no es verdadera, se descarta esa combinación nombre-salario y se examina la próxima combinación nombre-salario.

Supongamos por ejemplo:

```
nombre = ADAMS
salario = 42.000
```

La expresión de calificación será verdadera si y sólo si existe alguna instancia de PROFESOR cuyo nombre es ADAMS, que tiene un salario de 42.000, y cuyo salario es mayor que 40.000. En este caso el salario es mayor que 40.000, pero la expresión de calificación como un todo es falsa porque no hay ningún profesor de nombre Adams que tenga un salario de 42.000 (revise la Fignra 14.6). Por lo tanto, la combinación (ADAMS, 42000) no se pone en el conjnnto solución.

Intentemos con la combinación:

```
nombre = ADAMS
salario = 38.000
```

Hay un miembro de profesor de nombre ADAMS que tiene nn salario de 38.000, de modo que las primeras dos subexpresiones son verdaderas. Pero la última subexpresión es falsa, ya que el salario *no* es mayor que 40.000. Por lo tanto, la expresión de calificación no es verdadera para la combinación (ADAMS, 38000) y esta solución tampoco se pone en el conjnnto solución.

Es fácil ver que (INMAN, 42000) y (STEPHENS, 45000) son las únicas dos combinaciones que satisfacen la expresión de calificación. Por tanto, estos dos pares constituyen el conjunto solución completo de esta consulta.

Esta consulta y su solución ilustran varias reglas del lenguaje TextQuery. Antes de continuar vamos a ver estas reglas y considerar algunas mejoras que se pueden usar en estas y en otras consultas.

1. Las variables en TextQuery que representan instancias de un conjunto de objetos se escriben en minúscula. El nombre en minúscula de un conjunto de objetos denota a una instancia de dicho conjunto de objetos.

2. Las variables que representan instancias de un conjunto de objetos también pueden definirse indicando que la variable está (IN) en el conjunto de objetos, tal como se muestra en los ejemplos siguientes:

```
p IN PROFESOR
profesor2 IN PROFESOR
profesor´ IN PROFESOR
```

Estas instrucciones definen a *p*, *profesor2* y *profesor´* como variables, cada una representando una instaucia en el conjunto de objetos PROFESOR.

3. Los nombres de atributos de un conjunto de objetos se pueden transformar a nombres de interrelaciones añadiéndole el prefijo "ES-" y el sufijo "-DE" al nombre del atributo [5]. De modo que

```
ES-NOMBRE-DE y ES-SALARIO-DE
```

son nombres de interrelaciones creados para los atributos NOMBRE y SALARIO, respectivamente.

4. Una solución TextQuery siempre es una definición de conjunto, consistente en una expresión —la lista resultado— que representa a un elemento típico del conjunto solución, seguida de dos puntos, seguida de una condición —la expresión de calificación— que califica precisamente los elementos a incluir en el conjunto solución. La definición se encierra entre claves {} para indicar que estamos definiendo un *conjunto* —en particular, el conjunto de todos los elementos cuyo formato es la lista resultado y que satisfacen la condición establecida por la expresión de calificación.

5. Dentro de la definición de un conjunto, una variable retiene su significado entre una ocurrencia y la siguiente. Así pues, en nuestro ejemplo, *salario* está definido en la lista resultado y ocurre en la expresión de calificación. *Esto significa lo mismo cada vez que ocurra*. Si durante el cálculo del conjunto solución para la consulta, *salario* asume el valor 38.000 en la lista resultado, entonces tiene el mismo valor cada vez que aparezca en la expresión de calificación.

6. Si una variable no aparece en la lista resultado, pero aparece en la expresión de calificación, se dice que la variable está *cuantificada existencialmente*. Esto significa que *existe* algún valor en el conjunto de objetos para la variable. Por ejemplo, en la expresión

```
salario ES-SALARIO-DE profesor
```

profesor está existencialmente cuantificada. La expresión puede interpretarse como: " Existe un miembro del profesorado tal que *salario* es el salario de dicho miembro."

[5] Para mayor comprensión se han puesto en español el prefijo ES (IS) y el sufijo DE (OF) (N. del T.).

Definir las direcciones de las interrelaciones

El primer ejemplo fue relativamente simple, puesto que relacionaba sólo a un conjunto de objetos y sus atributos. Sin embargo, la solución a otras consultas puede involucrar conjuntos de objetos que están enlazados por una serie de interrelaciones. De modo que necesitamos echar una mirada más de cerca a estas interrelaciones y a qué significan intuitivamente. En la medida en que hagamos esto iremos desarrollando los medios para resolver consultas más complejas. Este concepto se explorará más adelante en esta sección. Seremos más precisos en la próxima sección, donde aplicaremos estas ideas para formular consultas en TextQuery.

Cuando se definieron los objetos y las interrelaciones en el Capítulo 4, se compararon los objetos con nombres y las interrelaciones con verbos. Esto sugiere que un lenguaje textual como TextQuery se asemeje al lenguaje natural en cierto sentido. Esto se puede ver si analizamos la Figura 14.5. Los nombres de cada una de las interrelaciones sugieren una *dirección* para leer la interrelación. De modo que diríamos

```
profesor ASESORA estudiante
```

no es lo mismo

```
estudiante ASESORA profesor
```

El nombre de la interrelación sugiere que la interrelación se lee de PROFESOR a ESTUDIANTE. Similarmente tenemos

```
profesor TRABAJA-PARA departamento
profesor ESTÁ-ENSEÑANDO curso
estudiante ESTÁ-MATRICULADO-EN departamento
estudiante HA-PASADO curso
estudiante ESTÁ-PASANDO curso
departamento OFRECE curso
```

ninguna de las cuales tendría sentido en el orden inverso. Cada una de las relaciones es una oración en lenguaje natural en el formato

```
SUJETO VERBO PREDICADO
```

que está en voz activa.

Cambiando ahora el nombre de la interrelación y usando la voz pasiva podemos invertir sn dirección en la mayoría de las instrucciones. De modo que tenemos

```
estudiante ES-ASESORADO-POR profesor
curso ES-ENSEÑADO-POR profesor
curso HA-SIDO-PASADO por estudiante
curso ESTÁ-SIENDO-PASADO-POR estudiante
curso ES-OFRECIDO-POR departamento
```

como instrucciones que muestran la *dirección inversa* para la lectura de las interrelaciones. Se puede obtener también la dirección contraria de las otras dos interrelaciones, renombrando apropiadamente el verbo de la forma siguiente:

```
departamento EMPLEA profesor
(EMPLEA sería el inverso de TRABAJA-PARA)
```

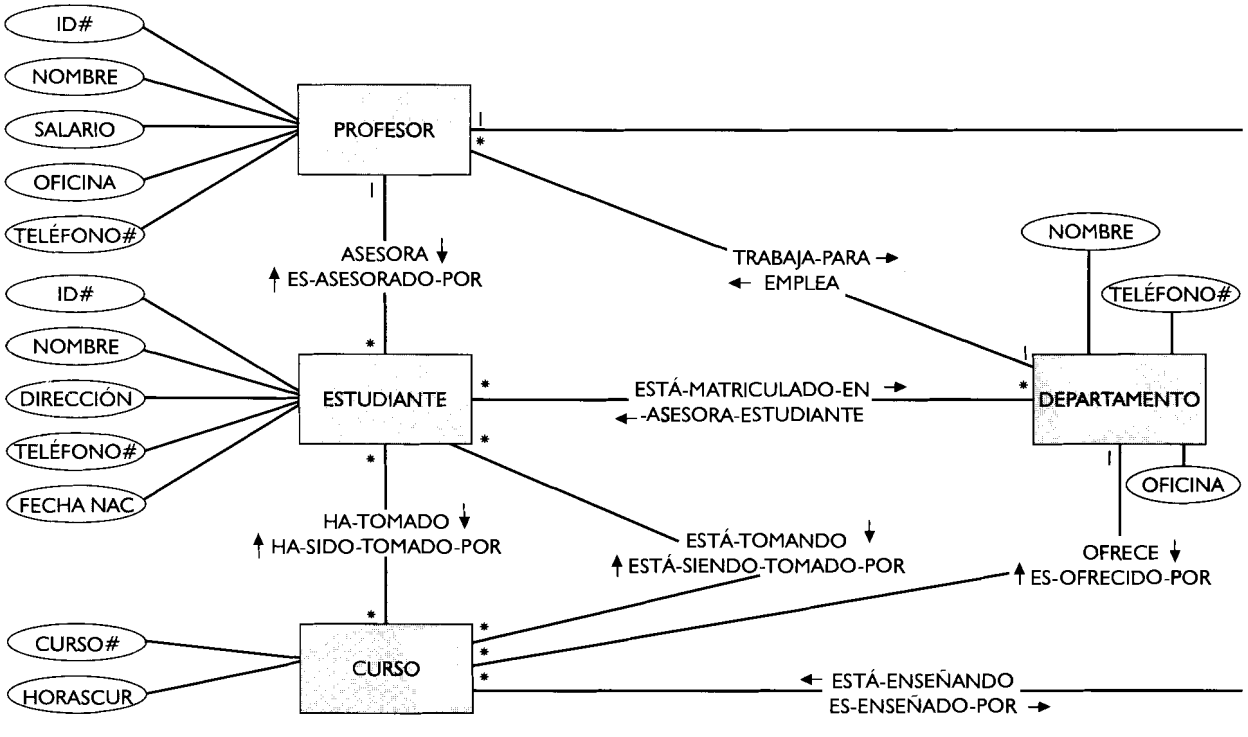
```
departamento TIENE-MATRICULADO-A estudiante
(TIENE-MATRICULADO-A es el reverso de ESTÁ-MATRICULADO-EN)
```

El nombre de la última relación (TIENE-MATRICULADO-A) muestra que siempre podemos encontrar un nombre que invierta la dirección de una interrelación, aun cuando el nuevo nombre pudiera ser raro o no conveniente.

Por tanto, el nombre que se escoja para una relación sugiere una dirección intuitiva que permitirá que se vea como una sentencia en lenguaje natural en el formato SUJETO-VERBO-OBJETO. También podemos crear un nombre sustituto para cada relación que haga posible su lectura en la dirección opuesta. Se puede diagramar esta idea actualizando la Figura 14.5 para mostrar la dirección de los nombres de las relaciones. La flecha que precede o sigue a cada nombre de relación en la Figura 14.9 indica esta dirección.

Es importante la dirección de los nombres de las relaciones para resolver consultas que requieran la navegación a través de una serie de conjuntos de objetos y sus interrelaciones. Dependiendo de las necesidades de la consulta, a veces vamos a navegar en una dirección y otras veces lo haremos en la dirección opuesta. Por tanto, es muy importante la capacidad de ir en ambas direcciones. Puesto que prácticamente cualquier consulta puede requerir este tipo de navegación, estamos estableciendo aquí el esquema general para la solución de consultas.

Se necesita hacer lo mismo para atributos que lo que hacemos para las relaciones. Recuérdese que podemos interpretar los atributos como interrelaciones. Como vimos antes, podemos crear frases de nombres y verbos para ellos incrustando el nombre del atributo entre ES y DE. Podemos invertir la dirección del nombre de la relación atributo poniéndole el prefijo TIENE al nombre del atributo:



```
profesor TIENE-NOMBRE nombre
profesor TIENE-SALARIO salario
profesor TIENE-OFICINA oficina
```

Finalmente necesitamos derivar nombres para las interrelaciones de especialización. Supongamos que tenemos un conjunto llamado CURSO-MAT que consiste en todos los cursos ofrecidos por el departamento de matemáticas. Luego CURSO-MAT es un subconjunto de CURSO. Se nombrará a esta especialización de la interrelación como ES-UN (IS-A), ¡este nombre vale en ambas direcciones! Esto es,

```
curso-mat ES-UN curso y
curso ES-UN curso-mat
```

pueden ser ambas verdaderas. La primera es verdadera para todas las instancias de CURSO-MAT, mientras que la segunda es verdadera sólo para algunas instancias de CURSO.

Navegar sobre atributos e interrelaciones

En la sección previa se vio que los nombres que se usaron para objetos, atributos e interrelaciones pueden acondicionarse para hacer más legible su interpretación como sentencias del lenguaje natural. En esta sección se mostrará cómo estas ideas nos pueden llevar a resolver consultas más complejas en el lenguaje de consulta TextQuery.

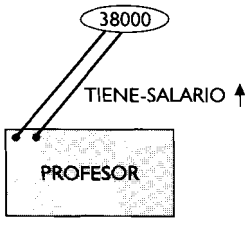
```
¿Cuáles son los nombres de los profesores que ganan $38.000?
```

Esta consulta, al igual que la primera consulta, requiere de datos que sólo se encuentran en el conjunto de objetos PROFESOR y dos de sus atributos. Sin embargo, en este caso, el salario sólo figura en la expresión de calificación, no en la lista resultado. La solución a esta consulta ilustra el principio de navegar a través de interrelaciones y conjuntos de objetos:

```
{ nombre :      nombre ES-NOMBRE-DE profesor AND
                profesor TIENE-SALARIO 38000 }
```

Vamos analizar la primera solución empezando desde atrás y trabajando hacia adelante:

```
profesor QUIEN TIENE-SALARIO 38000
```



Esta es una sentencia sobre un profesor que puede evaluarse como verdadera o falsa.

Es verdadera sólo para aquellos que pertenecen al conjunto de objetos PROFESOR y que tienen un atributo SALARIO de 38.000. Esos miembros del profesorado constituyen un subconjunto de PROFESOR (Figura 14.10). Vemos que hay dos profesores que ganan $38.000.

Para explicar el resto de la solución de la consulta reescribamos la expresión de calificación con paréntesis:

```
nombre ES-NOMBRE-DE (profesor QUIEN TIENE-SALARIO 38000)
```

Esta sentencia es verdadera para un conjunto de nombres: {ADAMS, BROWN}. Los nombres en el conjunto serán aquellos que correspondan, bajo la interrelación ES-NOM-

BRE-DE, con las instancias de PROFESOR que ganen $38.000 (la definición en la última mitad de la consulta). Esto se muestra en la Figura 14.11.

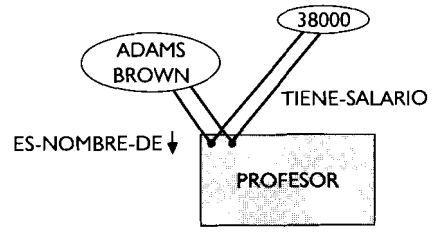
Note cuán estrechamente el formato de la solución en TextQuery se corresponde con el formato de la consulta original.

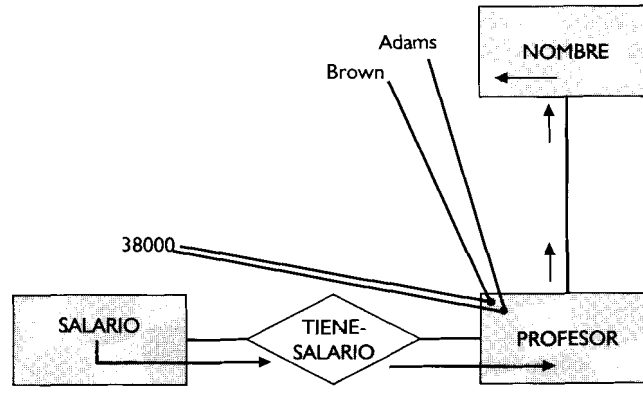
```
Consulta Usuario: ¿Cuáles son los nombres de profesores que ganan $38.000?

Solución en TextQuery: nombre ES-NOMBRE-DE profesor QUIEN TIENE-SALA-
RIO 38000
```

Puesto que esto sigue el formato de las consultas en lenguaje natural, se dice que tales soluciones están en *formato interrogativo*. Sin embargo, usando un modelo de datos para visualizar esta solución trabajamos *hacia atrás* (*backward*). Esto se puede ver en la Figura 14.12. Comenzamos con el atributo SALARIO y encontramos 38.000. Luego vamos atrás hacia PROFESOR para encontrar aquellas instancias de PROFESOR que "TIENE-SALARIO" de 38.000. Después continuamos yendo atrás un paso más hacia NOMBRE, usando ES-NOMBRE-DE. Si se pusiera junto a la solución TextQuery después de esta visualización, se puede rehacer el trazo de nuestros pasos que van de NOMBRE a través de ES-NOMBRE-DE hacia PROFESOR y luego a través de TIENE-SALARIO hacia un SALARIO de 38.000. Esta vuelta atrás (*backtracking*) sugiere que puede haber un formato alternativo que refleje más de cerca la solución que hemos visualizado en el diagrama. Podemos formular una solución en este formato alternativo como sigue:

```
{ nombre : 38000 ES-SALARIO-DE profesor QUIEN TIENE-NOMBRE nombre }
```





Vemos que el orden de la expresión de calificación aquí es opuesto al del formato interrogativo. Debido a que este segundo formato está en una forma más declarativa se le llama *formato declarativo*. Están ambos formatos permitidos. Por lo tanto, en el desarrollo de soluciones en TextQuery a las consultas de los usuarios debe usar el formato que encuentre más natural.

En ambos casos hemos simplificado la lista resultado a justamente un nombre de variable, mientras que la expresión de calificación hemos hecho recorridos a través de los conjuntos de objetos y sus atributos vía las interrelaciones que los enlazan. Los pronombres relativos tales como QUIÉN y QUÉ se usan en estas expresiones para aumentar la legibilidad. Estas son *palabras neutras* en el sentido de que están presentes en la expresión, pero no tienen efecto en su significado.

Veamos ahora dos ejemplos más complejos que nos brindan mayor ilustración del concepto de navegación sobre las interrelaciones.

Consulta usuario: ¿Qué profesores trabajan para el departamento de Inglés?

```
Solución en TextQuery: { profesor : profesor TRABAJA-PARA departamento
                                    QUE TIENE-NOMBRE 'Inglés' }
```

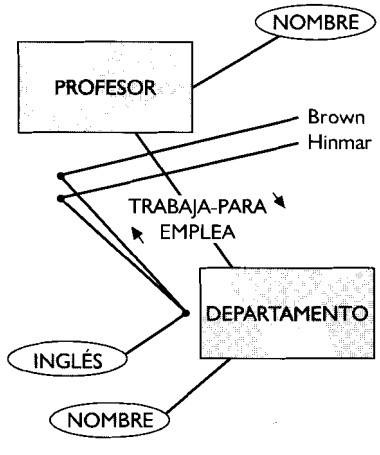
Aunque esta solución responde, la consulta es inadecuada porque el conjunto de miembros del profesorado que se ha creado no se puede desplegar en un dispositivo de salida. Si queremos producir una solución humanamente legible, debemos crear un conjunto de valores de atributos que sea legible. Por lo tanto, cambiamos la solución por:

```
{ nombre : nombre ES-NOMBRE-DE profesor QUIEN TRABAJA-PARA departamento
                                    QUE TIENE-NOMBRE 'Inglés' }
```

En un formato declarativo, la solución es:

```
{ nombre : 'INGLES' ES-NOMBRE-DE departamento QUE EMPLEA
                    profesor QUE TIENE-NOMBRE nombre }
```

En la Figura 14.13 se muestra el conjunto de solución para esta consulta.



Consulta de usuario: ¿Cuáles son los nombres de profesores con los que está tomando clases Kelly?

En formato interrogativo:

```
{ nombre  :   nombre ES-NOMBRE-DE profesor QUE-ESTÁ-ENSEÑANDO curso
              QUE ESTÁ-SIENDO-TOMADO-POR estudiante
              QUE TIENE-NOMBRE 'Kelly' }
```

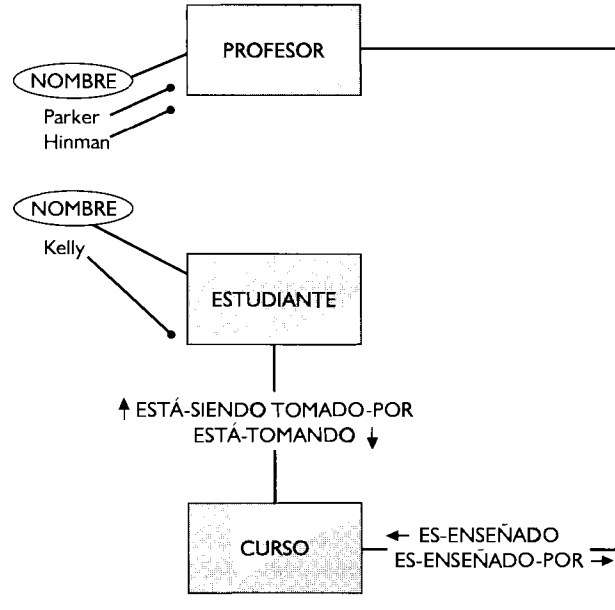
En formato declarativo

```
{ nombre  :   'Kelly' ES-NOMBRE-DE estudiante QUE ESTÁ-TOMANDO curso
              QUE ESTÁ-SIENDO-IMPARTIDO-POR profesor
              QUE TIENE-NOMBRE nombre }
```

En la Figura 14.14 se muestra el conjunto solución.

La Figura 14.14 *no* muestra las instancias de los cursos que Kelly está tomando, pero de los ejemplos anteriores debería ser fácil visualizar qué está pasando aquí. El nombre 'Kelly' está conectado a la instancia de estudiante que tiene ese nombre. Esta instancia de estudiante está a su vez conectada con el conjunto de cursos que Kelly está tomando. Cada uno de estos cursos está conectado con el profesor que está enseñando el curso y el profesor está conectado con su nombre.

Incidentalmente esta consulta nos da un ejemplo de utilización de un conjunto de objetos abstracto (CURSO) sin usar ninguno de sus atributos. La solución a la consulta no es identificar, por ejemplo, el número preciso de cursos que Kelly está tomando con vistas a identificar con qué profesores está pasando cursos Kelly. Es suficiente usar las instancias de cursos en su forma abstracta (sin atributos).



Asignar nombres a los conjuntos derivados

Los conjuntos solución resultantes de las consultas que han sido ilustradas fueron derivados del uso de TextQuery. A menudo tales conjuntos son útiles por sí mismos. Podríamos, por ejemplo, encontrar conveniente o más rápido usar estos conjuntos derivados en otras consultas TextQuery sin tener que derivarlos de nuevo. Podemos hacer esto si simplemente asignamos un nombre al conjunto derivado mediante el cual podemos referirnos a él más adelante. Por ejemplo, si queremos identificar el conjunto de todos los cursos de matemáticas, podemos asignarle el nombre CURSO-MAT al conjunto apropiado tal y como sigue:

```
CURSO-MAT := { curso : curso ES-OFRECIDO-POR departamento
                      QUE TIENE-NOMBRE 'MAT' }
```

Este ejemplo ilustra que los nombres se asignan poniendo primero el nombre del conjunto, seguido de ":=" y luego de la definición TextQuery del conjunto. El símbolo ":=" puede leerse como " es el nombre que se le asigna a".

Expresiones calificadas de comparación de conjuntos

Algunas consultas se pueden resolver utilizando expresiones calificadas que involucran la comparación de conjuntos. Considere la consulta:

Consulta usuario: ¿Qué estudiantes han tomado todos los cursos que se han ofrecido por el departamento de matemáticas?

Esta consulta se resuelve generando el conjunto de cursos ofrecidos por el departamento de matemáticas y luego, para cada estudiante, comparando ese conjunto con el conjunto de cursos que el estudiante ha tomado. Si el conjunto de todos los cursos de matemáticas *está contenido* en el conjunto de los cursos del estudiante, entonces es que el estudiante ha tomado todos los cursos de matemáticas y su nombre se pondrá en el conjunto solución de la consulta. Nuestra solución TextQuery consta de tres pasos y utiliza la facilidad de asignar nombres a los conjuntos derivados:

```
Solución TextQuery:

   CURSO-MAT := { curso : curso ES-OFRECIDO POR departamento
                          QUE TIENE-NOMBRE 'MAT' }

   ESTUDIANTE-DE-TODAS-LAS-MAT :=

   { estudiante : CURSO-MAT IS-CONTAINED-IN
                  SET-OF (curso QUE HA-SIDO-TOMADO-POR estudiante) }

   NOMBRE-ESTUDIANTE-DE-TODAS-LAS-MAT := { nombre : nombre ES-NOMBRE-
                                           DE estudiante AND estudiante
                                           IN ESTUDIANTE-DE-TODAS-LAS-MAT }
```

Estos tres pasos se pueden explicar como sigue:

1. El primer paso define CURSO-MAT, el conjunto de todos los cursos de matemáticas y es idéntico a la definición de CURSO-MAT que se dio en la sección anterior.
2. El segundo paso define ESTUDIANTE-DE-TODAS-LAS-MAT, el conjunto de estudiantes que han tomado todos los cursos de matemáticas e incluye una notación no familiar que se explicará a continuación.

Recuerde que para cada estudiante queremos comparar el conjunto de los cursos de matemáticas —CURSOS-MAT— con el conjunto de los cursos que ha tomado el estudiante. En el paso 1 se definió el conjunto de los cursos de matemáticas. ¿Cómo definimos el conjunto de cursos tomado por cada estudiante en particular? Esto se hace con la construcción:

```
SET-OF (curso QUE HA-SIDO-TOMADO-POR estudiante )
```

Esta construcción SET-OF se usa para generar el conjunto de cursos que han sido tomados por el estudiante particular que está siendo evaluado por la consulta. Por ejemplo, si el estudiante es Ann, entonces esta construcción es equivalente a:

```
{ curso : curso HA-SIDO-TOMADO-POR 'ANN' }
```

Este es un ejemplo de un *conjunto generado dinámicamente*. Esto es, se genera un conjunto diferente para cada valor de la variable estudiante. Al ejecutar la consulta, la construcción SET-OF generará un conjunto para Mary, un conjunto para John, un conjunto para Ann, un conjunto para Kelly y un conjunto para Roger.

En el paso 2, el sistema evalúa cada estudiante en ESTUDIANTE comparando el conjunto CURSO-MAT con el conjunto de cursos tomados por el estudiante, usando el operador de comparación IS-CONTAINED-IN. Si el conjunto completo de cursos de matemáticas está contenido en el conjunto de cursos del estudiante, entonces la expresión de calificación que usa el operador IS-CONTAINED-IN se evalúa como "verdadero". En ese caso se selecciona el estudiante para incluirlo en el conjunto ESTUDIANTE-DE-TODAS-LAS-MAT.

3. NOMBRE-ESTUDIANTE-DE-TODAS-LAS-MAT nos da los nombres de los estudiantes que han tomado todos los cursos de matemáticas. Se podría haber evitado este paso ampliando la expresión de calificación del paso 2 para incluir este paso de expresión de calificación, pero esto hubiera complicado el paso 2 y dificultado su comprensión. Por la misma razón hemos roto el paso 1 como un paso separado. Por tanto, se puede ver que la facilidad de asignarle nombres a los conjuntos derivados permite descomponer la solución de la consulta en una serie de pasos. Esto hace al proceso más simple y fácil de manejar.

Aplicando esta consulta a la base de datos de nuestro ejemplo y usando la información de la Figura 14.7(c) y 14.7(e), vemos que Roger es el único estudiante que ha tomado todos los cursos de matemáticas. Por tanto, la respuesta a la consulta es { ROGER}.

El inverso del operador IS-CONTAINED-IN es el operador CONTAINS. Si lo hubiésemos usado en el ejemplo precedente, el paso 2 de la solución hubiese sido:

```
ESTUDIANTE-DE-TODAS-LAS-MAT :=

{ estudiante : SET-OF (curso QUE HAN-SIDO-TOMADO-POR estudiante)
               CONTAINS CURSO-MAT }
```

Veamos otro ejemplo.

Consulta usuario: ¿Qué estudiantes han tomado cada uno de los cursos ofrecidos por el departamento en el que está matriculado?

En el problema anterior comparamos cada conjunto de cursos de un estudiante con un conjunto fijo de cursos (CURSO-MAT). En este problema debemos comparar los cursos de cada estudiante con un conjunto de cursos que varía por cada estudiante, puesto que el departamento en el que está matriculado varía de un estudiante a otro.

```
Solución TextQuery:
   ESTUDIANTE-DE-TODOS-LOS-CURSOS := { estudiante :
                SET-OF (curso QUE HA-SIDO-TOMADO-POR estudiante )
                CONTAINS SET-OF (curso QUE ES-OFRECIDO-POR
            departamento QUE ESTÁ-DIRIGIENDO-ESTUDIANTE
                                    estudiante) }
```

Este ejemplo muestra que se pueden comparar dos conjuntos que han sido generados dinámicamente. Por supuesto, esto sólo nos da los estudiantes, no sus nombres. El conjunto de nombres se obtiene de la misma forma que se hizo en la consulta anterior.

Conectores booleanos

En el primer ejemplo de TextQuery se usó el *conector booleano* (o lógico) AND en la expresión de calificación. Las expresiones de calificación para las otras consultas se consideraron *atómicas* en el sentido de que estaban constituidas por expresiones simples no conectadas por ningún AND, OR o NOT. Como en la mayoría de los lenguajes de computadores, TextQuery nos permite combinar expresiones de calificación atómicas usando los conectores booleanos. Además se pueden usar paréntesis para agrupar las expresiones y eliminar ambigüedades. Esto le da a TextQuery una mayor potencialidad.

Consulta usuario: ¿Qué estudiantes están repitiendo clases?

```
Solución en TextQuery:
  ESTUDIANTE-REPITENTE := { estudiante : estudiante ESTÁ-TOMANDO curso
                              AND estudiante HA-TOMADO curso }
```

Recordemos que la variable *curso* en esta expresión de calificación tiene el mismo valor a lo largo de la expresión. Esto es, el significado de *curso* en

```
estudiante ESTÁ-TOMANDO curso
```

es el mismo que en

```
estudiante HA-TOMADO curso.
```

El conector booleano AND, que ha sido usado aquí para juntar dos expresiones atómicas, significa que la expresión de calificación como un todo es verdadera para un estudiante y un curso sólo si *ambos*

```
estudiante ESTÁ-TOMANDO curso
```

y

```
estudiante HA-TOMADO curso
```

son verdaderas.

Como antes, se necesita otro paso para obtener los nombres de los estudiantes. Para mantener simple esta presentación, omitiremos este paso a lo largo de esta sección.

Consulta usuario: ¿Qué estudiantes han tomado o están tomando un curso de historia?

```
{ estudiante : estudiante HA-TOMADO curso QUE ES-OFRECIDO-POR
               departamento QUE TIENE-NOMBRE 'HISTORIA' OR
               estudiante ESTÁ-TOMANDO curso QUE ES-OFRECIDO-POR
               departamento QUE TIENE-NOMBRE 'HISTORIA' }
```

Una segunda versión más simple crea el subconjunto historia de CURSO y usa el operador IN.

```
CURSO-HISTORIA := { curso : curso ES-OFRECIDO-POR
                    departamento QUE TIENE-NOMBRE 'HISTORIA'}

ESTUDIANTE-HA-TOMADO-HISTORIA :=
    { estudiante :  ( estudiante HA-TOMADO curso AND
                curso IN CURSO-HISTORIA) OR
                 (estudiante ESTÁ-TOMANDO curso AND
                    curso IN CURSO-HISTORIA ) }
```

Aquí el operador booleano OR se interpreta para hacer la expresión de calificación verdadera bien cuando sea verdadera la primera parte o la segunda parte (o ambas) de la expresión de calificación. También usamos AND en cada parte, por lo que agrupamos las partes entre paréntesis para prevenir ambigüedades potenciales.

Consulta usuario: ¿Qué estudiantes asesorados en matemáticas aún no han tomado un curso de matemáticas?

```
CURSO-MAT := { curso : curso ES-OFRECIDO-POR departamento
                       QUE TIENE-NOMBRE 'MAT' }

ESTUDIANTE-HA-TOMADO-MAT :=
        { estudiante : (estudiante HA-TOMADO curso AND
            curso IN CURSO-MAT ) OR
        (estudiante ESTÁ-TOMANDO curso AND
            = curso IN CURSO-MAT ) }

NUEVO-ESTUDIANTE-MAT :=
        { estudiante : estudiante ESTÁ-MATRICULADO-EN departamento QUE
              TIENE-NOMBRE 'MAT'
         AND estudiante NOT IN ESTUDIANTE-HA-TOMADO-MAT }
```

La última línea de la expresión de calificación podría haber sido escrita:

```
AND NOT estudiante IN ESTUDIANTE-HA-TOMADO-MAT
```

Es decir, el NOT se puede meter dentro de la parte atómica de la expresión para negar el efecto del IN, o puede ser parte del conector booleano AND para negar la expresión atómica completa. El NOT también se puede aplicar a otros operadores de comparación (CONTAINS, IS-CONTAINED-IN, <, >, =).

TextQuery proporciona un medio simple, elegante y natural de extraer los datos de una base de datos orientada a objetos. Es similar a muchos lenguajes de programación tradicionales que usa la representación textual de los datos, las variables y las sentencias. Tiene también una atractiva conexión con el lenguaje natural, puesto que los conjuntos de objetos y las interrelaciones sobre las que opera se modelan como nombres y verbos [6].

[6] Aquí el lector habrá notado una combinación de términos en inglés del TextQuery con nombres y verbos del español. Para sentir más aún esta conexión del TextQuery con el lenguaje natural tendríamos que tener un TextQuery para el español (N. del T.).

▼ Introducción a los sistemas basados en el conocimiento

El desarrollo de los sistemas de bases de datos inicialmente estuvo motivado por la necesidad de un almacenamiento eficiente y de recuperar grandes cantidades de diferentes tipos de datos. En la medida en que se ha ido progresando en el cumplimiento de estos objetivos, ha ido surgiendo un interés paralelo en extender la potencialidad de los sistemas de bases de datos, de modo que incluyan reglas que operen sobre los hechos (datos) almacenados, permitiendo inferir otros hechos (datos). Esta es una perspectiva excitante, puesto que la incorporación de tal expresividad en un sistema de base de datos eleva sus capacidades para servir a las necesidades de información a otro nivel.

Incorporar conocimiento en un sistema de base de datos fue sugerido por los desarrollos en el campo de la inteligencia artificial (IA), que está dedicada al estudio de cómo programar comportamiento inteligente. La investigación en IA incluye estudios sobre la representación de reglas lógicas que operan sobre los datos. Los **sistemas expertos** son un tipo especial de desarrollo en IA que busca representar las reglas y los procedimientos que usan los expertos para resolver problemas en un dominio particular de problemas, tal como la medicina, la reparación de motores, el diseño por computador y otros. Tales sistemas descansan sobre una base de conocimientos para razonar sobre conjuntos de datos.

Sistemas expertos. Un sistema que modela el proceso de toma de decisión de los expertos en varios dominios de problema, tales como diagnóstico médico, decisiones de auditoría, etcétera; un tipo especial de desarrollo de IA.

Aunque los estudios de los sistemas expertos han estado fuertemente influenciados por los métodos de representación del conocimiento usando reglas lógicas, los sistemas expertos no son sistemas basados en el conocimiento porque no ofrecen todas las capacidades de gestión de datos de un SGBD. Por ejemplo, PROLOG, que es un lenguaje popular de sistemas expertos, podría verse como que proporciona un puente natural entre los sistemas de bases de datos y los sistemas basados en el conocimiento, ya que PROLOG está basado en el cálculo de predicados y sus predicados pueden verse como relaciones. También tiene la capacidad de expresar la lógica que puede usar una persona para convertir hechos en la base de datos en información de ayuda a la toma de decisiones. No obstante, las versiones actuales de PROLOG no ofrecen el rango necesario de capacidades de los SGBD —tales como el procesamiento de transacciones, la seguridad y la recuperación, y la gestión del almacenamiento secundario— que requeriría un sistema basado en el conocimiento. DATALOG y Logical Data Language (LDL) [7] son versiones extendidas de PROLOG que están en desarrollo y que atienden este problema.

sistema de base en el conocimiento. Un sistema que proporciona el rango completo de capacidades para almacenamiento y manipulación de datos, así como facilidades para la creación, almacenamiento y ejecución de reglas de inferencia en tablas de datos almacenadas.

Como el estudio de los sistemas basados en el conocimiento es nuevo, hay varias definiciones relacionadas con esto que aún no han sido definidas —un problema que puede oscurecer una discusión sobre los sistemas basados en el conocimiento—. Por ejemplo, algunos autores usan el término **sistemas basados en el conocimiento** (*knowledge based systems*), mientras que otros dicen **sistemas de base de conocimientos** (*knowledge-base systems*). Ambos términos han sido ampliamente utilizados para representar el mismo tema. Hemos seguido el ejemplo de Ullman (1990), quien ha sido una figura prominente en el desarrollo de los conceptos de los sistemas de base de conocimientos.

sistema basado en el conocimiento. Otra forma de nombrar a un sistema de base de conocimientos.

Para aclarar este término, vamos a ofrecer una definición de trabajo de un *sistema de base de conocimientos SBC* (*knowledge base system KBS*) como sigue:

1. Una base de datos de hechos básicos, como los de un sistema de base de datos.
2. Una base de datos de reglas que permiten hacer deducciones a partir de los hechos que están en la base de datos.
3. Un software llamado **sistema de gestión de base de conocimientos (SGBC)** [8] que soporta las funciones usuales de los SGBD, así como la gestión de procesos deductivos de las base de datos de reglas que operan sobre la base de datos de hechos.

sistema de gestión de base de conocimientos. Sistema de software que soporta el rango usual de las funciones de los SGBD, así como gestiona el proceso deductivo de la base de reglas que opera sobre la base de hechos.

[7] Lenguaje de datos lógico (N. del T.).

[8] Conocido en inglés por la siglas KBMS (*Knowledge Base Management System*) (N. del T.).

Aunque hay varias definiciones ligeramente diferentes en la literatura, pensamos que la definición anterior recoge las ideas esenciales.

En la actualidad hay un número de áreas activas de desarrollo de los SBCC. Se ha diseñado hardware para mejorar la velocidad con la que razonamos con las reglas. Se han desarrollado métodos para el mantenimiento automático de la integridad semántica de las bases de conocimientos usando reglas expresadas en lenguajes basados en la lógica formal. El uso de la lógica para expresar y razonar con el conocimiento, involucrando la incertidumbre, las suposiciones y el tiempo, está siendo estudiado y refinado.

En las secciones siguientes nos extenderemos en algunos de estos temas. Primero examinaremos el concepto de conocimiento y cómo se relaciona con las bases de datos.

▼ Conocimiento y bases de datos

El amplio éxito de los sistemas de bases de datos, combinado con las necesidades de gestión de información y los desarrollos que han emanado del estudio de la inteligencia artificial, han dado como resultado un interés creciente en extender los sistemas de bases de datos a sistemas de base de conocimientos. Esta es una transición a altos niveles de abstracción de la información, Por ejemplo,

```
El vendedor Smith radica en Los Angeles
```

es un hecho típico que se puede mantener en una base de datos. Hay una pequeña dificultad para pensar en esta información en términos de cómo podría estar contenida en una tupla relacional:

```
vendedor(Smith, Los Angeles, ...)
```

Por otro lado, información como

```
El vendedor Smith es un suministrador dependiente
```

no es precisamente un hecho, pero representa una alta forma de información. Tales formas de información a menudo dependen de interrelaciones lógicas entre los hechos de la base de datos. Por ejemplo, un " vendedor dependiente" puede identificarse a partir de las interrelaciones entre fechas de órdenes y fechas de envío, así como también como embarques parciales y completos. Ser capaces de especificar tales interrelaciones eleva la utilidad de la base de datos hasta donde se puede usar el conocimiento para definir, controlar e interpretar los datos que ésta mantiene. Qué significa esto se resume en la taxonomía siguiente desarrollada por Wiederhold (1984):

conocimiento estructural. Conocimiento sobre dependencias y restricciones entre los datos.

conocimiento general procedimental. Conocimiento que sólo se puede describir por un procedimiento.

conocimiento específico de la aplicación. Conocimiento que se determina por las reglas y las convenciones que se aplican a un dominio específico de problema.

conocimiento de dirección-empresa. Conocimiento que ayuda a una empresa a tomar decisiones.

1. **Conocimiento estructural** es el conocimiento sobre las dependencias y restricciones entre los datos. Por ejemplo, la "inserción en la relación CLIENTE depende de establecer que el cliente tenga un buen crédito".
2. **Conocimiento general procedimental** es el conocimiento que sólo puede ser descrito por un procedimiento. Por ejemplo, " haga corresponder el total de producto ordenado con el total de producto recibido para autorizar los pagos".
3. **Conocimiento específico de la aplicación** es el conocimiento que está determinado por las reglas y convenios que se aplican en un dominio de problema en particular. Por ejemplo, "determine el plan de menor-costo para dos semanas de vacaciones en Hawai".
4. **Conocimiento dirección-empresa** es el conocimiento que ayuda a una empresa a tomar decisiones. Por ejemplo, un fabricante de alimentos ligeros integra información sobre los costos, las ganancias, las ventas y los productos competitivos para auxiliarse en la toma de decisiones estratégicas.

conocimiento extensional. Hechos que se almacenan en las relaciones de la base de datos.

conocimiento intensional. Conocimiento que se deduce del conocimiento extensional por la aplicación de reglas.

El conocimiento que existe en los sistemas de bases de datos analizados con anterioridad a este capítulo está constitnido por hechos e instancias que están almacenados en las relaciones. Esto se conoce como **conocimiento en extenso**. El conocimiento que se define más allá del contenido de hechos de la base de datos, usualmente mediante mecanismos lógicos, se denomina **conocimiento en intenso**. La mayoría de los sistemas de bases de conocimientos que están en desarrollo almacenan conocimiento en intenso en la forma de reglas lógicas.

Aclaremos estas ideas con nn ejemplo. Examine la Figura 14.15. Nuestra base de datos consiste de una única tabla, PARTES, que contiene el atributo SUBPARTE y PARTE. Podemos recnperar los nombres de todas las partes, para la cnal 250 sea una subparte mediante la consulta del cálculo relacional:

```
GET(X) :- PARTES(250, X)
```

Esta sintaxis es sólo ligeramente diferente de la que hemos visto previamente y se interpreta de una forma familiar. En español podría leerse como "De la relación PARTES recuperar todas las tuplas que tienen el valor 250 en la columna SUBPARTE, luego asignar el valor correspondiente de PARTE a la variable X y almacenar el valor de X como nna tupla en la relación GET".

En este ejemplo, los valores resultantes para X son 300 y 315.

PARTES =	*SUBPARTE*	*PARTE*
	200	315
	250	300
	250	315
	300	324
	315	350

Llevando la información de esta consulta un paso más adelante, vemos que la parte 300 de la cual 250 es una subparte, es a su vez una subparte de la parte 324. Esto se puede ver inspeccionando la relación PARTES. Por transitividad, entonces, 250 es también una subparte de la parte 324 y por lo tanto la parte 324 debería estar probablemente en la solución de la consulta. Lógicamente, esta transitividad entre parte y subparte podría seguir y seguir, pero en este ejemplo de PARTES hay sólo dos niveles de subpartes. Sin embargo, esto basta para apreciar que sería conveniente si pudiéramos escribir una consulta para encontrar todas las partes para la cual una parte dada es una subparte primitiva (segundo nivel). (Por ejemplo, la parte 250 es una subparte primitiva de la parte 324.) Una alternativa es añadir una tabla que relacione las partes y sus subpartes primitivas, tal como se ilustra en la Figura 1.16. La consulta para encontrar las partes para la cual la parte 250 es una subparte primitiva es

```
GET(X) :- PARTES_PRIMITIVAS(250, X)
```

El resultado es 324 y 350.

PARTES_PRIMITIVAS =	*PARTE PRIM*	*PARTE*
	200	350
	250	324
	250	350

Desafortunadamente, la tabla PARTES_PRIMITIVAS contiene datos redundantes, esto es, datos que ya están disponibles en PARTES. Ésta se derivó de la tabla PARTES, usando nuestro conocimiento de que una subparte primitiva es una subparte de segundo nivel. Construir tablas especiales para responder a cada necesidad imaginable de información sería muy ineficiente. Es mejor si sólo tenemos que darle a la base datos una regla que establezca las interrelaciones deseadas y opere directamente sobre la base de datos extensional original. Este es precisamente el tipo de capacidad que se consigue con un SBC.

Volviendo al ejemplo, sea PARTES la base de datos en extenso. Se define entonces la base de datos en intenso como

```
PARTES_PRIMITIVAS(X, Z) :- PARTES(X, Y) AND PARTES(Y, Z)
```

Esto se puede leer como

> En la relación PARTES_PRIMITIVAS , X es una subparte primitiva de Z [PARTES_PRIMITIVAS(X, Z)] si (:-) en la relación PARTES, X es una subparte de Y [PARTES(X,Y)] e Y es una subparte de Z [PARTES(Y,Z)]

Hemos definido simplemente una regla para operar sobre la tabla PARTES para recuperar información sobre las subpartes primitivas. Ahora podemos escribir una consulta simple para recuperar la información deseada de PARTES:

```
GET(Z) :- PARTES_PRIMITIVAS(250, Z)
```

El resultado es de nuevo 324 y 350 y se ha eliminado el problema de estar manteniendo datos redundantes.

Uno puede imaginarse lo tedioso que resulta nn sistema en el que tales consultas se satisfagan creando tablas separadas, tales como PARTES_PRIMITIVAS. La forma en intenso permite la especificación de los mismos datos, pero de forma compacta: Se reduce la redundancia y se ahorra espacio de almacenamiento.

Para ilustrar más los conceptos fundamentales y darle al lector conocimiento suficiente para que lleve a cabo por sus propios medios ejercicios y proyectos, vamos a introducir un lenguaje de implementación lógica. Los conceptos de lenguaje que se presentarán son simples y proporcionan una vía concreta y fácil de transición de los ejemplos que se han discutido a aplicaciones más amplias. Se ha escogido PROLOG, una implementación de las mismas ideas del cálculo relacional que estudiamos antes, que también incluye capacidades para crear reglas que operen sobre bases de datos en extenso. Mas aún, PROLOG forma la base para algunos de los lenguajes de SBC que se están desarrollando, tales como DATALOG y LDL.

Aunque la mayoría de las versiones de PROLOG usan una notación más o menos estándar, pueden aparecer algunas pequeñas diferencias. Usamos una forma general.

▼ Representación del conocimiento con reglas

En esta sección elaboraremos los mecanismos para expresar lógica con reglas. Describiremos la sintaxis de las reglas básicas y daremos un ejemplo general de combinación de las reglas en las estrategias de evaluación con encadenamiento hacia atrás (*backward-chaining*) y con encadenamiento hacia adelante (*forward-chaining*). Luego se presentará la expresión de las reglas en PROLOG.

Formación de reglas

Las reglas son un método muy intuitivo de representar el conocimiento. Confiamos en la intuición del lector en la interpretación de las reglas que ilustramos en la discusión anterior, a su vez también podrá probablemente escribir algunas de sus propias reglas al estudiar dichos ejemplos.

Una regla está constituida por una hipótesis y una conclusión. Si se satisface la hipótesis entonces se puede inferir una conclusión. Por ejemplo, "*Si P es un perro (hipótesis), entonces D tiene cuatro patas (conclusión)*". La conclusión también puede ser una acción, como en el caso, "*Si la fiebre está acompañada de una congestión (hipótesis), debiera irse a ver al médico (conclusión-acción)*".

Reglas simples como éstas a menudo no son útiles si no se combinan con esquemas más ricos de representación del conocimiento. Hay dos métodos generales de combinar las reglas: Uno es el encadenamiento hacia adelante (*forward chaining*) y la otra es el encadenamiento hacia atrás (*backward chaining*). Para explicar estos conceptos usaremos una analogía muy simple.

Piense en un vuelo desde Salt Lake City hasta Bangkok como en un conjunto de reglas en las que Salt Lake City es el punto de partida y Bangkok es el punto de destino. La deducción, que combina las reglas, se aplica a la tarea de encontrar los vuelos que conecten las ciudades.

No hay vuelos directos entre Salt Lake City y Bangkok. Lo que debemos hacer es examinar el libro de renta con el cronograma que describe todos los vuelos y tratar de encontrar los vuelos que conecten. Que vayan, digamos, a los Angeles, luego a Honolulú, luego a Tokio y luego a Bangkok. Encontramos, por ejemplo, que hay varios vuelos llegando desde Tokio. Cada segmento de llegada desde una ciudad hasta Bangkok puede verse como una regla. De modo que

```
GET(X) :- SEGMENTO(Bangkok, X)
```

genera uua lista de ciudades desde las cuales hay vuelos hasta Bangkok. Tokio estaría en esta lista. En español mondo y lirondo esto se diría de la forma siguiente:

> Si una ciudad tiene vuelos a Bangkok (hipótesis), entonces esa ciudad es un candidato para nuestro viaje (conclusión).

Simplificando esto un poco, ya que Tokio es una de estas ciudades, podemos hacer de Tokio nuestro destino y ver todos los vuelos que lleguen a Tokio, entre los cuales hay uno saliendo desde Honolulú. Podemos convertir entonces a Honolulú en nuestro destino y continuar trabajando de esta manera hasta que encontremos una conexión con Salt Lake City. En este caso estamos trabajando hacia atrás desde Bangkok, por lo que el proceso se llama **encadenamiento hacia atrás** (*backward chaining*).

encadenamiento hacia detrás. Una cadena lógica de reglas que van de una conclusión a una hipótesis.

Alternativamente podríamos haber comenzado en Salt Lake City y comenzar hacia adelante. En este caso buscamos todos los vuelos que salen de Salt Lake City y vemos que van, digamos, a Los Angeles o Honolulú, o Denver o Chicago para ir a Tokio y de Tokio a Bangkok. Esta es a grandes rasgos la metodología que se conoce como **encadenamiento hacia adelante** (*forward chaining*).

encadenamiento hacia delante. Una cadena de reglas que van de una hipótesis a una conclusión.

Reglas en PROLOG

En PROLOG las reglas se expresan de forma familiar.

$$C :- H_1 \ \& \ H_2 \ \& \ \ldots\ldots\ldots \ \& \ H_k.$$

Donde el símbolo ":-" se puede leer como "if" , algo que hemos estado haciendo informalmente. La cláusula anterior de PROLOG se lee como

> *C es verdadero, si H_1 y H^2 ... y así sucesivamente hasta Hk son verdaderos.*

También podría usarse "or" en lugar de " and". El punto (.) denota el fin de la cláusula de la regla.

Estos son algunos ejemplos concretos y familiares.

```
progenitor(X, Y) :- madre(X, Y) o padre(X, Y)
```

(Siguiendo el convenio de PROLOG hemos pasado a usar letras en minúscula para denotar las relaciones.)

Esto podría leerse como

X es progenitor de Y si X es la madre de Y o si X es el padre de Y.

¿Cómo determinamos la veracidad de esta pregunta? ¿Es X un progenitor de Y o no? Para encontrar la respuesta, X e Y deben asociarse a instancias reales (valores) de la base de datos. Luego la verdad de una sentencia dependerá de la verdad de sus partes. Supongamos que los siguientes hechos existen en la base de datos extensional. (Los valores en PROLOG deben empezar con letras minúsculas, por eso "Jane" se escribe " jane" y así.)

```
madre(jane, alex)
madre(tami, anne)
padre(john, ane)
```

Ahora nos planteamos la consulta PROLOG,

```
?progenitor(john, alex).
```

En la regla de progenitor, X está ahora asociado con john y Y con alex. Para calcular la solución, se debe demostrar que la regla es verdadera o falsa. En este caso, tenemos

```
madre(john, alex)
```

no es un hecho de la base de datos, y

```
padre(john, alex)
```

tampoco es un hecho de la base de datos. Por lo tanto,

```
progenitor(john, alex)
```

no se puede deducir de la base de datos y la respuesta es FALSE (falso). Sin embargo, la consulta

```
?progenitor(tami, anne)
```

se interpretará como verdadera puesto que

```
madre(tami, anne)
```

es un hecho de la base de datos, lo que es suficiente para determinar que

```
progenitor(tami, ane)
```

es TRUE (verdadero).

Supongamos que lo que se quiere es conocer quiénes son los progenitores de Anne. Esto es,

```
?progenitor(X, anne).
```

En palabras estaríamos diciendo:

> Buscar los nombres que califiquen como progenitores de Anne. Luego asignar éstos, uno a uno, a la variable X.

(Los nombres de variable en PROLOG deben empezar con una letra mayúscula.) Una búsqueda en los hechos de la base de datos encuentra

```
madre(tami, anne), y
padre(john, anne)
```

Luego el resultado de la consulta será

```
X = tami
X = john
```

Los lenguajes basados en la lógica brindan las capacidades de responder a preguntas que no se pueden responder fácilmente con los lenguajes de manipulación de datos de los lenguajes convencionales. Aunque la creación de vistas en los lenguajes convencionales es similar al uso de reglas para definir bases de datos en intenso, las capacidades de definición de vistas de los SGBDs relacionales no son tan poderosas como las reglas lógicas. En particular, los lenguajes relacionales no pueden expresar consultas recursivas que son a menudo útiles para formular consultas complejas (ver Ullman, 1991, para detalles.) Las interrelaciones transitivas tales como la jerarquía entre partes, las listas de ancestros y las jerarquías de dirección son ejemplos importantes.

Supongamos que tenemos trabajadores que son dirigidos por los jefes de departamento y que los jefes de departamento son dirigidos por los jefes de división. Tanto los jefes de división como los jefes de departamento pueden entonces estar implicados en la evaluación del rendimiento del trabajador. Esta interrelación lógica se puede expresar recursivamente de la forma siguiente:

```
evalúa(X, Y) :- jefede(X, Y).      (1)

(se lee como " X evalúa a Y si X es jefe de Y").

evalúa(X, Y) :- evalúa(Z, Y) & jefede(X, Z).      (2)

(se lee como " X evalúa a Y si Z evalúa a Y y X es el jefe de Z" ).
```

Suponga que los hechos siguientes existen en la base de datos en extenso:

```
jefede(john, bob)

jefede(john, ray)

jefede(bob, frank).
```

Podemos plantearnos la siguiente consulta:

```
?evalúa(john, bob).
```

El resultado es verdadero por la regla (1), ya que

```
jefede(john, bob)
```

es un hecho. Sea ahora la consulta

```
?evalúa(john, frank).
```

El resultado es verdadero por la regla (2), ya que

```
evalúa(bob, frank) y jefede(john, bob)
```

son ambas verdadero.

Supongamos que se quiere una lista de todos los empleados que son evaluados por John. Esto podría expresarse como sigue:

```
?evalúa(john, X).
```

El resultado sería

```
X = bob     por la regla (1)
X = ray     por la regla (1)
X = frank   por la regla (2).
```

Se ha presentado un ejemplo en el que el camino de la evaluación (jerárquico) es de longitud dos —hay un máximo de dos niveles en la evaluación—. El uso de una regla recursiva extiende con facilidad el camino a cualquier longitud. Veamos una simple demostración de extender el camino a longitud tres. Supongamos que se añade un hecho a la base de datos anterior:

```
jefede(frank, carl).
```

Y hacemos la consulta:

```
?evalúa(john, carl).
```

El resultado será verdadero mediante la secuencia lógica siguiente:

1. La regla (2) establece la veracidad de "evalúa(bob, carl)," ya que "evalúa(frank, carl)" es verdadero por la regla (1) y " jefede(bob, frank)" es un hecho de la base de datos.
2. De nuevo una aplicación de la regla (2) nos lleva al resultado, ya que "evalúa(bob, carl)" ya ha sido establecido y " jefede(john, bob)" es un hecho de la base de datos.

Los conceptos básicos que se han presentado son:

1. Las reglas se pueden expresar como cláusulas que tienen la forma

```
<conclusión> :- <lista de hipótesis>.
```

La parte de la izquierda del ":-" es verdadera si se puede establecer la veracidad de la parte a la derecha del ":-" a partir de hechos de la base de datos o de la veracidad de otras reglas.

2. Se pueden desarrollar bases de datos en intenso (deductivas) a partir de bases de datos en extenso a las que se le añaden tales reglas.

Veamos ahora más de cerca a PROLOG como lenguaje para expresar reglas.

▼ Una aplicación simple de base de datos en PROLOG

En esta sección destacaremos la sintaxis fundamental de PROLOG y demostraremos una aplicación modesta de base de datos.

Aspectos fundamentales de PROLOG

lógica de primer orden. Una estructura lógica que se caracteriza por un conjunto de objetos, un conjunto de predicados (cada uno de los cuales evalúa verdadero o falso) y un conjunto de funciones.

cláusula. En PROLOG el medio por el cual se expresan los hechos y el conocimiento: <conclusión> :- <lista de hipótesis>

PROLOG (*Programación Lógica*)[9] es un lenguaje cuyas instrucciones son fórmulas de la **lógica de primer orden**, las bases para codificar el conocimiento como reglas. PROLOG es el lenguaje fundacional del proyecto de quinta generación de computadores, cuyo objetivo es desarrollar sistemas de cálculo altamente inteligentes que puedan almacenar vastos volúmenes de información. PROLOG es también una herramienta importante en la programación en inteligencia artificial, particularmente en el desarrollo de sistemas expertos.

La componente fundamental de PROLOG es la **cláusula**, el medio mediante el cual se expresan los hechos y el conocimiento. La Figura 14.17 muestra dos cláusulas básicas: padre(harry, jane) y padre(X, jane). La primera cláusula dice, " harry es el padre de jane." La segunda cláusula especifica que " X es el padre de jane, " donde X especifica a una variable.

También podemos escribir cláusulas condicionales que expresen reglas de la forma que hemos visto. La Figura 14.17 nos muestra dos reglas: La primera dice que "X es el progenitor de Y" es verdadero si " X es la madre de Y" es verdadero (progenitor(X,Y) :- madre(X,Y)); la segunda dice, " X es progenitor de Y" es verdadero si " X es padre de Y es verdadero" (progenitor(X, Y) :- padre(X, Y)).

Las conjunciones (ANDs) se denotan usando comas y las disyunciones (ORs) se denotan por puntos y coma. Por ejemplo, podemos tener una regla básica como " X es el abuelo de Y, si X es el padre de Z, y Z es un progenitor de Y."

```
abuelo(X, Y) :- padre(X, Z), progenitor(Z, Y).
```

Toda cláusula en PROLOG debe terminar con un punto.

```
(a)   CLÁUSULAS

      padre        (harry , jane).
      padre        (X, jane).

(b)   CLÁUSULAS CONDICIONALES

      progenitor   (X, Y) :-    madre   (X, Y).
      progenitor   (X, Y) :-    padre   (X, Y).

(c)   CONJUNCIONES Y DISYUNCIONES

      abuelo       (X, Y) :-    padre   (X, Z), progenitor  (Z, Y).
      progenitor   (X, Y) :-    madre   (X, Y); padre       (X, Y).
```

[9] En inglés, *Programming in Logic* (N. del T.).

La estructura de una aplicación en PROLOG

símbolos predicados. Nombres que se aplican a los argumentos para expresar un predicado.

Las instrucciones en PROLOG se componen de fórmulas que incluyen **símbolos de predicados** (tales como " progenitor", " padre" o " abuelo") aplicadas a los argumentos para producir valores "verdadero" o "falso". Estos argumentos pueden incluir constantes (tales como "harry" y " jane") y variables (tales como X, Y y Z). (Las constantes también pueden ser enteros.) El convenio es que en PROLOG los símbolos predicados y las constantes comiencen con letra minúscula.

Vamos a aclarar estas nociones. Indiquemos primero la similitud entre un nombre relacional y sus atributos y la noción lógica de un predicado con sus argumentos. Refiérase de nuevo a la Figura 14.15. Esta muestra una relación de nombre PARTES con los atributos asociados, SUBPARTE y PARTE. En PROLOG, esta relación es un predicado, PARTES, que tiene los argumentos SUBPARTE y PARTE. De modo que si expresamos el predicado

```
?partes(200, 315),
```

deberá retornarse como respuesta TRUE (verdadero), ya que la tupla <200, 315> es una de las tuplas de la relación correspondiente.

Se puede construir un ejemplo algo más complejo usando la Figura 14.18. Los nombres de tabla y los nombres de atributo tienen la interpretación siguiente:

Nombre de atributo	*Interpretación*
v	vendedor
vp	vendedor de partes
p	partes
númv	números de vendedor
nombrev	nombre de vendedor
localidad	localidad
númp	número de parte
nivel-c	nivel de calidad
nombrep	nombre de parte
tipop	tipo de parte
grado	grado

dominio. El término PROLOG para la especificación de un tipo de dato.

predicado. Una expresión que evalúa verdadero o falso.

Observe que los **dominios** corresponden a los tipos de datos, los **predicados** corresponden a estructuras relacionales y las cláusulas representan las tuplas de las relaciones. Definimos " vistala" mediante la fórmula lógica

```
vistala(12, Smith) :- v(12, smith, la)
```

La regla dice que (12, smith) es un hecho de la vistala si (12, smith, la) es un hecho del predicado v (vendedor). Un valor tal como "la" [10] apareciendo a la derecha de ":-", (pero no a la izquierda) puede interpretarse como una condición existencial. Esto es, para la regla anterior pudiéramos decir que

```
vistala(númv, nombrev) es verdadero si existe una tupla en v tal que
v(númv, nombrev, la) es verdadero.
```

ilustraremos ahora la estrnctura básica de una implementación PROLOG usando una versión simplificada de la Figura 14.18.

```
/* Ejemplo */
```

[10] Siglas como se conoce en inglés a la ciudad de Los Angeles (N. del T.).

```
domains
nombrev, localidad, nombrep, tipop, grado = string
númv, númp, nivel-calidad = integer

predicates
v(númv, nombrev, localidad)
vp(númv, númp, nivel-calidad)
p(númp, nombrep, tipop, grado)

clauses
v(1, james, la).
v(2, cline, london).
v(3, marx, denver).
v(4, myers, sf).
vp(1, 25, 8).
vp(1, 37, 9).
vp(1, 28, 6).
vp(1, 29, 7).
vp(1, 39, 9).
vp(2, 25, 9).
vp(2, 37, 9).
vp(3, 28, 7).
vp(3, 37, 7).
vp(3, 29, 8).
vp(4, 37, 8).
vp(3, 39, 8).
vp(4, 28, 7).
vp(4, 29, 7).
p(25, reborde, acero, a).
p(28, estribo, latón, aa).
p(29, cerrojo, hierro, a).
p(37, pinza, latón, aa).
```

```
domains
nombrev, localidad = string
númv = integer

predicates
v(númv, nombrev, localidad)

clauses
v(1, james, la).
v(2, cline, london).
v(3, marx, denver).
v(4, myers, sf).
```

La sección "domains" (dominios) permite especificar los tipos de los valores (tipos de datos) que se brindarán por el esquema relacional definido en la sección "predicados". La sección cláusulas permite la entrada de datos, o valores, para los respectivos atributos relacionales. La sección *clauses* (cláusulas) puede verse como las instancias de una base de datos relacional. Esta es la estructura básica de todas las implementaciones PROLOG.

A partir de nuestra versión simplificada de la Figura 14.18 hemos creado una base de datos extensional. Ésta contiene hechos y podemos ejecutar consultas ordinarias de for-

ma elemental. Supongamos que queremos listar los nombres de todos los vendedores de Los Angeles (" la").

La consulta sería:

```
v(_, Nombre_vendedor, la).
```

El resultado será

```
Nombre_vendedor = james,
```

¿Cómo se interpreta esta consulta? Esencialmente ésta transcurre de la manera siguiente:

1. Mira en la relación v las tuplas que tengan " la" como valor de localidad. Recupera el valor asociado para nombrev y asigna éste a la variable de nombre Nombre_vendedor.
2. Toda palabra que comience con letra mayúscula se considera una variable —este es el caso de Nombre_vendedor.
3. Toda palabra que comience con letra minúscula se interpreta como un valor de atribnto.
4. El uso de "_ " significa que el valor correspondiente de atributo no tiene ningún significado en la solución de la consulta y por lo tanto se ignora.

Aplicación de base de datos

Demostraremos ahora una aplicación de base de datos usando PROLOG. Esta base de datos se representará como se muestra en la Figura 14.18. Los datos que se introducen son fácilmente reconocibles. Observe que las cadenas de caracteres son todas letras en minúsculas. Hemos entrado las instancias de la base de datos en la sección de programa PROLOG identificada por "clauses" (cláusulas). Así las cosas tenemos una base de datos en extenso. Recordemos que la sección "predicates" (predicados) requiere una definición de la base de datos en términos de nombres de relación y sus correspondientes atributos y que la sección "domains" (dominios) define los tipos de datos que corresponden a los dominios de los atributos.

Cuanto más lejos vayamos, mejor. Para implementar una base de datos en intenso necesitamos hacer algo más. Supongamos que deseamos conocer aquellos vendedores cuyos productos sean todos de nivel 8 o superior. Una forma de hacer esto es encontrar aquellos vendedores que suministren partes que estén evaluadas con menos de 8. La información deseada podemos encontrarla buscando aquellos suministradores que están en el primer grupo, pero no en el segundo. Esto se ilustra a continuación:

```
Primer Grupo : altonivel(X) :- pv(X, _, Q) and Q >= 8.
```

Interpretación: altonivel(X) es el resultado que queremos para el primer grupo. La relación de nombre "altonivel" se ha escogido arbitrariamente. Los nombres de relación pueden ser cualquier combinación de letras, dígitos y caracteres de subrayado, pero deben empezar con una letra minúscula. X es una variable y el hecho de que la usemos en mayúscula significa que queremos imprimir el resultado. En español esta consulta podría interpretarse como: Si alguna tupla de la relación pv tiene un valor Q mayor o igual que 8, entonces su valor X (número de vendedor) es una solución y será por lo tanto miembro de la relación altonivel.

```
Segundo Grupo: bajonivel(X) :- -pv(X, _, Q) and Q < 8
```

Interpretación: Similar a la descrita anteriormente

Solución: `mejorcalidad(X) :- -altonivel(X) and (not (bajonivel(X))`

Interpretación: Un sumistrador está en la relación mejor calidad si está en el grupo que suministra las partes con niveles de calidad mayor o igual que 8, pero no está en el grupo de los suministradores de partes que tienen nivel de calidad menor que 8.

En la Figura 14.19 se muestra una combinación de una base de datos en intenso y en extenso. Debería comprobar que tipo a(X) produce una lista de los suministradores que proporcionan partes de grado-a.

```
domains
nombrev, localidad, nombrep, tipop, grado = string
númv, númp, nivel-calidad = integer

predicates
v(númv, nombrev, localidad)
vp(númv, númp, nivel-calidad)
p(númp, nombrep, tipop, grado)

clauses
v(1, james, la).
v(2, cline, london).
v(3, marx, denver).
v(4, myers, sf).
vp(1, 25, 8).
vp(1, 37, 9).
vp(1, 28, 6).
vp(1, 29, 7).
vp(1, 39, 9).
vp(2, 25, 9).
vp(2, 37, 9).
vp(3, 28, 7).
vp(3, 27, 7).
vp(3, 29, 8).
vp(4, 37, 8).
vp(3, 39, 8).
vp(4, 28, 7).
vp(4, 29, 7).
p(25, reborde, acero, a).
p(28, estribo, latón, aa).
p(29, cerrojo, hierro, a).
p(37, pinza, latón, aa).

altonivel(X) :- vp(X, __, Q) and Q >= 8.
bajonivel(X) :- vp(X, __, Q) and Q < 8.
mejorcalidad(X)    :-    altonivel(X)    and    not
(bajonivel(X)).
tipo1(X) :- p(Y, __, __, a) and vp(X, __, __) and not
(vp(X, Y)).
tipo2(X) :- vp(X, __, __).
tipoa(X) :- tipo2(X) and not (tipo1(X)).
```

▼ Datalog

Mientras PROLOG es una implementación útil de la programación lógica, no tiene la funcionalidad completa de un sistema de base de datos. Este no es el propósito para el cual fue diseñado. Sin embargo, su poderosa implementación de la lógica de primer orden ha llevado a que se intente acoplarlo con los sistemas tradicionales de bases de datos relacionales. La mayoría de esos esfuerzos no han producido sistemas ideales. DATALOG es un intento de integrar conceptos de la lógica con plenas capacidades de sistemas de bases de datos.

DATALOG se basa en la programación lógica y la demostración automática de teoremas. DATALOG es en muchos sentidos una versión simplificada de la programación lógica. En particular, DATALOG no permite símbolos de funciones en los argumentos, como sí lo permite PROLOG. DATALOG mantiene una estrecha relación con las bases de datos relacionales, usando símbolos de predicados para representar las relaciones, tal como se hizo con PROLOG.

▼ Lenguaje de datos lógicos (LDL)

LDL representa un intento de proporcionar en un solo lenguaje la expresividad de PROLOG combinado con la capacidad de ejecutar transacciones (actualizar, borrar y otras) de un entorno de base de datos relacional. La diferencia más importante entre PROLOG y LDL es que LDL nos da la expresividad de la lógica, así como la completa funcionalidad de un SGBD. El ejemplo siguiente es similar al que da Chimenti y otros (1990). El registro de un empleado podría tener la forma siguiente:

```
empleado (nombre (joe, smiley), admin, educación(secundaria, 1967)).
```

Si queremos tener más información sobre los empleados, cada subargumento podría refinarse en una descripción más detallada. Esto nos llevaría a un registro educacional completo para una persona:

```
empleado(nombre(joe, smiley), admin, educación( { (secundaria, 1967),
(escuela(harvard, licenciatura, mat), 1971) (escuela(harvard,
maestría, ing), 1973) } ) ).
```

El anidamiento de conjuntos le da una gran potencia y flexibilidad. Además LDL ha adaptado los convenios para expresar reglas de PROLOG en un lenguaje que tiene amplias capacidades de gestión de bases de datos.

En este capítulo hemos introducido los conceptos fundamentales de las SGBDO y los SBC.

Primero ilustramos cómo un lenguaje de OO implementa los conceptos que se presentaron en el Capítulo 4. Luego hicimos un resumen de los prototipos de SGBDOs y de los SGBDOs comerciales y presentamos un lenguaje orientado a objetos. Los sistemas basados en el conocimiento intentan extender la potencialidad de los sistemas de bases de datos más allá de los datos propiamente hacia información que se puede deducir de los datos. Con esto como premisa, consideramos el papel del conocimiento en los sistemas de bases de datos y cómo las reglas permiten la representación del conocimiento. Este tema se ilustró con la construcción de una base de datos que incluía conocimiento, usando PROLOG como vehículo de descripción.

Luego dimos un resumen de LDL, que ha extendido las capacidades de PROLOG en funciones de gestión de bases de datos, así como en la representación del conocimiento.

Hay una necesidad creciente de extender los sistemas de bases de datos para incorporarles las características analizadas en este capítulo. Es de esperar que en la próxima década ocurran desarrollos interesantes y útiles en estas áreas. Afortunadamente, los métodos de modelado desarrollados anteriormente se adaptan adecuadamente a estos avances.

1. Defina con sus propias palabras cada uno de los términos siguientes:
   a. clase
   b. objeto complejo
   c. sistema experto
   d. sistema de base de conocimientos
   e. conocimiento en extenso
   f. conocimiento en intenso

2. ¿Por qué un sistema de bases de datos orientado a objetos es preferible a un sistema de base de datos relacional?

3. Contraste los predicados y las cláusulas en PROLOG. Dé un ejemplo de cada uno.

4. ¿Por qué pudiera ser preferible un sistema basado en el conocimiento a un sistema de base de datos?

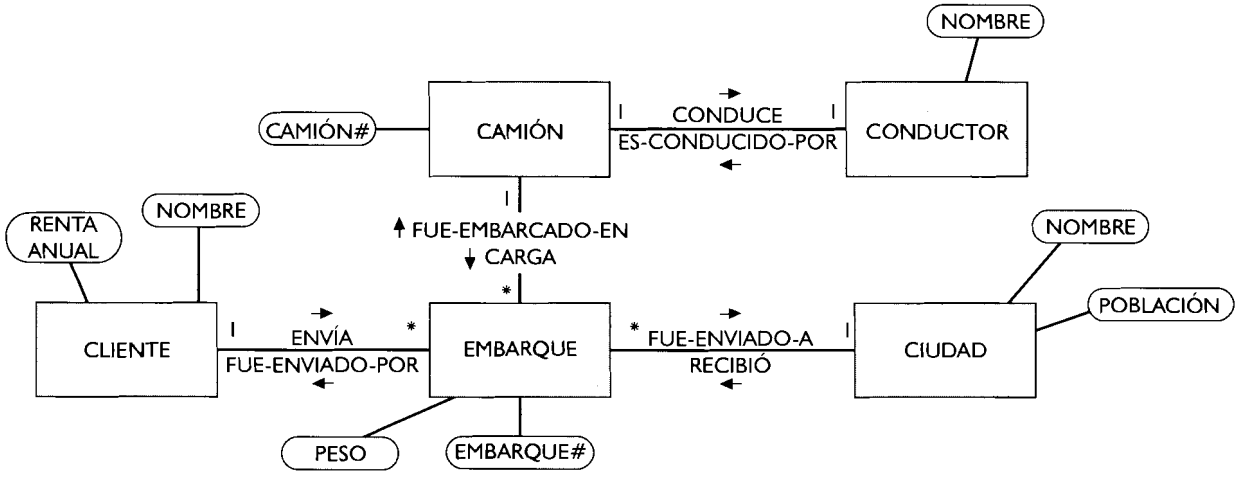
Parte A

1. Haga corresponder cada término con su definición:

—*clase derivada*	a. Una forma de combinar reglas, que va de la hipótesis a la conclusión.
—*objeto*	b. Una estructura lógica basada en objetos, predicados y funciones.
—*sistema de base de datos de objetos*	c. Sistema de software que gestiona el almacenamiento de datos y la manipulación de datos y reglas.
—*encadenamiento hacia atrás*	d. Una instancia de clase.
—*encadenamiento hacia adelante*	e. Una forma de combinar reglas, que va de la conclusión a la hipótesis.
—*lógica de primer orden*	f. Una clase que hereda características de otra clase.
—*sistema de gestión basado en el conocimiento*	g. Un sistema de software que gestiona el almacenamiento de datos y la manipulación de datos y objetos.

Parte B

1. Dibuje un fragmento de un modelo orientado a objetos que tenga dos conjuntos de objetos base: ESTUDIANTE y EMPLEADO. Luego muestre las especializaciones respectivas de esos conjuntos: INGENIERO y DIRECTOR. Haga que EMPLEADO tenga los atributos NÚMEMP y NOMBRE. Haga que ESTUDIANTE tenga los atributos NÚMEST y NOMBRE. INGENIERO tiene su propio atributo TIPO (ingmec, ingelec, etc.) y DIRECTOR tiene su propio atributo DPTO.



Siguiendo el ejemplo de este capítulo, haga corresponder su modelo orientado a objetos con clases en C++.

2. Usando el modelo de la Figura 14.1E, cree soluciones para cada una de las consultas siguientes en TextQuery:
   - **a.** Una lista de números de embarques para envíos que pesen más de 20 libras.
   - **b.** Nombre tres compañías con más de $10 millones de renta anual.
   - **c.** La población de Atlanta.
   - **d.** El conductor del camión #45.
   - **e.** El nombre y la población de las ciudades que hayan recibido embarques que pesen más de 100 libras.
   - **f.** El nombre y la renta anual de los clientes que han enviado embarques que pesen más de 100 libras.
   - **g.** Los números de los camiones que han transportado embarques de más de 100 libras.
   - **h.** Los nombres de los conductores que han transportado embarques que pesen más de 100 libras.
   - **i.** Ciudades que han recibido embarques de clientes que tengan más de $15 millones de renta anual.
   - **j.** Clientes que tengan $5 millones de renta anual y que hayan hecho envíos de menos de 1 libra.
   - **k.** Clientes que tengan más de $5 millones de renta anual que hayan hecho envíos de menos de 1 libra o que hayan hecho envíos a San Francisco.
   - **l.** Clientes que hayan hecho envíos transportados por el camión conducido por Jensen.
   - **m.** Conductores que han conducido embarques a clientes con renta anual superior a $20 millones a ciudades con población por encima del millón.
   - **n.** Clientes que hayan hecho envíos conducidos por cada conductor. (Idea: Para la soluciones TextQuery de los ejercicios n-q use la construcción SET-F y la comparación de conjuntos.)
   - **o.** Ciudades que hayan recibido embarques de todos los clientes.
   - **p.** Conductores que hayan transportado embarques para cada una de las ciudades.

```
madre(X, Y) :- progenitor(X, Y), femenino(X).
padre(X, Y) :- progenitor(X, Y), masculino(X).
hija(X, Y) :- progenitor(Y, X), femenino(X).
hijo(X, Y) :- progenitor(Y, X), masculino(X).
hermanos(X, Y) :- padre(Z, X), padre(Z, Y),
                  madre(W, X), Madre(W, Y), (X Y).
hermano(X, Y) :- hermanos(X, Y), masculino(X).
hermana(X, Y) :- hermanos(X, Y), femenino(X).
tio(X, Y) :- hermano(X, Z), progenitor(Z, Y).
tio(X, Y) :- progenitor(Z, X), hermanos(W, Z).
             casado(W, X), masculino(X).

casado(alan, belinda).          progenitor(felicia, john).
masculino(alan).                progenitor(felicia, mary).
femenino(belinda).              progenitor(neva, darlene).
masculino(charles).
femenino(darlene).
masculino(edward).
progenitor(alan, charles).
progenitor(alan, darlene).
progenitor(alan, edward).
casado(charles, felicia).
casado(edward, grace).
femenino(grace).
femenino(felicia).
```

q. Clientes que hayan hecho envíos a cada ciudad con población sobre 500.000. (Idea: Crear primero el conjunto de ciudades con población por encima de los 500.000.)

r. Dar una lista de clientes y de las rentas anuales de aquellos clientes cuyas rentas anuales sea la máxima de los clientes en la base de datos.

s. Dar una lista de los clientes que tienen todos los embarques por encima de las 25 libras.

t. Dar una lista de los clientes que envían todos sus embarques a una única ciudad. (Nota: La ciudad puede o no puede ser la misma para cada uno de los clientes.)

3. Para cada una de las consultas siguientes:

i. Si la consulta se puede respouder usando la Figura 14.1E, dé un procedimiento paso a paso para encontrar el resultado.

ii. Si la consulta no se puede responder usando la Figura 14.1E, indique por qué.

a. ¿Quiénes son los hermanos de Alan?
b. ¿Quiénes son los hijos de Felicia?
c. ¿Quiénes son los padres de Darlene?
d. ¿Quiénes son los hijos de Alan?
e. ¿Quiénes son los hermanos de Edward?
f. ¿Es Charles hermano de Darlene?
g. ¿Es Grace hermana de Charles?
h. ¿Tiene alguna hermana Belinda?
i. ¿Quién es el tío de Belinda?
j. ¿Quiénes son los tíos de Grace?

4. Para cada una de las consultas en (2) que no pueda responderse a partir de la Figura 14.1E modifique la base de datos para poder satisfacer dichas consultas.

5. Modifique las reglas de la base de datos y añada los hechos necesarios para determinar la interrelación de abuelo.

6. Escriba consultas en PROLOG para los siguientes problemas. Use la Figura 14.18.
   a. ¿Qué vendedores snministran partes con niveles de calidad mayores que 8?
   b. ¿Qné vendedores suministran cerrojos?
   c. ¿Qué vendedores suministran pinzas?
   d. ¿Qué partes están hechas de acero?
   e. ¿Cuáles son las localidades de los vendedores que suministran estribos de hierro?
   f. ¿Qué vendedores snministran partes de latón?
   g. ¿Qué vendedores suministran partes de grado aa?
   h. ¿Qné partes suministra James?
   i. ¿Cuáles son los niveles de calidad de las partes suministradas por el vendedor w?
   j. ¿Qué partes son suministradas por vendedores localizados en LA?

1. Haga alguna investigación bibliográfica sobre los SGBDOs. Escriba un ensayo que se centre en las similitudes y diferencias en estos productos.

2. Haga una investigación bibliográfica sobre los sistemas basados en el conocimiento. Escriba un ensayo sobre qué representación del conocimieuto (lógica, redes semánticas o armazones —*frames*— es más apropiada para construir los futuros SBCs). Explique las razones de su conclusión.

3. Genere una pequeña base de datos para una empresa de ventas hipotética. Muestre cómo extendiendo ésta a una base de conocimientos se mejoraría la toma de decisiones de la empresa.

PARTE

SEIS

EL LEGADO DE LOS SISTEMAS DE BASE DE DATOS

Los primeros sistemas de bases de datos han tenido un amplio uso comercial. Aunque no tan elegantes como los sistemas relacionales, orientados a objetos y las bases de conocimientos, dichos sistemas se siguen utilizando en la actualidad, tienen capacidades poderosas para muchas aplicaciones y son dignos de estudio.

En el Capítulo 15 se presenta la definición del modelo en red. Se define e ilustra la representación del modelo conceptual y se introduce y se muestra el lenguaje de manipulación de los datos del modelo en red.

El Capítulo 16 presenta el modelo jerárquico. Examina la representación del modelo conceptual de manera similar a como se hizo con el modelo en red. Además, se presenta el lenguaje de manipulación de los datos del IMS.

C A P Í T U L O

15

EL MODELO DE DATOS EN RED

Antecedentes históricos
Conceptos básicos y definiciones
- Estructura de tres niveles
- Registros y conjuntos

Interrelación del modelo en red con la semántica del modelado conceptual
- Transformación de conjuntos de objetos e interrelaciones uno-muchos
- Transformación de interrelaciones n-arias
- Transformación de interrelaciones muchos-muchos

Lenguaje de definición de los datos (DDL) del DBTG
- Del modelo de datos al esquema
- Del esquema al subesquema

Lenguaje de manipulación de los datos (DML) del DBTG
- Facilidades de recuperación y actualización del DBTG
  - Ejemplo 1: Recuperación simple de un registro
  - Ejemplo 2: Recuperación de los registros con una característica particular
  - Ejemplo 3: Eliminación de registros utilizando la orden ERASE
  - Ejemplo 4: Modificación del contenido de un registro
  - Ejemplo 5: Adición de un nuevo registro a la base de datos
- Facilidades del DBTG para el procesamiento de conjuntos
- Operaciones de conjuntos
  - Ejemplo 6: Inclusión de un registro en un conjunto
  - Ejemplo 7: Anulación de un registro de un conjunto
  - Ejemplo 8: Modificación de la asociación de un conjunto
- Clasificación
  - Ejemplo 9: Estado de la inserción y de la retención en un conjunto

IDMS/R-Un SGBD del DBTG
Evaluación de CODASYL DBTG
- Representación de los datos
- Lenguaje de manipulación de los datos
- Restricciones de integridad
- Implementación

Resumen
Preguntas de repaso
Problemas y ejercicios
Proyectos y cuestiones profesionales

Rita Minkowski, la directora de los sistemas de información de la Corporación Zeus, asistía a la reunión mensual de la sección local de la Asociation for Computing Machinery (ACM). El tema de esa tarde era "Los sistemas de bases de datos relacionales versus los sistemas de bases de datos jerárquicos».

John Stiller, el administrador regional de ventas para un SGBD en red popular, comenzó: "Se puede esperar que los sistemas relacionales trabajen bien en cuanto a la satisfacción de las necesidades de consultas ad hoc *por parte de los usuarios, pero nunca pueden lograr el rendimiento necesario para los sistemas de producción. Sin embargo, la mayoría de los sistemas de bases de datos se convierten en una mezcla de transacciones de producción —tales como la actualización de inventarios y el envío de facturas— y consultas* ad hoc*, con énfasis en los sistemas de producción. Desafortunadamente, los sistemas de producción altamente eficientes y las consultas* ad hoc *son actividades mutuamente excluyentes. Por eso nosotros continuamos soportando el modelo en red. Este modelo apoya las necesidades de la mayoría de los usuarios con la mayor velocidad y eficiencia."*

El conferenciante siguiente fue Andrea Villa, profesora titular de una universidad local. «Los sistemas jerárquicos y en red se dedican típicamente casi por entero a actividades de producción porque es lo que hacen mejor.» Continuó: "No obstante, no proporcionan la flexibilidad necesaria para soportar completamente las consultas ad hoc*. Con un SGBD en red, se puede abarcar la información de las ventas por regiones, pero puede consumir más tiempo recuperar el tipo de resumen de los datos que necesita un gerente comercial. Aún más, existe una generación completa de directivos que quieren —y esperan— este tipo de capacidad. Esta es la razón por la cual tantos profesionales han apoyado el modelo relacional, que ofrece la flexibilidad necesaria para responder a la clase de consultas* ad hoc *que se plantean los directivos en cuanto a informaciones resumidas."*

En la refutación, Stiller citó su propia investigación, que mostró que los sistemas relacionales tienden a ser más lentos que los sistemas de bases de datos jerárquicos y en red, que han registrado récords de rendimiento. Señaló que los usuarios tendrían que emprender una renovación muy cara de su equipamiento y rediseñar su sistema para conseguir el mismo rendimiento con un modelo relacional.

Villa movió su cabeza. "Desde luego, algunos de los sistemas relacionales más recientes no pueden tener el mismo comportamiento estadístico en cuanto al rendimiento que algunos de los sistemas jerárquicos y en red ya establecidos durante mucho tiempo. Y, en situaciones en las que las estructuras de los datos y los patrones de las transacciones se conocen muy bien con antelación al diseño y a la implementación, los sistemas jerárquicos y en red pueden prepararse para producir rendimientos más impresionantes que algunos de los SGBDs relacionales." Preguntó: "¿Pero qué sucede si las estructuras de los datos cambian? ¿O si el gerente comercial quiere resúmenes de datos que no son parte de la estructura de los datos o de los patrones de las transacciones originales? Para obtener la información, se debe escribir un nuevo programa de aplicación o, inclusive, rediseñar parte del sistema de base de datos. Cualquiera de estas opciones consume mucho tiempo y es muy cara. Sin embargo, en el modelo relacional se puede extraer fácilmente esta información. Esto significa que es posible darle a los directivos la información que necesitan para fundamentar sus decisiones estratégicas. A largo plazo, esta capacidad hace que los sistemas relacionales sean más eficientes para un mayor número de usuarios."

Entonces la reunión se abrió a preguntas y respuestas. Lo que dio como resultado una rica discusión que sugirió que hoy día las bases de datos jerárquicas y en red están vivas y gozan de buena salud, pero que, quizás, no son los modelos del futuro.

grafo orientado. Una estructura matemática en la que los puntos o nodos se conectan por aristas orientadas o arcos.

nodo. Parte de una estructura de retículo representada por un punto.

arista. Parte de una estructura de retículo representada por una arista orientada.

modelo de datos en red. Representa los datos en estructuras de retículo de tipos de registros conectados por interrelaciones uno-uno o uno-muchos.

Conference on Data Systems Languages (CODASYL). Una organización compuesta por representantes de vendedores de hardware, vendedores de software y usuarios; conocida principalmente por el desarrollo del lenguaje COBOL.

Database Task Group (DBTG). Un subgrupo de CODASYL con la responsabilidad de desarrollar normas para los sistemas de gestión de bases de datos.

Integrated Data Store (IDS). Uno de los primeros sistemas de gestión de bases de datos; su arquitectura influyó grandemente en las recomendaciones del DBTG para un modelo de base de datos en red.

En este capítulo se presentan los fundamentos del modelo de datos en red. Después de leer este capítulo, debería ser capaz de:

- Describir la estructura de datos básica a partir de la cual se construye el modelo de datos en red.
- Explicar la terminología utilizada para describir el modelo de datos en red.
- Utilizar los métodos fundamentales para representar un modelo orientado a objetos en un modelo de datos en red.
- Explicar cómo se utiliza el lenguaje de descripción de los datos para implementar las estructuras de datos en redes.
- Describir cómo opera el lenguaje de manipulación de los datos del DBTG para recuperar y actualizar los datos.
- Evaluar el modelo CODASYL DBTG.

▼ Antecedentes históricos

Las redes constituyen una manera natural de representar las interrelaciones entre los objetos. Se utilizan ampliamente en las matemáticas, la investigación operativa, la química, la física, la sociología y otros campos. Como los objetos y sus interrelaciones constituyen maneras útiles de modelar muchos de los fenómenos que nos conciernen en los negocios, no es sorprendente que la arquitectura de retículos se aplique también a la organización de las bases de datos.

Generalmente, las redes se pueden representar mediante una estructura matemática llamada **grafo orientado**. Los grafos orientados tienen una estructura simple. Se construyen con puntos o **nodos** conectados por arcos orientados o **aristas**. Dentro del marco de los modelos de datos, los nodos pueden considerarse como tipos de registros de los datos y las aristas pueden considerarse como la representación de las interrelaciones uno-uno o uno-muchos. Así, el **modelo de datos en red** representa los datos en estructuras de retículos de los tipos de registros conectados por interrelaciones uno-uno o uno-muchos. La estructura del grafo facilita la representación simple de las interrelaciones jerárquicas (como los datos genealógicos), las interrelaciones de pertenencia (como el departamento al que se asigna un empleado) y muchas otras. Además, una vez que se ha establecido una interrelación entre dos objetos, la recuperación y la manipulación de los datos asociados puede realizarse eficientemente.

Como se explicará en breve, una jerarquía es un caso particular de un retículo. En correspondencia con ello, el modelo de datos jerárquico, que será discutido en el próximo capítulo, es un caso particular del modelo de datos en red. Si bien el modelo de datos jerárquico históricamente precede al modelo de datos en red, parece útil discutir primero la estructura de grafos del modelo en red que es más general. De este modo, en este capítulo se trata el modelo de datos en red y en el capítulo siguiente el modelo jerárquico.

La organización **Conference on Data Systems Languages (CODASYL)** [1], formada por representantes de los vendedores de hardware, de los vendedores de software y de los usuarios más importantes, inicialmente desarrolló y normalizó el lenguaje COBOL al comienzo de la década de los años sesenta. A finales de esta década, esta organización nombró un subgrupo denominado **Database Task Group (DBTG)** para desarrollar las normas para los sistemas de gestión de bases de datos. Este subgrupo DBTG estaba fuertemente influido por la arquitectura utilizada para el primero de los SGBDs, el **Integrated Data Store (IDS)**, desarrollado por General Electric. Esta influencia condujo a las recomendaciones para un modelo en red que aparecen publicadas en un informe preliminar en 1969.

[1] Conferencia sobre Lenguajes de Sistemas de Datos. Por ser un nombre establecido, se mantienen las siglas en inglés (N. del T.).

Este primer informe dio lugar a numerosas sugerencias para su perfeccionamiento y se publicó un informe oficial revisado en 1971 que se sometió a la consideración del American National Standards Institute (ANSI) para su posible adopción como una norma nacional para los SGBDs. La organización ANSI no realizó acción alguna al respecto y los informes modificados de 1978, 1981 y 1984 sucedieron al informe de 1971.

Sin embargo, el documento de 1971 permanece como la proposición fundamental del modelo en red, que se convirtió en el modelo DBTG de CODASYL. Ha servido de base para el desarrollo de los sistemas de gestión de bases de datos en redes por parte de varios vendedores. El IDS (Honeywell) y el IDMS (Computer Associates) son dos de las implementaciones comerciales mejor conocidas.

Aunque el modelo de datos en red cada vez más parece ceder el paso en el futuro al modelo de datos relacional como el SGBD a elegir, actualmente se utiliza con efectividad en numerosos sistemas de bases de datos.

▼ Conceptos básicos y definiciones

Estructura de tres niveles

esquema. La vista lógica de todos los datos y sus interrelaciones en la base de datos.

subesquema. Los subconjuntos del esquema que se definen por la vista del usuario de la base de datos.

El modelo en red DBTG corresponde a la arquitectura de tres niveles de una base de datos de la ANSI/SPARC (Capítulo 3), como se indica a continuación:

- El nivel conceptual (la vista lógica de todos los datos y sus interrelaciones en la base de datos) se denomina **esquema**.
- El nivel externo (las vistas de los usuarios de los datos necesarias para diversas aplicaciones) se denomina **subesquema**.
- El nivel interno (los detalles físicos de almacenamiento) está implícito en la implementación.

Registros y conjuntos

tipo de registro. Una colección de elementos de los datos lógicamente relacionados.

conjunto. En el modelo DBTG, una interrelación uno-muchos entre dos tipos de registros.

tipo de registro dueño. El tipo de registro en el lado "uno" de la interrelación uno-muchos de un conjunto DBTG.

tipo de registro miembro. El tipo de registro en el lado "muchos" de la interrelación uno-muchos de un conjunto DBTG.

Solamente hay dos estructuras de datos fundamentales en el modelo en red, los tipos de registros y los conjuntos. Los **tipos de registros** se definen de manera usual como colecciones de elementos de los datos lógicamente relacionados. Por ejemplo, el tipo de registro de un cliente podría incluir los elementos de los datos siguientes: ID_Cliente, Nombre, Dirección, Cantidad_Que_Debe y Fecha_De_Último_Pago. Observe que esta colección se identifica como un tipo de registro del cliente, de manera que el tipo de registro se especifica por un nombre. Todos los tipos de registros reciben nombres como CLIENTE, FACTURA, REPRESENTANTE_DE_VENTAS y así sucesivamente.

Un **conjunto** en el modelo DBTG expresa una interrelación uno-muchos (o uno-uno) entre dos tipos de registros. (Observe que ésta no es la definición matemática usual de conjunto, que es simplemente una colección de elementos.) Por ejemplo, un conjunto podría expresar la interrelación uno - muchos entre los registros de los clientes y sus facturas pendientes. En cada conjunto de un retículo, un tipo de registro es el dueño y el otro tipo de registro es el miembro. En el ejemplo recién planteado, el tipo de registro del cliente es el dueño y el tipo de registro de la factura es el miembro. La interrelación uno-muchos comprende la posibilidad de que cero, uno o muchos registros de factura puedan estar interrelacionados con el registro de un cliente dado. Esto es, en un momento determinado, un cliente puede tener, digamos, diez, una o ninguna facturas pendientes. El lado "muchos" realmente expresa una capacidad más que una restricción.

Por supuesto, hay situaciones en las que una interrelación es estrictamente uno-uno, tal como ocurre con un camión y su conductor, pero esto se maneja del mismo modo. Una vez que se determinan el **tipo de registro dueño** y el **tipo de registro miembro**, todas las definiciones anteriores son aplicables.

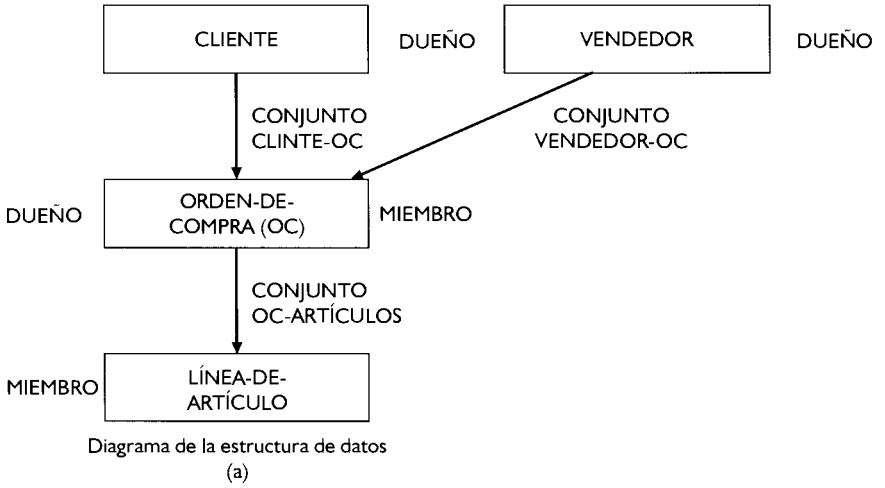
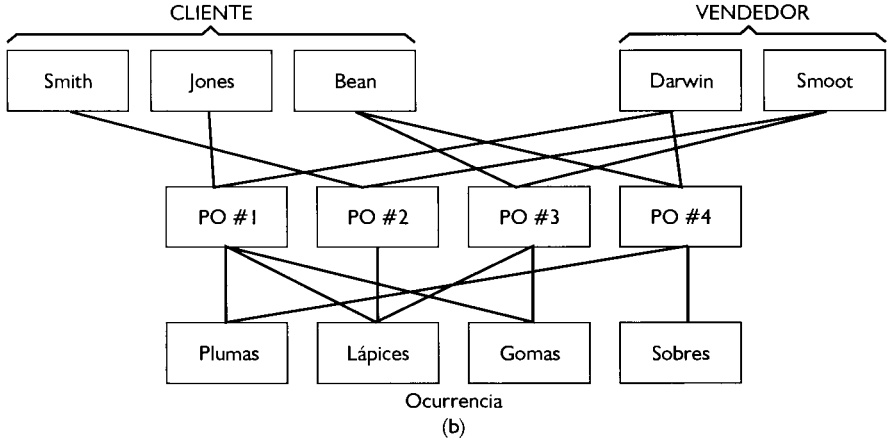


Diagrama de la estructura de datos
(a)



Ocurrencia
(b)

Estas convenciones se ilustran en el ejemplo mostrado en la Figura 15.1. La Figura 15.1(a) presenta la forma general de la estructura de datos. Ésta se denomina diagrama de Bachman en honor a Charles Bachman, quien cooperó en el desarrollo del sistema IDS en General Electric. La Figura 15.1(b) muestra los valores reales denominados **instancias** u **ocurrencias**, que pueden encontrarse en la estructura del diagrama de Bachman.

instancia. Valores actuales del registro expresados en una estructura de datos.

ocurrencia. Un sinónimo para instancia.

En la Figura 15.1(a) se observan ciertas convenciones representadas en los diagramas. Primero, los conjuntos se denotan por una arista entre los tipos de registros, con la flecha indicando el tipo de registro miembro (el lado "muchos" en la interrelación uno-muchos). Segundo, cada tipo de conjunto se construye a partir de un tipo de registro dueño, un tipo de registro miembro y un nombre para el tipo de conjunto. El nombre del conjunto es la etiqueta que se le asigna a la arista. Esto corresponde a un grafo en el que los nodos son los tipos de registros y las aristas se representan por líneas con flechas que conectan los tipos de registros. Tercero, la estructura de datos se construye a partir de las interrelaciones de los conjuntos simples. La Figura 15.1(a) muestra tres conjuntos: el conjunto CLIENTE_OC con CLIENTE como dueño y ORDEN_DE_COMPRA como miembro; el conjunto VENDEDOR_OC con VENDEDOR como dueño y ORDEN_DE_COMPRA como miembro; y el conjunto OC_ELEMENTO con ORDEN_DE_COMPRA como dueño y LÍNEA_DE_ARTÍCULOS como miembro.

La Figura 15.1(a) proporciona un ejemplo de la diferencia entre el modelo de datos en red y el modelo de datos jerárquico. Fíjese que ORDEN_DE_COMPRA es un tipo de registro miembro de dos conjuntos: CLIENTE_OC y VENDEDOR_OC. En el modelo jerárquico, ningún tipo de registro puede ser miembro de dos conjuntos diferentes. Sin embargo, *esto* se permite en el modelo de datos en red. Así, en la estructura de datos en red se garantiza un poder de representación adicional. Esta distinción es importante porque es una de las diferencias principales entre el modelo de datos en red y el modelo de datos jerárquico, que se discutirá en el capítulo siguiente.

Como el modelo en red permite únicamente las interrelaciones uno -muchos entre los tipos de registros, alguien se podría preguntar cómo se modela una interrelación muchos-muchos. La Figura 15.2 es un ejemplo clásico de una interrelación muchos-muchos. Un estudiante puede estar matriculado en muchos grupos y un grupo determinado puede tener muchos estudiantes. El modelo DBTG permite solamente **retículos simples** en las que todas las interrelaciones son uno-uno o uno-muchos. Un retículo **complejo**, que incluye una o varias interrelaciones muchos - muchos, no puede implementarse directamente en el modelo DBTG. Sin embargo, hay un método para transformar un retículo complejo, como la de la Figura 15.2, en un retículo simple, como requiere la implementación del DBTG.

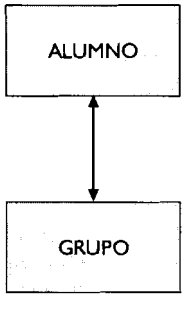
retículo simple. Una estructura de datos en la que todas las interrelaciones binarias son uno-muchos.

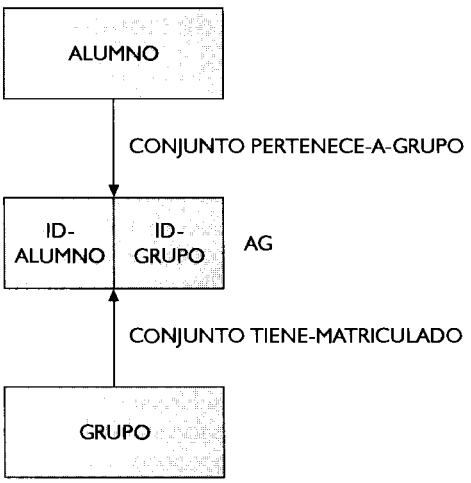
retículo complejo. Una estructura informática en que una o más interrelaciones binarias son muchos-muchos.

El método es similar al método mediante el cual los modelos conceptuales se transforman en tablas relacionales. Recordemos que cuando la cardinalidad entre dos conjuntos de objetos es muchos-muchos, se crea una tabla relacional que contiene los atributos claves de los dos conjuntos de objetos relacionados. Aquí se aplica un método similar. Cuando dos tipos de registros, como ALUMNO y GRUPO, se conectan en una interrelación muchos-muchos, se crea una intersección o **tipo de registro enlace** que consiste, por lo menos, en las claves de los registros ALUMNO y GRUPO. A discreción del diseñador pueden añadirse otros atributos.

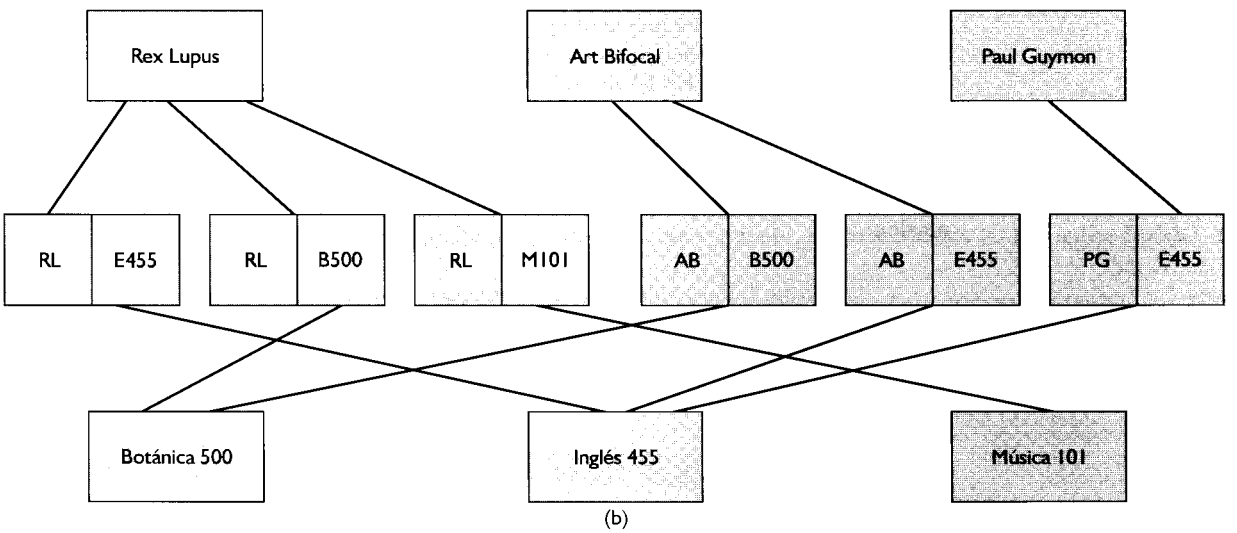
tipo de registro enlace. Un registro ficticio que se crea para convertir un retículo complejo en un retículo simple equivalente; también se denomina una intersección.

En la Figura 15.3 se ilustra este procedimiento. Se crea un tipo de registro enlace nombrado AG, que contiene los campos ID_ALUMNO e ID_GRUPO. Como se muestra en la Figura 15.3(a), el tipo de registro AG particiona la interrelación muchos-muchos en *dos* interrelaciones uno-muchos. Por consiguiente, el tipo de registro AG se convierte en miembro de dos conjuntos, el SET PERTENECE_AL_GRUPO y el SET TIENE_MATRICULADO. La Figura 15.3(b) ilustra cómo se crea una instancia del registro AG para cada par alumno/grupo. Por ejemplo, Rex Lupus posee tres registros AG, uno por cada grupo en los que él está matriculado. De manera similar, Botánica 500 posee dos registros AG, uno por cada estudiante matriculado. La necesidad de memoria y los requisitos de procesamiento adicionales son inherentes a la creación de estos registros artificiales, pero ahora el modelo de datos está en forma de retículo simple y satisface los requisitos del DBTG.





(a)



(b)

▼ Interrelación del modelo en red con la semántica del modelado conceptual

Como hemos considerado el enfoque del modelado conceptual de los datos para estudiar los sistemas de bases de datos, resulta apropiado examinar la interrelación entre el modelo de datos en red y el modelo de datos conceptual. En esta sección se muestra cómo un modelo de datos conceptual se puede transformar en un modelo en red.

Transformación de conjuntos de objetos e interrelaciones uno-muchos

registro lógico. Un tipo de registro visto desde la perspectiva del usuario.

enlace físico. Un medio de conectar los registros utilizando sus direcciones en disco.

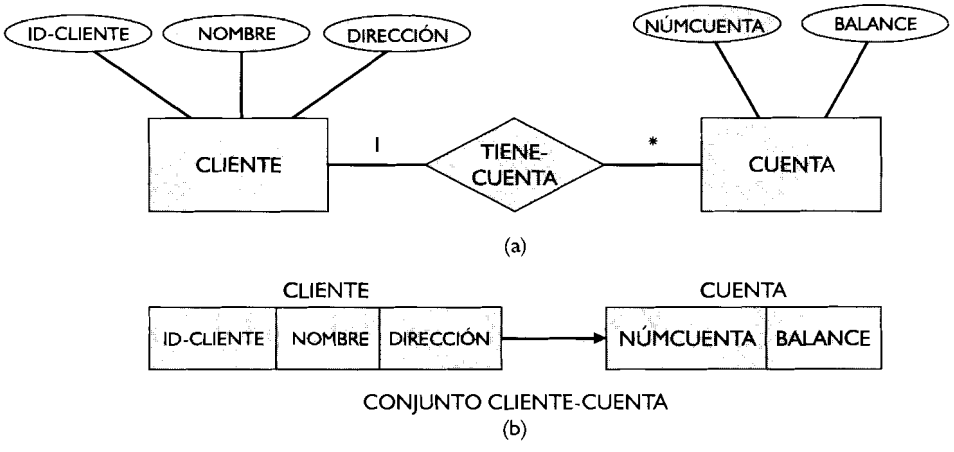
El modelo en red puede considerarse como un modelo de datos conceptual con todas las interrelaciones restringidas a conjuntos binarios (de dos objetos) e interrelaciones uno-muchos o uno-uno. Esto permite una representación gráfica de las estructuras de datos sin complicaciones. En lugar de los conjuntos de objetos del modelo de datos conceptual, se tienen **registros lógicos** que se conectan con otros registros lógicos mediante **enlaces físicos** que consisten en direcciones de los registros en el disco. Cada enlace representa exactamente una interrelación entre dos registros. Un conjunto es la interrelación entre dos tipos de registros conectados por un enlace binario.

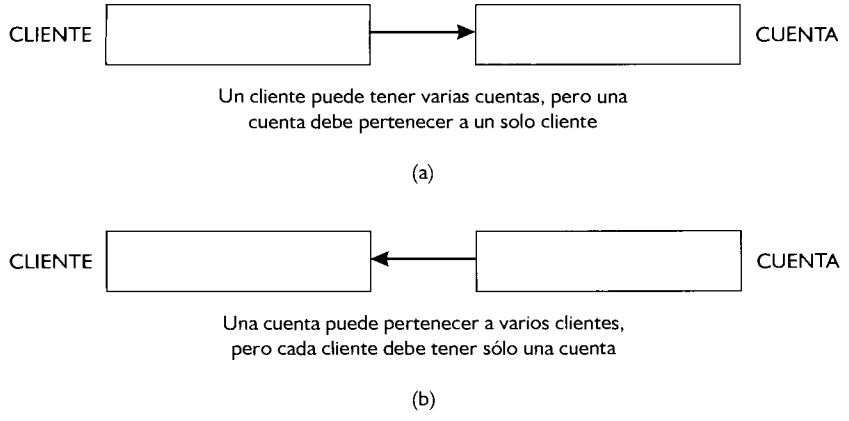
Para aclarar esto, veamos otro ejemplo de la compañía International Product Distribution (IPD). Consideremos los mismos hechos del caso anterior y agreguemos que la compañía IPD se organiza en los departamentos funcionales usuales, tales como contabilidad, comercialización y otros. La Figura 15.4(a) es un fragmento de un modelo de datos conceptual para la IPD que ilustra la interrelación entre los clientes y las cuentas de IPD. La Figura 15.4(b) muestra cómo el fragmento del modelo se transforma en una estructura de datos en red.

La correspondencia es bastante directa. Los nombres de los objetos se convierten en los nombres de los registros. Los atributos de los objetos se convierten en los campos de los registros. La interrelación entre los objetos se convierte en la interrelación entre los registros. Si la realidad es que un cliente puede tener muchas cuentas y que una cuenta puede pertenecer a un único cliente, como se indica en la Figura 15.4(a), entonces se añade una flecha en el extremo CUENTA del enlace TIENE_CUENTA (Figuras 15.4(b) y 15.5(a)). Si un cliente puede tener una única cuenta, pero esa cuenta puede pertenecer a varios clientes, entonces la flecha se añade en el extremo del cliente del enlace TIENE_CUENTA, como se muestra en la Figura 15.5(b).

Los ejemplos anteriores nos conducen a la expresión de estas reglas de transformación:

> *Regla 1*: Para cada conjunto de objetos O en un esquema conceptual, crear un tipo de registro R en la estructura de datos en red. Todos los atributos de O se representan como campos de R.

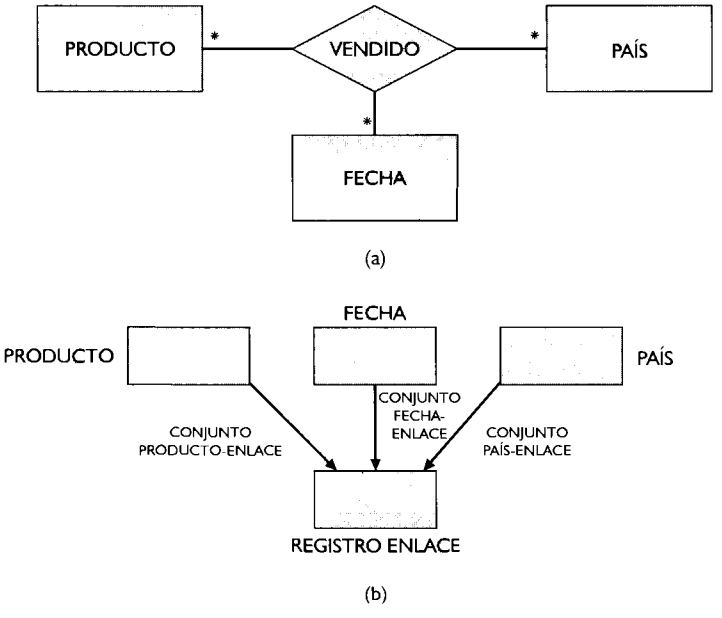




Regla 2: Para interrelaciones uno-muchos, el tipo de registro en el lado "uno" de la interrelación se convierte en el dueño, y el tipo de registro en el lado "muchos" de la interrelación se convierte en el tipo de registro miembro. Si la cardinalidad de una interrelación es estrictamente uno-uno, entonces los tipos de registros dueño y miembro pueden elegirse arbitrariamente.

Transformación de interrelaciones *n*-arias

La compañía IPD también tiene algunas interrelaciones ternarias, como se muestra en la Figura 15.6(a), que no satisfacen el requisito de interrelación binaria. Sin embargo, existe un camino sencillo para satisfacer ese requisito, como se ilustra en la Figura 15.6(b). Se



crea un registro enlace, que, al menos, debe estar compuesto por el campo clave de cada conjunto de objetos. La creación de este registro enlace permite la representación de la estructura del modelo conceptual en forma de retículo. Ahora las tres interrelaciones son binarias.

Observe que la interrelación entre los registros existentes y el registro enlace es uno-muchos, con el registro enlace siempre como registro miembro en el conjunto. Las ocurrencias se muestran en la Figura 15.7. La estrategia de crear tipos de registros enlace puede extenderse sin modificación para la transformación de las interrelaciones n-arias a la forma requerida. La tercera regla de transformación se expresa como:

> *Regla 3:* Para cada interrelación n-aria, $n > 2$, crear un registro enlace L y convertirlo en el tipo de registro miembro de n tipos de conjuntos. Designar el dueño de cada conjunto como el tipo de registro en el lado "uno" de las interrelaciones uno - muchos resultantes.

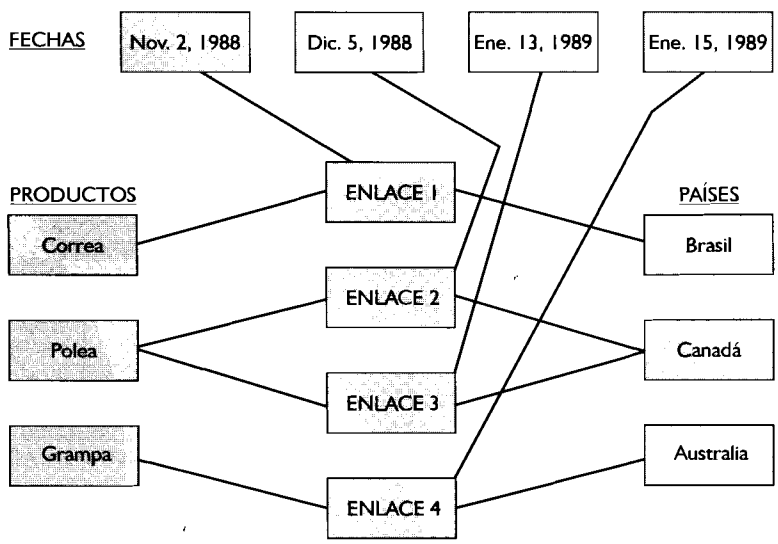
Transformación de interrelaciones muchos-muchos

Como se vio en el ejemplo anterior ALUMNO-GRUPO, una situación similar podría surgir cuando aparecen interrelaciones muchos-muchos. Como otro ejemplo, consideremos una situación que existe en la compañía IPD. Un fabricante puede producir muchos productos y cualquiera de estos productos puede ser elaborado por varios fabricantes. En la Figura 15.8(a) se muestra un diagrama de esta situación. La implementación requiere que se cree un registro enlace, que se nombra REG_ENLACE. Esto se muestra en la Figura 15.8(b). La interrelación entre PRODUCTO y REG_ENLACE es uno - muchos, así como la interrelación entre FABRICANTE y REG_ENLACE. Se satisfacen los requisitos del modelo en red DBTG.

En la Figura 15.9 se presenta un ejemplo de este modelo de datos. El fabricante Smith suministra los productos 115 y 116. El producto 115 también se suministra por Shirdlu, Inc. ¿Puede ver qué sucedería si Joe Bean Mfg. también suministra el producto 116, o si Shirdlu comienza a fabricar el producto 120 o un nuevo producto 135?

La última regla se refiere a esta situación muchos-muchos.

> *Regla 4:* Para cada interrelación muchos - muchos entre los conjuntos de objetos O_1 y O_2, crear un tipo de registro enlace L y convertirlo en el tipo de registro miembro de los dos tipos de conjuntos, en los cuales los dueños de los tipos de conjuntos son los tipos de registros correspondientes a O_1 y O_2.



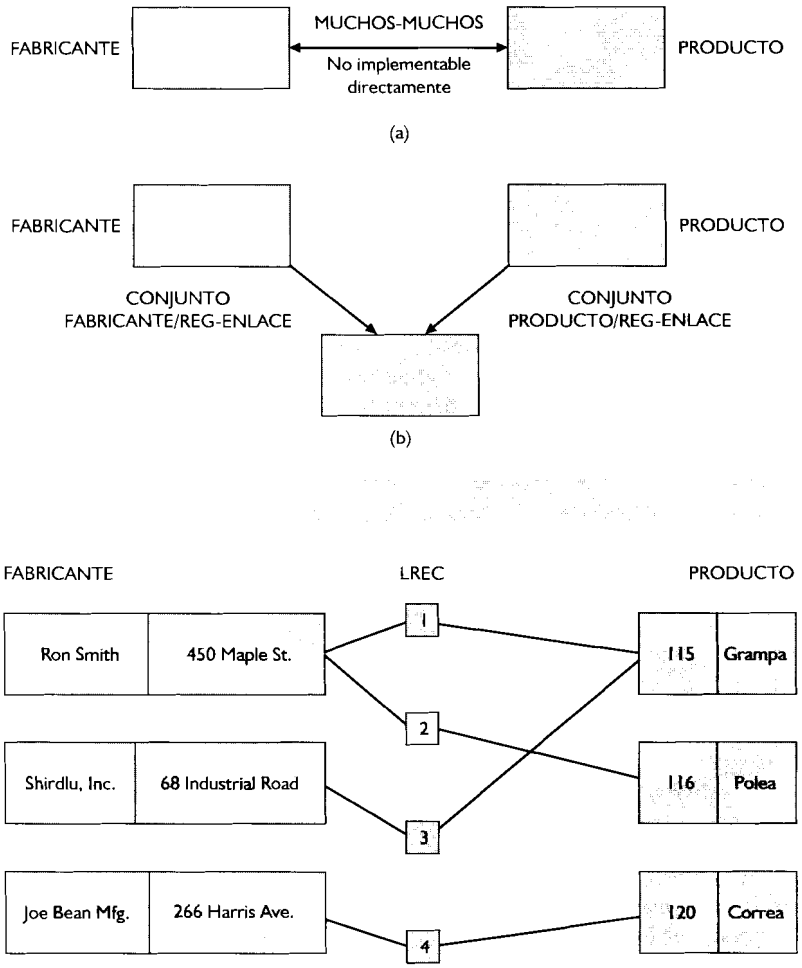


FIGURA 15.9 EJEMPLO DE USO DE UN REGISTRO ENLACE PARA CONVERTIR INTERRELACIONES MUCHOS-MUCHOS A INTERRELACIONES UNO-UNO

▼ Lenguaje de definición de los datos (DDL) de DBTG

Analicemos, ahora, los lenguajes mediante los cuales se implementa el modelo de datos en red. Estos lenguajes constituyen los medios mediante los que se especifica la estructura de datos o el esquema y mediante los que se almacenan y manipulan los datos. El lenguaje utilizado para especificar el esquema se denomina **lenguaje de definición de los datos (Data Definition Language-DDL)** y el lenguaje utilizado para almacenar y manipular los datos se denomina **lenguaje de manipulación de los datos (Data Manipulation Language-DML)**. En esta sección se presenta el DDL DBTG y en la siguiente el DML DBTG.

lenguaje de definición de los datos (DDL). El lenguaje utilizado para especificar un esquema de la base de datos.

lenguaje de manipulación de los datos (DML). El lenguaje utilizado para almacenar y manipular los datos.

Del modelo de datos al esquema

Como se planteó anteriormente, la definición general de una base de datos en red —sus registros y conjuntos— se denomina esquema. En particular, el esquema describe las interrelaciones entre los tipos de registros, identifica los elementos de los datos que constituyen cada tipo de registro y define los tipos de registros dueño-miembro que determinan los

tipos de conjuntos. A continuación se indica un procedimiento para la utilización del DDL en la definición de un esquema:

- Crear el modelo de datos conceptual.
- Transformar el modelo de datos conceptual en los diagramas de la estructura de datos en red.
  - Verificar si existen interrelaciones uno-muchos entre los tipos de registros. Pueden implementarse directamente como conjuntos DBTG.
  - Si existen interrelaciones muchos-muchos, transformarlas en dos conjuntos que tengan interrelaciones uno-muchos, construyendo los registros enlaces necesarios.
  - Si hay interrelaciones *n*-arias, convertirlas en interrelaciones binarias mediante el método ilustrado previamente.
- Utilizar el DDL para implementar el esquema.

Un esquema está compuesto por los componentes siguientes:

sección del esquema. La sección del esquema del DBTG que nombra el esquema.

sección de registros. La sección del esquema del DBTG que define cada registro, sus elementos de los datos y su localización.

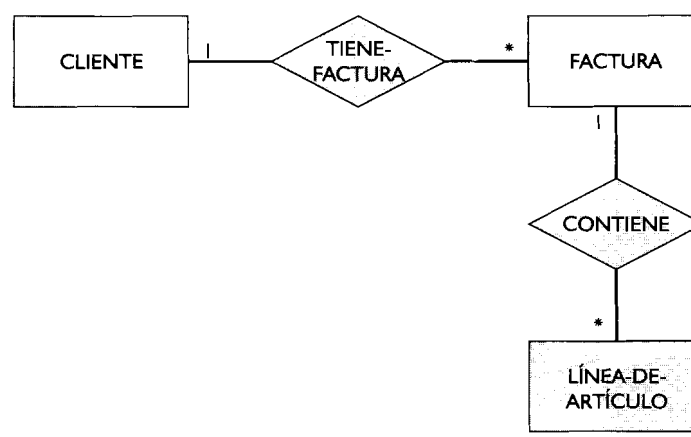
sección de conjuntos. La sección del esquema del DBTG que define los conjuntos e incluye los tipos de registros dueños y los tipos de registros miembros.

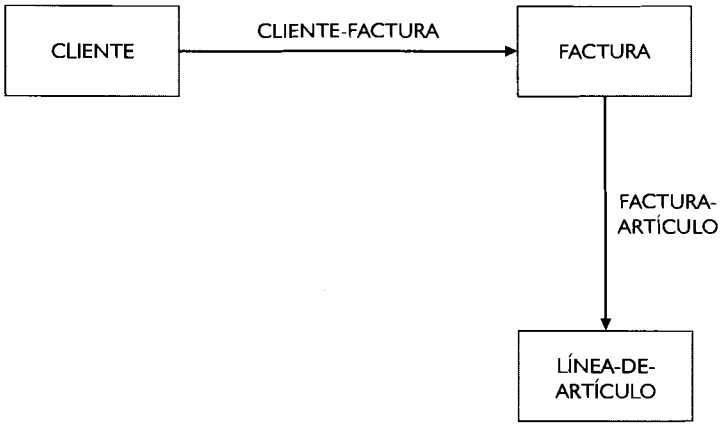
- Una **sección del esquema**, que nombra el esquema.
- **Secciones de registros**, que proporcionan las especificaciones de cada estructura de registro, sus elementos de los datos y su localización.
- **Secciones de conjuntos**, que especifican todos los conjuntos, incluyendo los tipos de registros dueños y miembros.

Considere el modelo de datos conceptual en la Figura 15.10. La compañía IPD desea implementar el esquema correspondiente. En la Figura 15.11 se muestra la transformación en la estructura de datos en red. El uso del DDL para implementar el esquema se muestra en la Figura 15.12. Las líneas de código se numeran para la referencia en nuestra discusión.

La sección del esquema se representa en la línea 1. Esta línea identifica lo que sigue como una descripción del esquema para una base de datos nombrada REGISTRO_DE_CUENTAS. Este nombre se suministra por el usuario.

De la línea 2 hasta la línea 15 corresponde la sección de registros. Cada tipo de registro se identifica por el nombre: CLIENTE, FACTURA y LÍNEA_DE_ARTÍCULO. Para cada tipo de registro, se definen los elementos de los datos de los componentes. Para CLIENTE éstos son ID_CUST, NOMBRE, DIRECCIÓN y BALANCE_DE_CUENTA. Para cada elemento de los datos se da un tipo de datos y una especificación de longitud. Por ejemplo, a BALANCE_DE_CUENTA se le asigna el tipo de datos NUMERIC y una





longitud de cinco caracteres, con dos caracteres a la derecha del punto decimal. Aquéllos que se identifican como INTEGER tienen una longitud implícita.

Los conjuntos pueden especificarse cuando se hayan definido todos los registros. En la Figura 15.12, de la línea 16 hasta la línea 21 se muestran ejemplos para el esquema REGISTRO_DE_CUENTAS. La definición de un conjunto requiere por lo menos las tres líneas mostradas. La primera línea se utiliza para nombrar el tipo de conjunto, la segunda línea para nombrar el tipo de registro dueño y la tercera línea para nombrar el tipo de registro miembro. En la Figura 15.12, un conjunto se nombra CLIENTE_FACTURA y tiene a CLIENTE como su tipo de registro dueño y a FACTURA como su tipo de registro miembro.

```
 1. SCHEMA NAME IS REGISTRO_DE_CUENTAS
 2. RECORD NAME IS CLIENTE
 3. ID_CLIENTE         TYPE IS  NUMERIC INTEGER
 4. NOMBRE             TYPE IS  CHARACTER 15
 5. DIRECCIÓN          TYPE IS  CHARACTER 20
 6. BALANCE_DE_CUENTA  TYPE IS  NUMERIC (5,2)

 7. RECORD NAME IS FACTURA
 8. NÚM_FACTURA        TYPE IS  NUMERIC INTEGER
 9. FECHA              TYPE IS  CHARACTER 9
10. CANTIDAD           TYPE IS  NUMERIC (5,2)
11. ESTADO             TYPE IS  CHARACTER 2

12. RECORD NAME IS LINEA_DE_ARTÍCULO
13. NÚM_INVENTARIO     TYPE IS  NUMERIC INTEGER
14. DESCRIPCIÓN        TYPE IS  CHARACTER 20
15. PRECIO             TYPE IS  NUMERIC (4,2)

16. CLIENTE_FACTURA
17. OWNER IS CLIENTE
18. MEMBER IS FACTURA

19. FACTURA_ARTÍCULO
20. OWNER IS FACTURA
21. MEMBER IS ARTÍCULO
```

Del esquema al subesquema

Mientras que el esquema define la estructura lógica general de la base de datos, el subesquema describe la vista externa de un usuario o programa de aplicación. Los subesquemas básicamente son subconjuntos del esquema. Sin embargo, la independencia de los datos (para permitir la variedad de terminología de los usuarios) se garantiza permitiendo que difieran algunos de los elementos del esquema. Los elementos de los datos que no se agruparon en el esquema pueden agruparse; los elementos de los datos, los registros y los conjuntos se pueden renombrar; y el orden de las descripciones puede cambiarse.

No se ha aceptado una norma DBTG para los subesquemas; sin embargo, usualmente se utilizan las divisiones siguientes:

división de título. Esta porción del subesquema del DBTG permite nombrar el subesquema y su esquema asociado.

división de correspondencia. La porción del subesquema del DBTG que permite cambiar los nombres utilizados en el esquema por los nombres elegidos para el subesquema.

división de estructura. La división del subesquema del DBTG donde se definen los registros, los elementos de los datos y los conjuntos a partir del esquema.

sección de registros del subesquema. La sección de la división de estructura que especifica los registros, los elementos de los datos y los tipos de datos del subesquema.

sección de conjuntos del esquema. La sección de la división de estructura que define los conjuntos que se incluyen en el subesquema.

- Una **división de título,** que garantiza nombrar el subesquema y su esquema asociado.
- Una **división de correspondencia,** que garantiza los cambios en los nombres entre el esquema y el subesquema, si se desea.
- Una **división de estructura,** que especifica los registros, los elementos de los datos y los conjuntos del esquema que están presentes en el subesquema. Esta división se compone de secciones de registros y de conjuntos. La **sección de registros del subesquema** define los registros que se incluyen en el subesquema y los elementos de los datos de esos registros conjuntamente con sus tipos de datos. **La sección de conjuntos del subesquema** identifica los conjuntos que se incluyen en él.

Una aplicación de la compañía IPD que calcula el número de facturas pendientes por cliente, así como las cantidades adeudadas en esas facturas, se utiliza como un ejemplo. A partir del esquema REGISTRO_DE_CUENTAS, la aplicación únicamente requiere los tipos de registros CLIENTE y FACTURA y el conjunto CLIENTE_FACTURA. En la Figura 15.13 se muestra el subesquema.

El nombre del subesquema (SS) es ESTADO_DE_FACTURAS. La división de correspondencia muestra que el registro CLIENTE del esquema se ha renombrado DEUDOR y el conjunto CLIENTE_FACTURA se ha renombrado DEUDOR_FACTURA en el subesquema ESTADO_DE_FACTURAS. Esto se realiza en la sección ALIAS, donde AD denota "descripción del alias." El registro CLIENTE del esquema se ha renombrado con el alias DEUDOR en el subesquema y se han incluido justamente tres de los cuatro elementos de los datos que comprenden el registro CLIENTE en el esquema. La instrucción

```
SS ESTADO_DE_FACTURAS WITHIN REGISTRO_DE_CUENTAS.
MAPPING DIVISION.
ALIAS SECTION.
AD RECORD CLIENTE IS DEUDOR.
AD SET CLIENTE_FACTURA IS DEUDOR_FACTURA.
STRUCTURE DIVISION.
RECORD SECTION.
01       DEUDOR.
         05       ID_CLIENTE.
         05       NOMBRE.
         05       BALANCE_DE_CUENTA.
01       FACTURA ALL.
SET SECTION.
SD DEUDOR_FACTURA.
```

órdenes de navegación. Órdenes del DML DBTG utilizados para encontrar los registros de la base de datos.

órdenes de recuperación. Órdenes del DML DBTG utilizados para recuperar los registros de la base de datos.

órdenes de actualización de registros. Órdenes del DML DBTG utilizadas para cambiar los valores de los registros.

órdenes de actualización de conjuntos. Órdenes del DML DBTG utilizadas para crear, cambiar o eliminar las instancias de los conjuntos.

área de trabajo del usuario (UWA). Porción de la memoria principal utilizada para retener las variables del programa que apuntan a o retienen los registros de diferentes tipos mientras sus contenidos estén siendo utilizados por el programa anfitrión.

indicadores actuales. Marcadores de la ubicación de los registros que se han encontrado.

banderas de estados. Variables utilizadas para denotar el éxito o el fallo de la última operación ejecutada por el programa de aplicación.

plantillas de los registros. Formatos utilizados para los registros que se leen en el UWA.

FACTURA ALL indica que se han incluido todos los elementos de los datos del registro FACTURA en el esquema, por lo que no hay necesidad de especificarlos nuevamente.

Como se puede ver, el subesquema permite al usuario utilizar un esquema predefinido para adaptarlo a los requisitos de una aplicación particular.

▼ Lenguaje de manipulación de los datos (DML) del DBTG

El lenguaje de manipulación de los datos (DML) del DBTG proporciona órdenes poderosas para manipular un sistema de base de datos en red. Esto es, los medios a través de los cuales pueden utilizarse los datos contenidos en la base de datos para soportar los requisitos de información de la organización. Una vez que se diseña la base de datos y se crea utilizando el DDL, el DML permite a los usuarios ejecutar operaciones sobre la base de datos con el propósito de proporcionar información e informes, así como también actualizar y modificar el contenido de los registros.

A diferencia de los DMLs de los sistemas de bases de datos relacionales, cuyos operadores procesan relaciones completas de una vez, los operadores del DML DBTG procesan los registros de uno en uno. Además, el DML tiene que estar inmerso en un lenguaje anfitrión, tal como COBOL. Las órdenes básicas utilizadas por el DML pueden clasificarse en **órdenes de navegación**, **órdenes de recuperación**, **órdenes de actualización de registros** y **órdenes de actualización de conjuntos**. En la Figura 15.14 se presentan las órdenes de cada uno de estos grupos.

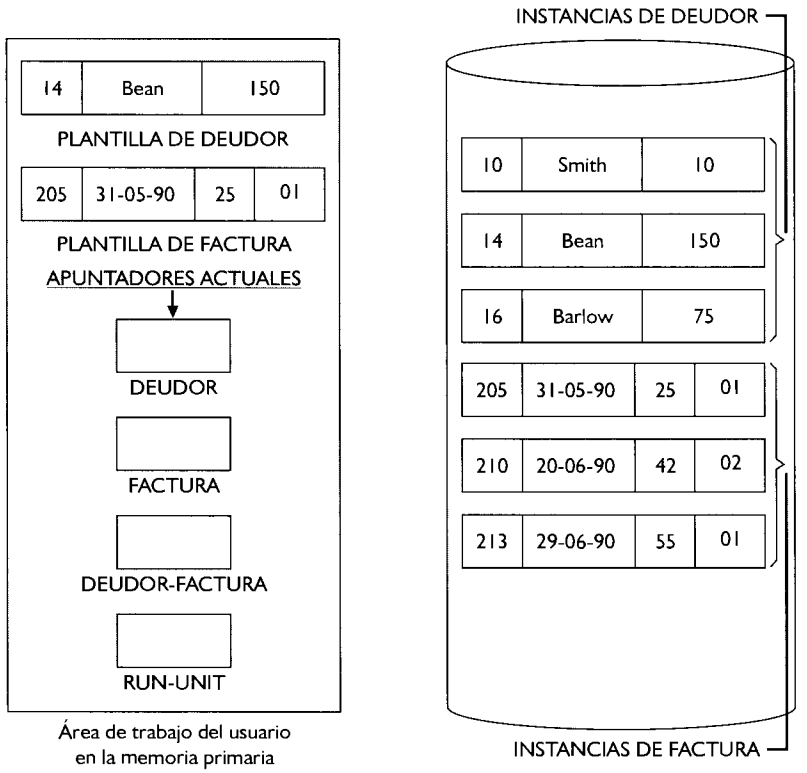
Antes de ilustrar el DML DBTG con ejemplos, se necesita definir los términos que se requerirán para la discusión.

El primer concepto es el de **área de trabajo del usuario (User Working Area-UWA)**. Cada usuario o programa de aplicación tiene un UWA. Los registros en el subesquema se almacenan en el UWA, conjuntamente con los **indicadores actuales** y los **indicadores de estados**. La forma en que el registro se estructura en el UWA se denomina **plantilla del registro**.

Consideremos la Figura 15.15. Aquí se muestra un área de trabajo posible correspondiente al subesquema definido en la Figura 15.13. El puntero actual DEUDOR indica la localización del último registro procesado de ese tipo de registro. El puntero actual FACTURA indica la localización del último registro procesado de ese tipo de registro. El puntero actual DEUDOR_FACTURA indica la localización del último registro asociado a ese conjunto al que se ha tenido acceso, sea dueño o miembro. El puntero actnal RUN_UNIT indica la localización del último registro de cualquier tipo al que se ha tenido acceso.

Se debe observar que esos indicadores actuales funcionan como marcadores de localización. Cuando el usuario emite una orden FIND (pendiente de discusión), se encuentra el registro y se marca su posición en el indicador actual correspondiente. Cuando se emite

Tipo	Orden
Navegación	FIND
Recuperación	GET
Actualización de registros	ERASE
	STORE
	MODIFY
Actualización de conjuntos	CONNECT
	DISCONNECT
	RECONNECT



un segundo orden, el SGBD se remite al indicador actual para determinar con qué registro trabajar. Brevemente, el tipo y la función de los punteros actuales son los siguientes:

- *Indicador de la unidad de ejecución*: La unidad de ejecución (Run Unit) se refiere al programa del usuario. El indicador de la unidad de ejecución contiene la dirección del registro, o de la instancia de un conjunto, a la cual el programa de aplicación ha tenido acceso más recientemente. En la Figura 15.15 es la factura 205.
- *Indicador de tipo de registro*: Hay un indicador actual para cada tipo de registro en el subesquema. Para cada tipo de registro, el UWA contiene la dirección del registro de ese tipo al cual el programa tuvo acceso más recientemente. En la Figura 15.15 se tienen los indicadores actuales para los tipos de registros DEUDOR y FACTURA y apuntan a los registros que corresponden a Bean y a la factura 205, respectivamente.
- *Indicador de tipo de conjunto*: Un puntero actual contiene la dirección del registro de un tipo determinado de conjunto al cual se tuvo acceso más recientemente. Para cada tipo de conjunto se mantiene por separado un puntero actual. El puntero puede señalar un registro dueño o miembro, dependiendo de a cuál se tuvo acceso más recientemente. La Figura 15.15 muestra el indicador actual para el único conjunto en este subesquema, DEUDOR_FACTURA, y señala un registro miembro, la factura 205.

Los punteros precedentes se actualizan automáticamente en la medida que ocurren los accesos al sistema de base de datos. Es instructivo pensar en los indicadores actuales

como variables en una tabla. La Figura 15.15 ilustra los indicadores actuales posibles para el subesquema ESTADO_DE_FACTURAS.

Las banderas de estados constituyen un conjunto de variables utilizado para comunicar el resultado de la última operación ejecutada en el programa de aplicación. La bandera más frecuentemente utilizada es *db-status*. Su valor se iguala a "0" si la operación más reciente tuvo éxito; de lo contrario, se le asigna un código de error. El uso más común de esta bandera es señalar el final de los datos. Entre otras banderas de estados se incluyen *db-set-name*, *db-record-name* y *db-data-name*. Estas banderas toman valores apropiados cuando la última operación falla, como un medio para identificar la fuente de un problema. A continuación se presentarán ejemplos de su uso en las consultas.

Facilidades de recuperación y actualización del DBTG

Las órdenes FIND seleccionan y localizan un registro deseado o una instancia determinada de un conjunto. Esta es la razón por la cual se denominan órdenes de navegación. Después, tiene que utilizarse una orden GET para recuperar realmente los datos. La orden FIND también podría estar seguida de una orden de actualización, tales como ERASE o MODIFY.

Sintácticamente, hay dos formas de la orden FIND, como se muestra a continuación. Las partes opcionales de las órdenes se denotan mediante corchetes ([...]) y los nombres que se suministran por el usuario se indican por corchetes angulares (<...>).

```
FIND ANY <nombre de registro> [USING <lista de campos>]
FIND DUPLICATE <nombre de registro > [USING <lista de campos>]
```

Veamos varios ejemplos.

Ejemplo 1: Recuperación simple de un registro. Supongamos que la compañía IPD quiere la información del Cliente 105. Se aplican las órdenes siguientes:

```
MOVE 105 TO ID_CLIENTE IN CLIENTE
FIND ANY CLIENTE USING ID_CLIENTE
GET CLIENTE
```

La orden FIND provoca que el indicador de la unidad de ejecución, el indicador CLIENTE y el indicador CLIENTE_FACTURA señalen el registro del Cliente 105. Entonces la orden GET trae el registro a la plantilla CLIENTE en el UWA.

Ejemplo 2: Recuperación de los registros con una característica particular. Supongamos ahora que la compañía IPD quiere todos los registros de los clientes que tengan los balances de sus cuentas iguales a cero.

```
MOVE 0 TO BALANCE_DE_CUENTA IN CLIENTE
FIND ANY CLIENTE USING BALANCE_DE_CUENTA
DOWHILE DB_STATUS = 0
     GET CLIENTE
     (procesamiento de un registro de cliente)
     FIND DUPLICATE CLIENTE USING BALANCE_DE_CUENTA
END_DO
```

En este ejemplo se tiene un lazo controlado por DB_STATUS. Esto es, DB_STATUS tomará un valor de código diferente de cero cuando no haya más cuentas de CLIENTE con balances iguales a cero. El primer FIND localiza el primer registro de CLIENTE cuyo balance sea cero. El siguiente FIND busca un DUPLICADO (DUPLICATE), que se refiere a otro registro que tenga el mismo valor de BALANCE_DE_CUENTA que el que tiene el CLIENTE actual.

Ejemplo 3: Eliminación de registros utilizando la orden ERASE. La compañía IPD quiere eliminar las cuentas de CLIENTE que tengan un BALANCE_DE_CUENTA igual a cero.

```
MOVE 0 TO BALANCE_DE_CUENTA IN CLIENTE
FIND FOR UPDATE ANY CLIENTE USING BALANCE_DE_CUENTA
DOWHILE DB_STATUS = 0
     ERASE CLIENTE
     FIND FOR UPDATE DUPLICATE CLIENTE USING BALANCE_DE_CUENTA
END_DO
```

Comparando esta consulta con la del Ejemplo 2, puede verse que la orden ERASE se utiliza en lugar del GET. Además, las palabras FOR UPDATE se añaden al orden FIND, lo que comunica al SGBD que ocurrirá una actualización, por lo que el registro tiene que bloquearse para la unidad de ejecución. El bloqueo del registro evita que otros usuarios lo actualicen mientras la unidad de ejecución lo procesa. FOR UPDATE se requiere cuando los registros se solicitan para operaciones diferentes de GET.

Volvamos ahora a un ejemplo de modificación del contenido de los registros almacenados.

Ejemplo 4: Modificación del contenido de un registro. Como se indicó anteriormente, una forma especial de la orden FIND —FIND FOR UPDATE— se utiliza para modificar los contenidos de los registros. FIND FOR UPDATE se utiliza para localizar el registro; GET se utiliza para trasladarlo al UWA, donde se realizan los cambios. Entonces, la orden MODIFY se utiliza para reemplazarlo en la base de datos.

Supongamos que el Cliente 502 de la IPD acaba de trasladar sus oficinas y necesita cambiar el contenido del campo de la dirección en su registro. El DML podría utilizarse de la manera siguiente:

```
MOVE 502 TO ID_CLIENTE IN CLIENTE
FIND FOR UPDATE ANY CLIENTE USING ID_CLIENTE
GET CLIENTE
IF DB_STATUS = 0
      THEN MOVE '455 Cherry Lane, San Marino, CA' TO
DIRECCIÓN IN CLIENTE
             MODIFY DIRECCIÓN
      ELSE (ejecutar una rutina de error)
END_IF
```

En este ejemplo, la instrucción MODIFY se utiliza para identificar el campo del registro que será cambiado. Si no se suministra algún identificador de un campo del registro, el SGBD supone que se cambiará el registro completo.

Ejemplo 5: Adición de un nuevo registro a la base de datos. La orden STORE se utiliza para insertar nuevos registros en la base de datos. El registro se construye en el UWA y, entonces, se introduce en la base de datos mediante STORE.

La compañía IPD tiene un nuevo registro de cliente por añadir. El nombre del nuevo cliente es Harry Z. Smith, localizado en 201 S. Main, San Marino, California. La compañía de Smith acaba de comprar mercancías por un valor de $500.00. IPD ha asignado al cliente el número de identificación 503.

```
MOVE 503 TO ID_CLIENTE IN CLIENTE
MOVE 'HARRY Z. SMITH' TO NOMBRE IN CLIENTE
MOVE '201 S. MAIN, SAN MARINO, CA' TO DIRECCIÓN IN
CLIENTE
```

```
MOVE 500.00 TO BALANCE_DE_CUENTA IN CLIENTE
STORE CLIENTE
```

Facilidades del DBTG para el procesamiento de conjuntos

Los conjuntos se utilizan para procesar registros entre los que se establece una interrelación. El DML DBTG proporciona facilidades para insertar y anular registros en las instancias de los conjuntos, así como para trasladar registros dentro de las instancias de los conjuntos. También se cuenta con opciones disponibles para especificar las restricciones sobre los miembros de un conjunto. Seguidamente se explican estas facilidades.

Operaciones de conjuntos

El lenguaje DBTG proporciona tres órdenes para procesar conjuntos. CONNECT añade un registro al conjunto. DISCONNECT anula un registro del conjunto. RECONNECT permite cambiar los miembros de un conjunto. Para añadir un nuevo registro a una instancia particular de un conjunto, primero el registro tiene que añadirse a la base de datos. Entonces, los punteros actuales del tipo de registro y del tipo de conjunto tienen que indicar el registro y la instancia del conjunto apropiados. Veamos el ejemplo siguiente.

Ejemplo 6: Inclusión de un registro en un conjunto. Suponga que el Cliente 431 de la compañía IPD acaba de hacer una compra de $100.00 sobre su cuenta y que se ha preparado la factura asociada (#231). El Cliente 431 es ahora el propietario de una nueva instancia de factura. Tiene que crearse la ocurrencia del registro FACTURA y, después, se conecta al conjunto CLIENTE_FACTURA. Esto se realiza de la manera siguiente:

```
MOVE 231 TO NÚM_FACTURA IN FACTURA
MOVE '7/7/90' TO FECHA IN FACTURA
MOVE 100.00 TO CANTIDAD IN FACTURA
STORE FACTURA
MOVE 431 TO ID_CLIENTE IN CLIENTE
FIND ANY CLIENTE USING ID_CLIENTE
CONNECT FACTURA TO CLIENTE
```

Las primeras cuatro instrucciones crean el nuevo registro de factura y lo insertan en la base de datos, como ocurre en el Ejemplo 5. En este punto del procesamiento, el nuevo registro FACTURA es al que señalan el indicador actual de la unidad de ejecución, el actual de FACTURA y el actual de CLIENTE_FACTURA. La orden CONNECT conecta el registro actual de FACTURA a la instancia existente del conjunto. En este ejemplo, el nuevo registro FACTURA se conecta a la ocurrencia CLIENTE_FACTURA, cuyo dueño es el Cliente 431.

Así, después la orden STORE, 431 se traslada al elemento de los datos ID_CLIENTE y el registro del Cliente 431 se convierte en el actual de CLIENTE_FACTURA. Entonces, la orden CONNECT coloca el nuevo registro FACTURA en esta instancia del conjunto CLIENTE_FACTURA.

La orden DISCONNECT anula el registro actual de la unidad de ejecución de uno o más conjuntos. Esta operación no borra un registro de la base de datos, *únicamente anula un registro de un conjunto*. Si se desea la eliminación, ésta se completa utilizando la orden ERASE, como se ilustró previamente. En el ejemplo 7 se muestra el uso de la orden DISCONNECT.

Ejemplo 7: Anulación de un registro de un conjunto. Cuando se paga una factura de la compañía IPD, ésta se suprime del conjunto de facturas pendientes para ese cliente. Por ejemplo, supongamos que la Factura 254 de IPD acaba de ser pagada totalmente.

```
MOVE 254 TO NUM_FACTURA OF FACTURA
FIND ANY FACTURA USING NUM_FACTURA
DISCONNECT FACTURA FROM CLIENTE_FACTURA
```

Las primeras dos instrucciones localizan la factura deseada. La instrucción final la desconecta del conjunto del cual es actualmente un registro miembro. El registro permanece todavía en la base de datos con vistas a su verificación y conservación.

La orden DISCONNECT permite que un registro cambie su asociación con un conjunto. Por ejemplo, supongamos que la Factura 510 se asigna erróneamente al conjunto cuyo dueño es el Cliente 425, cuando debería haberse conectado con el conjunto cuyo dueño es el Cliente 431. Para conseguir el cambio que corrige esta situación se procedería como se muestra en el ejemplo siguiente.

Ejemplo 8: Modificación de la asociación de un conjunto

```
MOVE 510 TO NÚM_FACTURA OF FACTURA
FIND ANY FACTURA USING NÚM_FACTURA
MOVE 431 TO ID_CLIENTE IN CLIENTE
FIND ANY CLIENTE USING ID_CLIENTE
RECONNECT FACTURA IN CLIENTE_FACTURA
```

Clase de inserción en un conjunto. En el DBTG, la manera en la que un registro miembro consigue colocarse en una ocurrencia de un conjunto puede ser manual o automática.

Las dos primeras instrucciones localizan el registro deseado FACTURA. Las instrucciones tercera y cuarta localizan el registro deseado CLIENTE. La última instrucción conecta la Factura 510 al conjunto cuyo dueño es el Cliente 431. Esto también efectúa la supresión de la Factura 510 del conjunto cuyo dueño es el Cliente 425.

Clasificación de la asociación de conjuntos

Clase de retención en un conjunto. En el DBTG se determina cómo y cuándo un registro miembro puede suprimirse de un conjunto: puede ser fija, obligatoria u optativa.

La **clase de inserción en un conjunto** y la **clase de retención en un conjunto** son dos clases de asociación de conjuntos. La inserción de un conjunto se refiere a la manera en que un miembro consigue colocarse en una ocurrencia del conjunto. Una vez que un registro miembro se asigna a un conjunto, la clase de retención del conjunto determina cómo y cuándo un registro miembro puede suprimirse de ese conjunto.

Los modos de inserción de un conjunto se definen por la instrucción

```
INSERTION IS <modo de inserción>,
```

donde las opciones del modo de inserción son manual o automático.

Modo manual de inserción. En el DBTG, se requiere que el registro miembro se coloque en un conjunto utilizando una orden CONNECT para enlazarlo con la ocurrencia deseada del conjunto.

- El **modo manual de inserción** requiere que el registro miembro se coloque en el conjunto utilizando una orden CONNECT para enlazarlo con la ocurrencia deseada del conjunto. La inserción manual se realiza mediante la instrucción

  ```
  CONNECT <tipo de registro > TO <tipo de conjunto>
  ```

  Supongamos que tenemos el fragmento de subesquema siguiente:

  ```
  SET NAME IS CLIENTE_FACTURA.
  OWNER IS CLIENTE
  MEMBER IS FACTURA
  INSERTION IS MANUAL
  RETENTION IS OPTIONAL.
  ```

  Un ejemplo de su uso se presentó en el Ejemplo 6.

Modo automático de inserción. En el DBTG, cuando un nuevo registro miembro se crea, el SGBD lo conectará automáticamente a la ocurrencia del conjunto correcta.

- El **modo de inserción automático** significa que cuando se crea un nuevo registro miembro, el SGBD lo conectará automáticamente a la ocurrencia correcta del conjunto. Esta conexión ocurrirá siempre que se ejecuten instrucciones del tipo siguiente:

```
STORE <tipo de registro>
```

Supongamos que en el fragmento del subesquema precedente, MANUAL se reemplaza por AUTOMATIC. El ejemplo 6 cambiaría de la manera siguiente:

```
MOVE 431 TO ID_CLIENTE IN CLIENTE
FIND ANY CLIENTE USING ID_CLIENTE
MOVE 231 to NUM_FACTURA IN FACTURA
MOVE '7/7/90' TO FECHA IN FACTURA
MOVE 100.00 TO CANTIDAD IN FACTURA
STORE FACTURA
```

Las dos primeras líneas convierten al Cliente 431 en el registro actual del conjunto CLIENTE_FACTURA. Las tres instrucciones siguientes crean el nuevo registro FACTURA. La orden STORE insertará el registro en el conjunto deseado CLIENTE_FACTURA, ya que la conexión para este conjunto se define como AUTOMATIC.
Las opciones de retención en un conjunto son:

- Fija, significa que una vez que un registro miembro se ha asignado a una ocurrencia de un conjunto, no puede anularse de esa ocurrencia del conjunto a menos que el registro se suprima de la base de datos.
- Obligatoria, significa que una vez que un registro miembro se ha colocado en una ocurrencia de un conjunto, siempre tiene que estar en alguna ocurrencia de ese conjunto. No puede desconectarse, o reconectarse, a un conjunto de otro tipo.
- Optativa, significa que no hay restricciones impuestas sobre las conexiones o las reconexiones a los tipos de conjuntos. Un registro designado de esta manera no necesita estar conectado a algún conjunto cualquiera.

El modo de retención en un conjunto también rige, lo que se permite cuando se suprime un registro dueño de un conjunto. Si el modo de retención es **fijo**, se borrará el conjunto completo. Si el modo de retención es **obligatorio**, entonces la operación de eliminación es ilegal, puesto que los registros miembros tienen que pertenecer a una ocurrencia del conjunto. Si el modo de retención es **optativo**, entonces el registro se suprimirá y los registros miembros del conjunto para los cuales es dueño se desconectarán y permanecerán en la base de datos sin asociación al conjunto.

retención fija. En el DBTG, una vez que un registro miembro se ha asignado a una ocurrencia de un conjunto, no puede anularse de esa ocurrencia del conjunto a menos que el registro se suprima de la base de datos.

retención obligatoria. En el DBTG, una vez que un registro miembro se ha colocado en una ocurrencia de un conjunto, siempre tiene que estar en alguna ocurrencia de ese conjunto.

retención optativa. En el DBTG no hay restricciones impuestas sobre las conexiones o las reconexiones para los tipos de conjuntos.

Ejemplo 9: Estado de la inserción y de la retención en un conjunto. Consideremos el esquema de la Figura 15.12. Se ilustra el uso de la especificación del estado de la inserción y de la retención en un conjunto extendiendo la definición de CLIENTE_FACTURA y FACTURA_ARTÍCULO. Supongamos que queremos especificar una inserción MANUAL y una retención OPTIONAL. En la Figura 15.16 se muestra el esquema modificado.

```
CLIENTE_FACTURA
OWNER IS CLIENTE
MEMBER IS FACTURA
      INSERTION IS MANUAL
      RETENTION IS OPTIONAL

FACTURA_ARTÍCULO
OWNER IS FACTURA
MEMBER IS LÍNEA_DE_ARTÍCULO
      INSERTION IS MANUAL
      RETENTION IS OPTIONAL
```

▼ IDMS/R - UN SGBD del DBTG

IDMS/R significa Integrated Database Management System/Relational. Se basa en el modelo en red DBTG y, quizás, es el que ha tenido más éxito de los productos basados en este modelo. La R se incluyó para indicar la adición de ciertas características relacionales al IDMS. En principio, la interfaz relacional resulta de interés para el usuario. La estructura básica del IDMS corresponde estrechamente con las especificaciones en redes del DBTG.

El IDMS aplica los conceptos estructurales de registro y de conjunto, como se definen por el DBTG. El esquema del IDMS está compuesto por una sección de descripción del esquema (SCHEMA DESCRIPTION), una sección de descripción de registros (RECORD DESCRIPTION) y una sección de descripción de conjuntos (SET DESCRIPTION), de acuerdo con el diseño del DBTG. Además, incluye una sección de descripción de archivos (FILE DESCRIPTION) que define todos los archivos internos y los asigna a archivos externos. El IDMS también incluye una sección de descripción de áreas (AREA DESCRIPTION), que asigna particiones de archivos a áreas específicas. Un **área** es una zona de almacenamiento que contiene uno o más tipos de registros.

área. En el IDMS/R, una zona de almacenamiento que contiene uno o más tipos de registros.

El subesquema del IDMS no contiene una división de correspondencia (MAPPING) porque, en general, no se permiten alias para los registros, los conjuntos o los elementos de los datos. Cuando un registro del esquema se utiliza completamente en el subesquema, ello se expresa por

```
ELEMENTS ARE ALL
```

Si se requiere algún subconjunto de los elementos de los datos del registro, se denota por

```
ELEMENTS ARE <elemento de los datos 1> <elemento de los datos 2>. . .
<elemento de los datos n>,
```

donde los elementos de los datos se pueden permutar según las necesidades de la aplicación.

Los conjuntos se definen de manera similar a la prescrita por el DBTG, con algunas diferencias importantes. No está disponible la retención FIXED en un conjunto. Sólo se brindan las capacidades de retención MANDATORY y OPTIONAL. También existen ciertas restricciones bajo las cuales los registros pueden incluirse en el subesquema. El IDMS/R obliga a que los conjuntos que pudieran suprimirse borrando un registro dueño estén incluidos en *cualquier* subesquema que contenga ese registro. Esto evita la propagación de la orden ERASE a los registros que no se incluyen en el conjunto definido en este subesquema. Ello podría suceder si el registro dueño se borrara, ocasionando el borrado de los registros miembros y éstos, a su vez, fueran los dueños de otro conjunto no incluido en este esquema.

En general, las estructuras del IDMS son fieles a las recomendaciones del informe del DBTG.

▼ Evaluación del CODASYL DBTG

En esta sección se compara el modelo CODASYL del DBTG con el modelo relacional para evaluar comparativamente sus fortalezas y debilidades.

Representación de los datos

Una diferencia importante entre el modelo relacional y el modelo en red es la manera en que se representan las interrelaciones. En el modelo relacional, los enlaces entre dos relaciones se establecen incluyendo un atributo con el mismo dominio de valores —frecuentemente con el mismo nombre— en ambas relaciones. Las filas que están lógicamente rela-

cionadas tendrán en cada relación los mismos valores para ese atributo. En el modelo en red DBTG, la cardinalidad uno - muchos entre dos tipos de registros se establece mediante la definición explícita del tipo de conjunto. Entonces, el SGBD conecta los registros en cada tipo de conjunto mediante punteros físicos.

Esto significa que los registros se conectan físicamente cuando participan en la misma ocurrencia de un conjunto. Esta representación explícita de los tipos de conjunto se ha considerado como una ventaja del modelo en red. Un argumento a tener en cuenta es que el modelo en red utiliza dos conceptos de modelado, el tipo de registro y el tipo de conjunto, mientras que el modelo relacional utiliza un único concepto, la relación.

Lenguaje de manipulación de los datos

Las operaciones de navegación y de recuperación del DML DBTG se efectúan sobre registros simples, en contraste con las operaciones del modelo relacional que se efectúan sobre relaciones completas. Estas operaciones del DBTG deben estar inmersas en un lenguaje de programación anfitrión como el COBOL. Como las operaciones de manipulación orientadas a los registros se basan en las operaciones tradicionales de procesamiento de archivos, el programador necesita estar íntimamente familiarizado con los indicadores actuales y sus significados para evitar errores.

Parecería que el modelo relacional puede tener ventaja cuando se tiene en consideración el DML. Esto está parcialmente confirmado por la adición al IDMS de una interfaz de usuario relacional, convirtiéndolo en IDMS/R. Los lenguajes de los sistemas relacionales poseen herramientas de alto nivel para operar sobre los conjuntos de tuplas. Las implementaciones comerciales de estos sistemas han incorporado herramientas complementarias para el agrupamiento, la ordenación y el cálculo aritmético simple. Además, los DMLs relacionales pueden utilizarse directamente o pueden estar inmersos en un lenguaje anfitrión.

Restricciones de integridad

El modelo en red DBTG proporciona un conjunto útil de restricciones de integridad. Es particularmente fuerte en cuanto a su capacidad para proteger la integridad de los conjuntos. Las características de retención en los conjuntos permiten al diseñador determinar cómo se deben comportar los registros dueños con respecto a los registros miembros, y viceversa. Por ejemplo, las retenciones FIXED y MANDATORY requieren que cada registro tenga un dueño, mientras que la OPTIONAL no lo requiere.

La capacidad que expresa una restricción semántica, tal como la limitación de las horas trabajadas a 60 o menos para almacenarlas en un registro de empleado, solamente puede implementarse en el programa de aplicación que opera sobre esos registros.

Implementación

El modelo en red DBTG es especialmente conveniente para los sistemas de bases de datos que se caracterizan por:

- Tamaño grande.
- Consultas repetitivas bien definidas.
- Transacciones bien definidas.
- Aplicaciones bien definidas.

Si todos estos factores se presentan, entonces los usuarios y los diseñadores del sistema de base de datos pueden centrar sus esfuerzos en asegurar que las aplicaciones se programen de la manera más eficiente. El lado negativo es que las aplicaciones futuras no anticipadas pueden no marchar bien, e incluso pueden requerir una reorganización del sistema de base de datos con un alto grado de dificultad.

En este capítulo se han presentado los fundamentos del modelo de datos en red DBTG. Este modelo tiene un historial rico y es la base para varios SGBDs exitosos. Se pudo observar que un retículo forma una estructura de grafo, que es la manera natural de representar las interrelaciones entre los datos. Se planteó la historia de las recomendaciones del DBTG para la normas en redes y se utilizaron esas recomendaciones como base para el resto del capítulo.

Se encontró que el modelo de datos en red utiliza dos construcciones básicas: el tipo de registro y el tipo de conjunto. Los registros se definen como colecciones de los elementos de los datos lógicamente relacionados. Los conjuntos se definen mediante registros dueños y miembros que tienen un enlace lógico. Un diagrama de la estructura de datos para el modelo de datos en red consiste en cajas, que representan los tipos de registros, y arcos, que establecen las interrelaciones entre los tipos de registros. Estas interrelaciones nombradas forman los tipos de conjuntos.

Se mostró la manera en que los modelos de datos conceptuales se transforman en las estructuras de datos en redes y se proporcionaron las reglas para guiar este proceso. Se presentaron los métodos de transformación de interrelaciones uno-uno, uno-muchos y muchos-muchos. También se mostró cómo convertir interrelaciones *n*-arias a un conjunto equivalente de interrelaciones binarias creando un registro enlace.

Entonces se presentaron los lenguajes DBTG. El DDL DBTG proporciona un medio para crear y definir la base de datos. Los conceptos utilizados para definir la base de datos incluyeron el esquema, que define la estructura lógica del retículo en términos de sus tipos de registros y de sus tipos de conjuntos.

Una vez que se define el esquema, se definen las vistas externas de esa estructura según las necesidades de los usuarios y las aplicaciones de la base de datos utilizando un subesquema para cada una de las vistas. El subesquema permite la selección de aquellos datos del esquema que únicamente se necesitan. Los elementos de los datos pueden renombrarse o reordenarse para ajustarlos a las necesidades de la aplicación.

Después se discutió el DML DBTG y se dieron varios ejemplos. Se destacó que el DML DBTG se orienta a registros simples y que proporciona diversas herramientas para asegurar la integridad de la base de datos. Las órdenes DML pueden clasificarse en órdenes de navegación, órdenes de recuperación y órdenes de actualización. Las órdenes de actualización se utilizan tanto para actualizar los registros como para actualizar las ocurrencias de los conjuntos.

Se esbozó brevemente la implemeutación comercial del modelo de datos en red DBTG más ampliamente utilizada, IDMS/R. Aunque la R indica que se han añadido algunas de las características de la interfaz de usuario relacional, IDMS/R sigue básicamente las recomendaciones de diseño del DBTG.

Posteriormente se presentó una evaluación resumida del modelo DBTG. Generalmente se acepta que para transacciones predeterminadas sobre la base de datos, los programas pueden desarrollarse de modo que el modelo en red sea muy eficiente. Sus desventajas parecen centrarse en pocas facilidades para adaptarse a requisitos variables y a consultas *ad hoc*.

1. Defina con sus propias palabras cada uno de los términos siguientes:
   a. CODASYL
   b. DBTG
   c. IDS
   d. modelo de datos en red

e. DML
f. registro lógico
g. enlace físico
h. DDL
i. división de título
j. división de correspondencia
k. división de estructura
l. ocurrencia de registro
m. sección de registros del subesquema
n. ocurrencia de un conjunto
o. tipo de registro dueño
p. tipo de registro miembro
q. sección de conjuntos del subesquema
r. área de trabajo del usuario
s. indicador actual
t. bandera de estado
u. órdenes de recuperación
v. órdenes de navegación

2. Defina brevemente qué función ejecuta cada una de las órdenes siguientes del DML DBTG:
   a. FIND
   b. GET
   c. STORE
   d. ERASE

3. Explique brevemente la arquitectura de tres niveles del modelo en red DBTG. Centre su atención en las funciones de cada uno de los tres de niveles y cómo se relacionan unos con otros.

4. ¿De qué modo difiere la definición de conjunto del DBTG de la definición matemática de conjunto como una colección de objetos?

5. Muestre cómo el uso de un registro enlace permite transformar un retículo complejo en un retículo simple, utilizando un ejemplo.

6. ¿Qué sucedería si una interrelación n-aria ($n > 2$) no fuera transformada en interrelaciones binarias equivalentes en la estructura de datos en red?

7. Compare las maneras en que el modelo de datos relacional y el modelo DBTG representan las interrelaciones.

8. ¿Cómo difieren los lenguajes de manipulación de datos relacionales del DML DBTG?

9. Suponga que su jefe le pidió que explicara las ventajas y las desventajas relativas a los sistemas de bases de datos relacionales versus el modelo en red. ¿Qué diría?

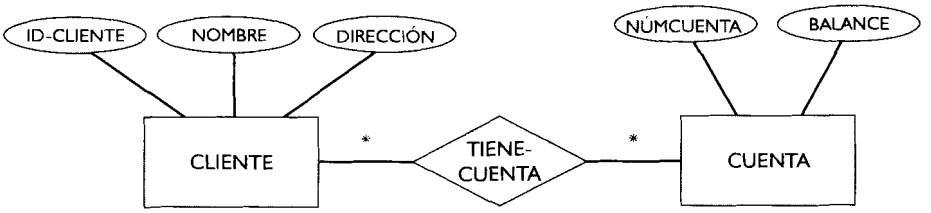
1. Haga corresponder los términos siguientes con sus definiciones:

—*esquema*

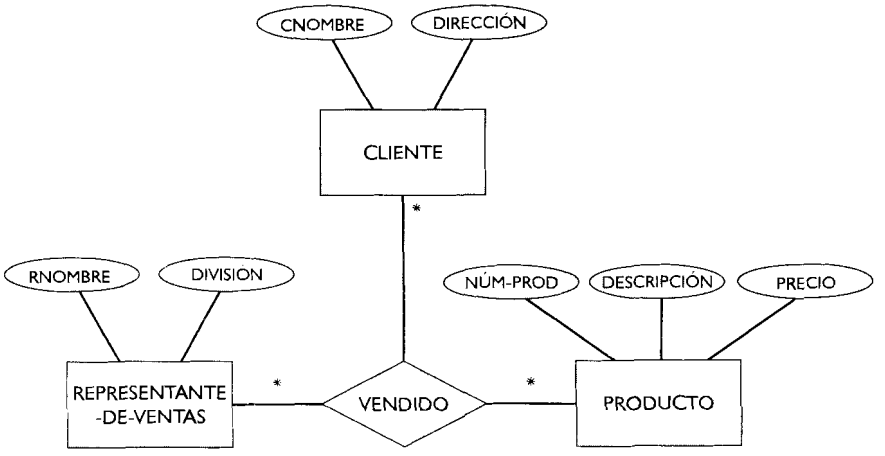
a. Un nombre dado a la interrelación entre un tipo de registro dueño y un tipo de registro miembro.

—*subesquema*	b. La vista del usuario de la estructura lógica de los datos de una base de datos.
—*tipo de registro*	c. En el DBTG determina cuándo un registro miembro puede suprimirse de un conjunto.
—*tipo de conjunto*	d. La estructura lógica de todos los datos y sus interrelaciones en la base de datos.
—*clase de retención de conjuntos*	e. El nombre dado a registros del mismo tipo.
—*retículo complejo*	f. Ocurre cuando por lo menos existe una interrelación muchos-muchos.
—*retículo simple*	g. Un registro ficticio creado a fin de permitir una implementación sobre un sistema DBTG.
—*registro enlace*	h. Una estructura de datos en la que todas las interrelaciones binarias son uno-muchos.

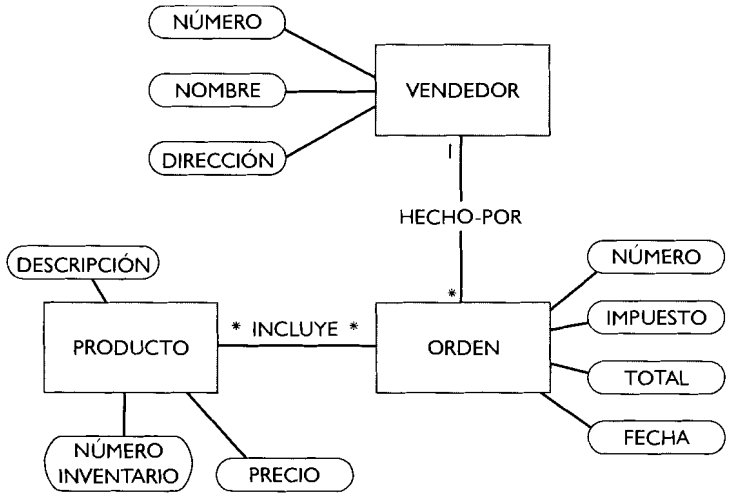
2. Exponga las reglas de este capítulo que se utilizarían para transformar el modelo conceptual de la Figura 15.1E en la estructura de datos del DBTG. Muestre la estructura de datos resultante.



3. Complete los requisitos del Problema 2 utilizando el modelo de datos conceptual de la Figura 15.2E.



4. Transforme el modelo de datos conceptual de la Figura 15.3E en una estructura de datos DBTG. Partiendo de esa estructura cree un esquema DBTG. Establezca las suposiciones necesarias.



5. A partir del esquema en el Problema 4, cree un subesquema para una aplicación en la que solamente se vean los tipos de registros VENDEDOR y ORDEN.

6. A partir de la estructura de datos DBTG creada en el Problema 4, escriba las instrucciones del DML para realizar lo siguiente:
   a. Recuperar el registro del Vendedor 13.
   b. Recuperar todas las órdenes con total igual a 100,00$.
   c. Eliminar la orden #256.
   d. Cambiar la dirección del Vendedor 13 a '912 Adams Street, Gainesville, FL'.
   e. Añadir el Vendedor 15 a la base de datos. El nombre del vendedor 15 es 'Mike Otteson' y su dirección es 'Buster Building, Suite 95, Toronto, Canadá'.
   f. Añadir la Factura #285 a la base de datos; conectarla al Vendedor 15.
   g. Eliminar la Factura #842 de la base de datos.

1. Investigue la literatura sobre las bases de datos con vistas a encontrar discusiones sobre los diversos modelos de bases de datos. En base a sus hallazgos, desarrolle un informe para ayudar a Rita Minkowski a tomar una resolución acerca del debate que ella escuchó en la reunión de la ACM.

2. Encuentre descripciones más detalladas del IDMS/R. ¿Qué características relacionales se han añadido? ¿Puede concluir que estas características superan cualesquiera de las ventajas que puedan haberse proclamado para los sistemas de bases de datos relacionales? Explique su respuesta.

3. ¿Qué sugerencias haría si se le encargara el desarrollo de recomendaciones para el perfeccionamiento del modelo en red DBTG? Sea específico.

4. ¿Puede perfeccionar el método para transformar un modelo de datos conceptual en una estructura de datos DBTG? ¿Cuáles serían sus propuestas y en qué medida mejorarían el método de transformación?

CAPÍTULO

16

EL MODELO DE DATOS JERÁRQUICO

Introducción
Conceptos básicos y definiciones
Las interrelaciones del modelo jerárquico para la semántica del modelado conceptual
  Transformando interrelaciones uno-muchos
    Regla 1
    Regla 2
  Transformar interrelaciones muchos-muchos
    Regla 3
    Regla 4
La arquitectura del sistema de gestión de información de IBM (IMS) [1]
  Definir la base de datos física, la DBD
  Definir la base de datos lógica, el BEP
Métodos de acceso del IMS
  HSAM
  HISAM
  HDAM
  HIDAM
El lenguaje de manipulación de datos del IMS
  El área de trabajo del programa
  DL/I : Una panorámica
  GET UNIQUE (GU)
    Ejemplo 1: Una simple recuperación de un segmento
    Ejemplo 2: Recuperación de un segmento dependiente
    Ejemplo 3: Recuperación del segmento dependiente cuando no se conoce la ocurrencia del padre
  GET NEXT (GN)
    Ejemplo 4: Recuperación de un conjunto de segmentos
    Ejemplo 5: Recuperar todos los segmentos de un tipo particular
  GET NEXT WITHIN PARENT (GNP)
    Ejemplo 6: Recuperar segmentos para un solo padre
    Ejemplo 7: Recuperar segmentos para un solo padre
  GET HOLD
  REPLACE (REPL)
    Ejemplo 8: modificando valores de campos de segmentos
  DELETE (DLET)
    Ejemplo 9: Borrar un segmento
  INSERT (INSRT)
    Ejemplo 10: Añadir un segmento
La evaluación del modelo de datos jerárquico
  La representación de los datos
  El lenguaje de manipulación de datos
Resumen
Preguntas de repaso
Problemas y ejercicios
Proyectos y cuestiones profesionales

[1] Siglas del término en inglés *IBM's Information Management System.*

William (Bill) Orange, el jefe de la sección local de la Association for Computing Machinery (ACM), estaba hablando por teléfono con Rita Minkowski. "Rita, nuestra discusión en la última reunión de los sistemas de base de datos fue bien recibida, por lo que hemos decidido dedicar las próximas tres reuniones al análisis de los modelos jerárquico, en red y relacional. ¿Piensas que podrías discutir el modelo jerárquico en la próxima reunión?"

"Naturalmente, Bill. ¿Qué te gustaría que presentara?"

"La filosofía del modelo y algunos de los aspectos más generales de la implementación de base de datos. Quizás podrías indicar algunas de las razones que pueden seleccionarlo por encima de los modelos en competencia. Creo que has tenido una experiencia completa con el sistema de base de datos IMS de IBM y eso solamente debería bastarte para describir el modelo jerárquico."

"Bueno, he invertido mucho tiempo con el IMS y sería agradable representar ese punto de vista en la próxima reunión. Gracias por tu invitación."

En este capítulo se presentan los fundamentos del modelo de datos jerárquico. Después de leer este capítulo, debería ser capaz de:

- Describir la estructura de datos básica desde la cual se construye el modelo de datos jerárquico.
- Explicar la terminología usada en la descripción del modelo de datos jerárquico.
- Usar los métodos fundamentales de transformación de un modelo de datos conceptual a un modelo de datos jerárquico.
- Describir la terminología y la estructura de la implementación de IMS.
- Explicar cómo el lenguaje de descripción de datos de IMS se usa para implementar estructuras de datos jerárquicos.
- Discutir cómo opera el lenguaje de manipulación de datos de IMS para la recuperación y la actualización de datos.
- Discutir las ventajas y las desventajas prácticas del modelo jerárquico.

IMS (IBM's Information Management System), SGBD basado en el modelo de datos jerárquico.

TDMS (System Development Corporation's Time-Shared Data Management System), SGBD basado en el modelo de datos jerárquico.

MARK IV (Control Data Corporation's Multi-Access Retrieval System), SGBD basado en el modelo de datos jerárquico.

System-2000 (del SAS Institute), SGBD basado en el modelo de datos jerárquico.

▼ Introducción

A diferencia del modelo de datos relacional, que se fundamenta firmemente en las matemáticas, y del modelo de datos en red, que se desarrolló a partir de un esfuerzo para establecer estándares detallados, el modelo de datos jerárquico se ha desarrollado a partir de la práctica. No existen documentos originales que definan el modelo jerárquico, como los hay para los otros dos modelos. Debido a que el modelo jerárquico no tiene un estándar, su estudio requiere el examen de los SGBDs usados en la práctica. Afortunadamente para el estudiante, las implementaciones de bases de datos jerárquicas están dominadas por un sistema, **IMS** (Sistema de Gestión de información de IBM). De hecho, IMS es actualmente el SGBD que generalmente más se usa. Las exposiciones en el modelo jerárquico invariablemente incorporan el vocabulario y las convenciones de IMS. Nosotros haremos lo mismo.

Sin embargo, se usan otros sistemas jerárquicos, incluyendo el **TDMS** (System Development Corporation's Time-Shared Data Management System), el **MARK IV** (Control Data Corporation's Multi-Access Retrieval System) y el **System-2000** (SAS Institute).

Ambos SGBDs, jerárquico y en red, fueron desarrollados a principios de 1960. El IMS se desarrolló con un esfuerzo conjunto entre IBM y North American Aviation (más tarde convertida en Rockwell) para desarrollar un SGBD como soporte al proyecto lunar Apolo —uno de los mayores proyectos de ingeniería acometidos en ese tiempo—. Un factor clave en el desarrollo de IMS fue la necesidad de manipular millones de piezas que se relacionaban unas con otras de una manera jerárquica. Esto significa que piezas más pequeñas se usaban para construir montajes más grandes, que a su vez se convertían en los componentes de módulos más grandes, y así sucesivamente.

Aunque el sistema relacional, DB2, está superando con rapidez a IMS en términos de número de instalaciones, para grandes sistemas de transacciones dirigidas que requieren respuestas rápidas, IMS continúa siendo un sistema competitivo. Una razón complementaria para la durabilidad de IMS es que muchas estructuras de datos son inherentemente jerárquicas. Por ejemplo, una compañía puede contener departamentos (primer nivel), los departamentos tienen empleados (segundo nivel) y los empleados tienen oficios (tercer nivel). Aunque esta estructura de datos podría implementarse en el modelo en red, que es el modelo más robusto en cuanto a capacidad de representación, puede ofrecer una complejidad de sistema mayor que la necesaria. De hecho, una razón por la que los que desarrollaron IMS no adoptaron el enfoque IDS usado en General Electric (ver Capítulo 9) fue que requería más espacio de disco que IMS.

padre. El punto de la cola de la flecha en una estructura de datos jerárquica.

hijo. El punto de la cabeza de la flecha en una estructura de datos jerárquica.

árbol. Una estructura de datos jerárquica, eso es, una estructura en red donde un tipo de segmento hijo es enlazado a sólo un tipo de segmento padre.

modelo de datos jerárquico. Un modelo de dato en el cual todas las interrelaciones son estructuradas como árboles.

tipo de segmento. Corresponde a un objeto en el modelo de dato orientado a objetos, también llamado un segmento.

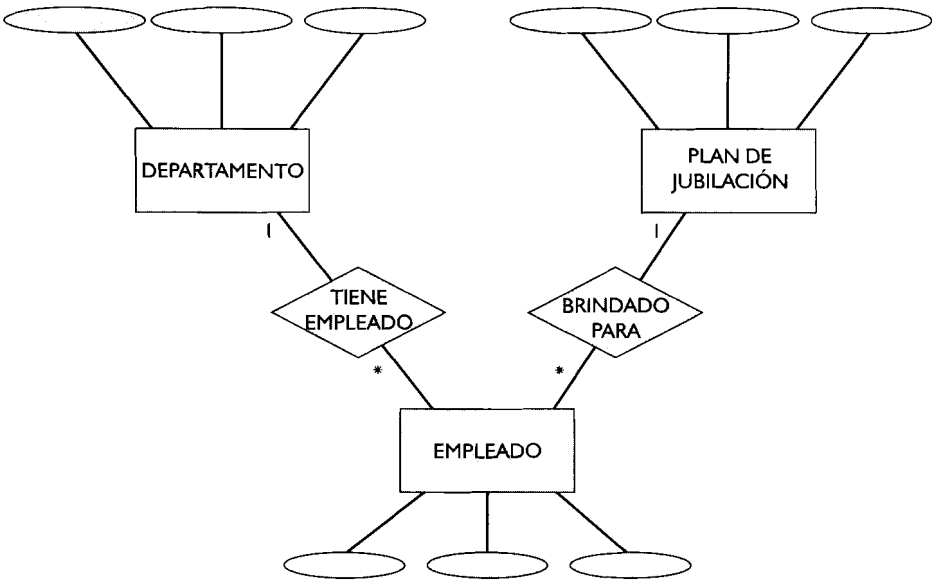
tipo de interrelación padre-hijo (tipo IPH).Interrelación lógica entre un tipo de segmento padre y un tipo de segmento hijo.

▼ Conceptos básicos y definiciones

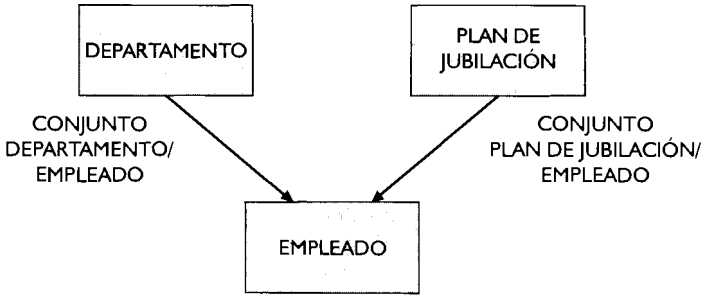
En el nivel conceptual, el modelo de datos jerárquico es simplemente un caso especial del modelo de datos en red. Como se describió en el capítulo anterior, una red es un grafo dirigido construido de puntos conectados por flechas. Aplicando este concepto a los modelos de datos, los puntos son tipos de registros de datos y las flechas representan interrelaciones de uno-uno o interrelaciones de uno-muchos. Una flecha en una red tiene un punto al final. El punto en la cola de la flecha en la terminología del modelo jerárquico se llama **padre**, y el punto en la cabeza de la flecha se llama **hijo.** La diferencia principal entre los modelos en red y jerárquico es que el modelo en red permite a un tipo de registro hijo tener más de un tipo de registro padre, mientras que el modelo jerárquico no. El tipo de estructura en red permitida por el modelo jerárquico se llama **árbol.** Las interrelaciones en el **modelo de datos jerárquico** se organizan como colecciones de árboles más bien que como grafos arbitrarios.

Aunque el vocabulario del modelo jerárquico varía un tanto en relación con el modelo en red, hay muchas similitudes de arquitectura natural. Igual que con el modelo en red, hay dos conceptos básicos asociados con la estructura de datos jerárquica: **tipos de segmentos** o simplemente segmentos y **tipos de interrelaciones padre-hijo** (tipos IPH). Los segmentos se usan análogamente al tipo de registro en red. Un tipo IPH es similar a un tipo de conjunto en red, excepto que un tipo de segmento sólo puede participar como un hijo en un tipo IPH.

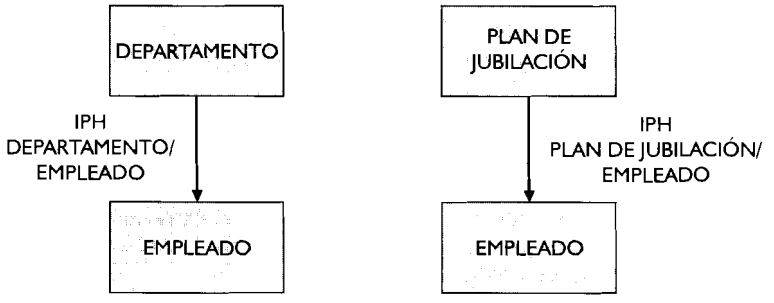
Como ejemplo, considérese la Figura 16.1. La Figura 16.1(a) muestra un modelo de datos conceptual, definiendo interrelaciones uno-muchos entre DEPARTAMENTO y EMPLEADO, y entre PLAN DE JUBILACIÓN y EMPLEADO. Cada conjuntos de objetos DEPARTAMENTO, EMPLEADO y PLAN DE JUBILACIÓN tiene sus propios atributos, aunque para el ejemplo considerado se omiten los nombres de los atributos. Además, la cardinalidad de las interrelaciones indica que cada departamento tiene muchos empleados, mientras que cada empleado se asigna a sólo un departamento. Además cada empleado tiene sólo un plan de jubilación, pero un plan de jubilación dado se brinda para muchos empleados. La representación de estos conjuntos de objetos y las interrelaciones en el modelo en red y en el modelo jerárquico se muestran en la Figura 16.1(b) y 16.1(c).



(a) Un modelo conceptual



(b) Un modelo de datos en red



(c) Un modelo de datos jerárquico

La Figura 16.1(b) ilustra una estructura de datos donde (en la terminología en red) el tipo de registro EMPLEADO es un miembro del conjunto DEPARTAMENTO/EMPLEADO, así como también lo es del conjunto PLAN DE JUBILACIÓN/EMPLEADO. Mientras esta estructura de datos puede implementarse directamente en el modelo

en red, no puede implementarse directamente en el modelo jerárquico. Para ser implementado en el modelo jerárquico, debe modificarse tal como se muestra en la Figura 16.1(c). Note que en la Figura 16.1(c) los conjuntos DEPARTAMENTO/EMPLEADO y PLAN DE JUBILACIÓN/EMPLEADO se han transformados a sus análogos jerárquicos: los tipos IPH DEPARTAMENTO/EMPLEADO y PLAN DE JUBILACIÓN/EMPLEADO. Además, el segmento EMPLEADO se ha mostrado dos veces, debido a que en el modelo jerárquico un segmento no puede participar como un hijo en más de un tipo IPH. El segmento EMPLEADO puede realmente participar sólo en *una* de estos IPHs. Como se explicará más adelante, EMPLEADO en otra IPH consistirá de punteros al segmento EMPLEADO original.

La razón para la transformación de la Figura 16.1(c) es que el modelo jerárquico usa el árbol como su estructura fundamental. Una estructura de datos en árbol se compone de una jerarquía de segmentos de acuerdo con los siguientes convenios:

segmento raíz. En un árbol, un tipo de segmento que no participa como un segmento hijo en ningún IPH.

1. Hay un segmento simple, llamado raíz, en el nivel superior. El **segmento raíz** no participa como segmento hijo en ningún tipo IPH.
2. Con excepción del segmento raíz, todo segmento participa como un segmento hijo en exactamente un tipo IPH.
3. Un segmento puede participar como un segmento padre en más de un tipo IPH (por ejemplo, EMPLEADO en la Figura 16.2).
4. Una ocurrencia de un segmento padre puede tener cualquier número de ocurrencias de segmento hijo (hijos), pero cada segmento hijo puede tener sólo un segmento padre. Esto establece una restricción uno-muchos en la interrelación entre los segmentos padre e hijo en un árbol.

segmento hoja. En un árbol, cualquier tipo de segmento que no tiene tipos de segmentos hijos.

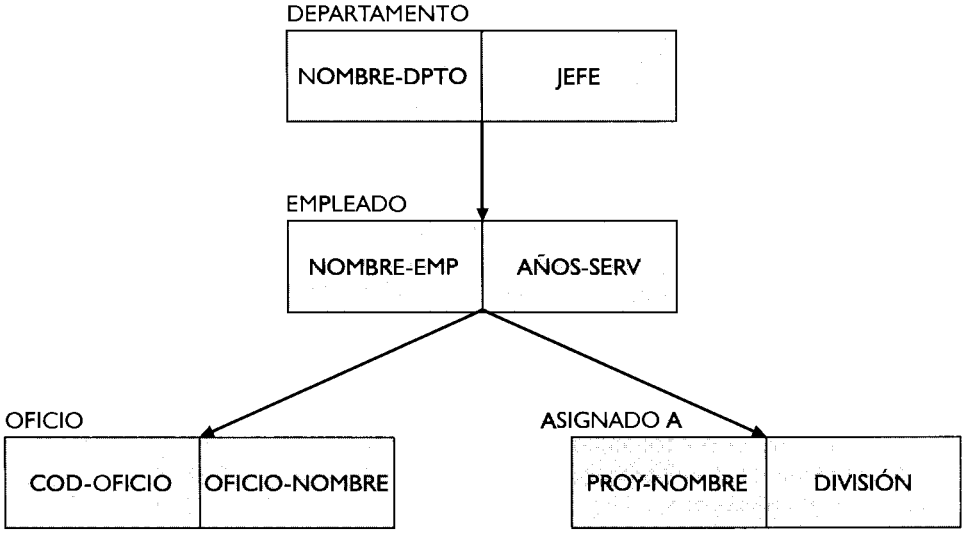
5. Un segmento que no tiene hijos se denomina un **segmento hoja.**

ancestro. En una jerarquía, un tipo de segmento que se encuentra en el mismo camino en un nivel superior en el árbol.

6. Para cualquier tipo de segmento A, existe un camino simple en el árbol desde la raíz hasta A. Los registros a lo largo de ese camino se llaman **ancestros** de A. A es un **segmento dependiente** de todos los segmentos de ese camino, incluyendo la raíz.

tipo de segmento dependiente. Todos los tipos de registros menos el tipo de segmento raíz.

7. Un segmento A puede ser la raíz de un subárbol.



árbol de la base de datos. Un árbol que tiene una raíz.

registro de la base de datos. Una ocurrencia de una raíz y todos sus tipos de segmentos dependientes.

La Figura 16.2 es un ejemplo de un árbol de tres niveles, en este caso representando los tipos de segmentos DEPARTAMENTO, EMPLEADO, OFICIO y ASIGNADO A y sus interrelaciones. Un esquema de base de datos jerárquico es una colección de árboles enraizados de este tipo. Cada tal árbol se denomina **árbol de la base de datos**. El árbol de la Figura 16.2 tiene DEPARTAMENTO como su segmento raíz y tiene dos tipos de subárbol, enraizados en el segmento EMPLEADO. Uno de estos tipos de subárbol va desde EMPLEADO hasta OFICIO y el otro desde EMPLEADO hasta ASIGNADO A. El segmento EMPLEADO tiene dos tipos de subárbol, enraizados en los tipos de segmento OFICIO y ASIGNADO A. Estos dos últimos tipos de subárboles son, en esencia, triviales, debido a que consisten simplemente de OFICIO sólo y de ASIGNADO A sólo. Una ocurrencia de una raíz y todos sus segmentos dependientes se denomina **un registro de la base de datos**. Un registro de la base de datos para el modelo de la Figura 16.2 consiste en una ocurrencia de un segmento DEPARTAMENTO, junto con todos sus segmentos asociados OFICIO y ASIGNADO A.

Observe que DEPARTAMENTO es el tipo de segmento padre del tipo IPH DEPARTAMENTO-EMPLEADO. Todos los tipos de segmentos que quedan (esto es, EMPLEADO, OFICIO y ASIGNADO A) son tipos de segmentos *dependientes*. EMPLEADO es un tipo de segmento hijo del tipo IPH DEPARTAMENTO-EMPLEADO. Además EMPLEADO funciona tanto como un tipo de segmento padre del tipo IPH EMPLEADO-OFICIO como un segmento padre del tipo IPH EMPLEADO-ASIGNADO A. Tal como en el modelo en red, la cabeza de la flecha indica el lado "mucho" de la interrelación uno-muchos.

La Figura 16.2 también nos da una oportunidad para comparar el modelo de datos jerárquico con el modelo de datos relacional. Las interrelaciones que en el modelo relacional estarían representadas por claves externas se representan en el modelo jerárquico por enlaces padre-hijo. Por ejemplo, el enlace entre DEPARTAMENTO y EMPLEADO podría realizarse en el modelo relacional poniendo el atributo DPTO-NAME en el registro EMPLEADO. En el modelo de datos jerárquico de la Figura 16.2, esta interrelación se representa por el enlace DEPARTAMENTO-EMPLEADO, el cual se implementa en las bases de datos jerárquicas, poniendo una dirección a disco física (o *puntero*) en el segmento DEPARTAMENTO.

segmentos gemelos. Ocurrencias de tipo de segmentos que tienen la misma ocurrencia de tipo de segmento padre (más de dos ocurrencias de segmentos pueden calificarse como gemelas).

árbol de ocurrencia jerárquico. Representación de segmentos de ocurrencias en una estructura de árbol que refleja todos los tipos IPH.

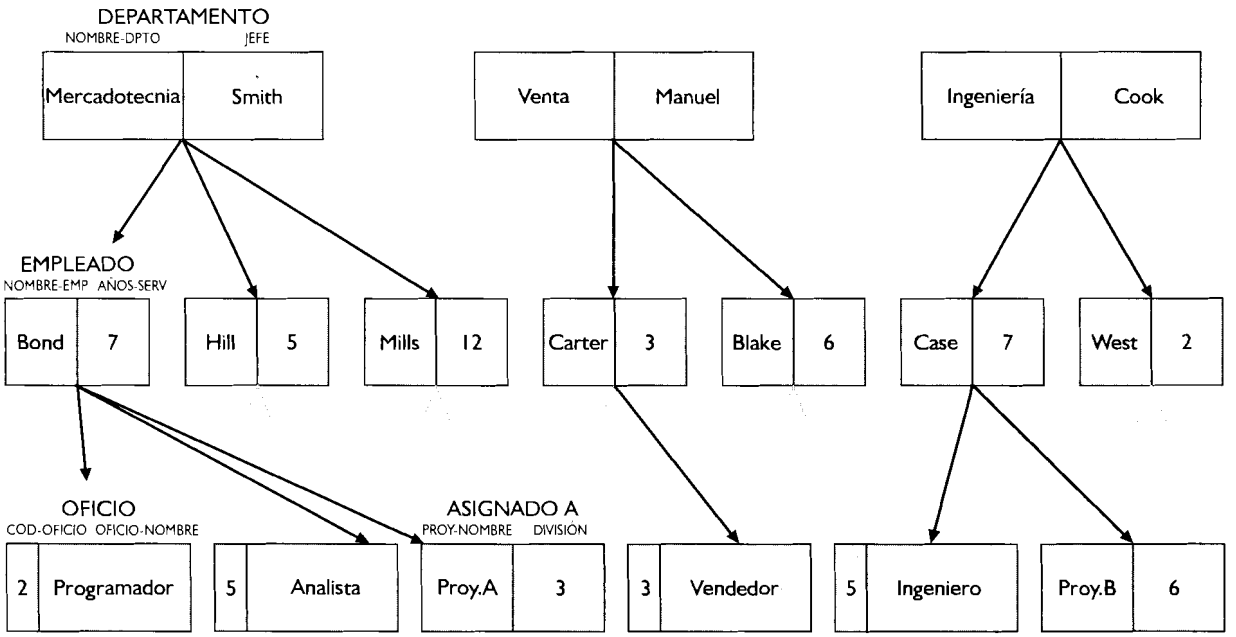
Volvamos a una consideración de ocurrencias de árboles de bases de datos. La Figura 16.3 expone una muestra de una ocurrencia de segmento para el modelo de la Figura 16.2. Para cada ocurrencia del árbol de la base de datos, existe por definición exactamente una ocurrencia de DEPARTAMENTO. Eso significa que cada departamento será la raíz de una ocurrencia distinta de árbol de base de datos. En este ejemplo, el departamento de Mercadotecnia está en un segmento raíz que tiene dos segmentos de empleados. Cada segmento de empleado posee uno o más segmentos OFICIO y uno o más segmentos ASIGNADO A. Esta ocurrencia del árbol de la base de datos incluye entonces tanto el segmento DEPARTAMENTO para Mercadotecnia como todos sus segmentos EMPLEADO, OFICIO y ASIGNADO A asociados.

Se usa la Figura 16.3 para añadir una nueva definición a nuestro vocabulario. Las ocurrencias del mismo tipo de segmento que tienen el mismo padre se denominan *gemelas* (twins). Para el segmento del Departamento de Mercadotecnia, los empleados Bond y Hill son ocurrencias de **segmentos gemelos** (aunque los segmentos gemelos no están limitados sólo a dos). El árbol mostrado en la Figura 16.3 se llama también **árbol de ocurrencia jerárquico.**

recorrido pre-orden. Un método de conversión de una estructura de árbol a un archivo plano que contiene la información necesaria sobre las interrelaciones jerárquicas.

Los árboles de ocurrencias jerárquicos pueden almacenarse usando el **método de recorrido en pre-orden** de creación de un archivo. Demostraremos el método en el árbol un poco más complejo de la Figura 16.4. El procedimiento es como sigue:

1. Comenzar en la raíz del árbol y registrar el segmento raíz en el archivo (el segmento A en la Figura 16.4).



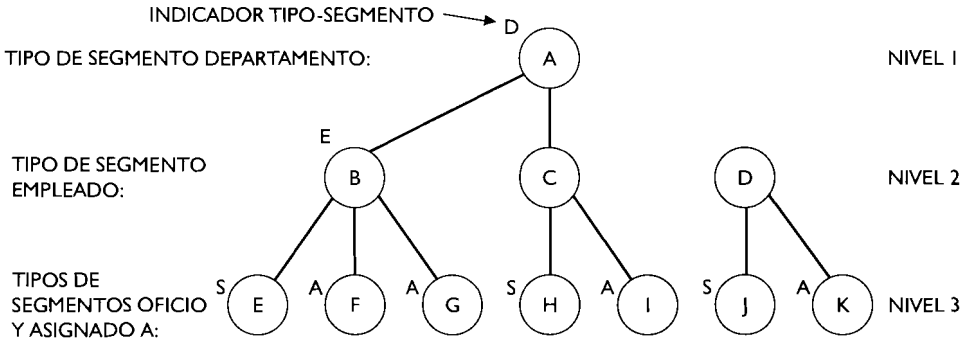
2. En cualquier punto, después del registro de un segmento, marcar el segmento hijo más a la izquierda del segmento registrado. Si el segmento no tiene segmentos hijos, moverse hacia atrás del árbol un nivel y marcar el segmento hijo no registrado más a la izquierda del segmento en ese nivel. Continuar hasta que todos los segmentos estén contabilizados.

Este procedimiento generaría el siguiente archivo:

Iteración	*Segmento*	*Indicador tipo-segmento*
1	*A*	*D*
2	*B*	*E*
3	*E*	*S*
4	*F*	*A*
5	*G*	*A*
6	*C*	*E*
7	*H*	*S*
8	*I*	*A*
9	*D*	*E*
10	*J*	*S*
11	*K*	*A*

El indicador tipo-segmento (*D* para DEPARTAMENTO, *E* para EMPLEADO) es necesario para la implementación porque el número de posibles segmentos hijos en cada IPH varía. Debido a que en el archivo se escribe secuencialmente, el sistema necesita conocer cuándo termina un grupo de segmentos de un tipo y cuándo comienza un grupo de un nuevo tipo.

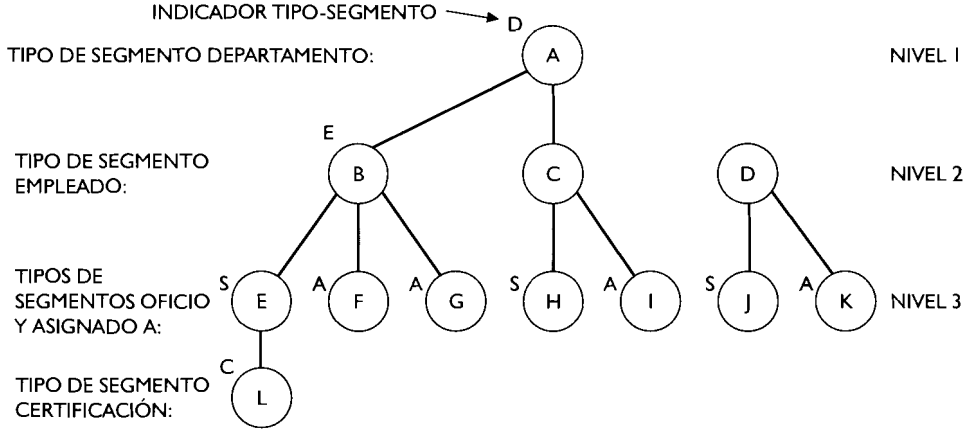
La Figura 16.4 muestra un árbol balanceado (ver Capítulo 10) en donde cada camino del segmento raíz al segmento hoja tiene la misma longitud (2). Esto es una coincidencia



con nuestro ejemplo y no es un requisito. La Figura 16.5 es un ejemplo de un árbol no balanceado. El camino del segmento *G* es de longitud 2, pero el camino del segmento *L* es de longitud 3.

▼ Las interrelaciones del modelo jerárquico para la semántica del modelado conceptual

La transformación de un modelo conceptual a una estructura de datos jerárquica es muy similar a la transformación al modelo en red, pero con algunas variaciones importantes debido al requisito de la estructura de árbol del modelo jerárquico. Se examinan varias estructuras del modelado de datos conceptual y se muestra cómo se pueden transformar en estructuras del modelo jerárquico.



Transformar interrelaciones uno-muchos

Examine el modelo conceptual de la Figura 16.6(a). En la Figura 16.6(b) se muestra la transformación de este modelo a la estructura de datos jerárquica correspondiente. El ejemplo sugiere que la transformación es idéntica a aquella aplicada para el modelo en red. Cada conjunto de objetos con sus atributos se convierte en un segmento lógico. Cada interrelación se convierte en un enlace binario y las interrelaciones se restringen a ser uno-muchos.

Sin embargo, recordemos que en el modelo en red el tipo de registro empleado podía pertenecer a más de un registro dueño, como se sugiere en el modelo de datos conceptual de la Figura 16.7(a). En una versión del modelo en red, el tipo de registro EMPLEADO pertenecería a dos tipos de registros: DEPARTAMENTO y PLAN DE JUBILACIÓN. En el modelo jerárquico, esto no se permite. La estructura de datos correspondiente requeriría dos árboles, como se sugiere en la Figura 16.7(b). Observe que esto introduce redundancia debido a que cada **ocurrencia del segmento** EMPLEADO se registra dos veces.

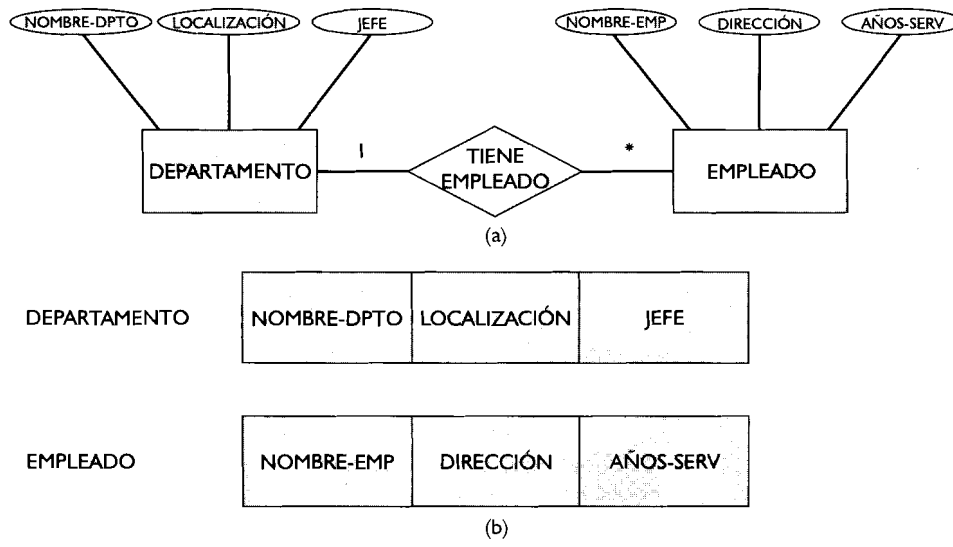
ocurrencia del segmento. Terminología del IMS para ocurrencias de registros.

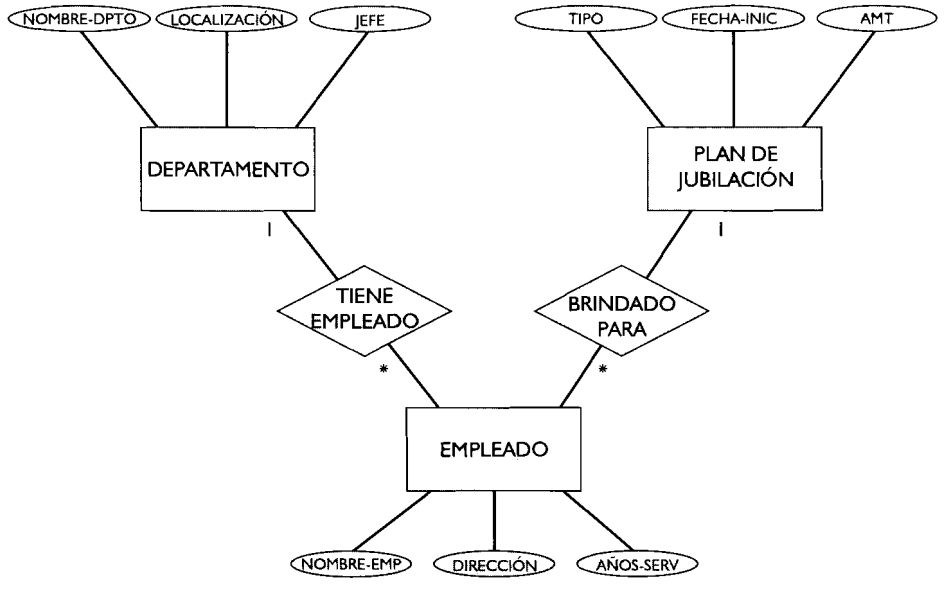
Sin embargo, esta redundancia puede limitarse. La primera ocurrencia de un segmento (en este caso, EMPLEADO) sería almacenada de la manera usual. La siguiente ocurrencia no almacenaría el segmento actual, sino un puntero en forma de dirección a disco física, dando la localización del segmento almacenado. Así, no hay redundancia de datos, pero hay un espacio de almacenamiento adicional requerido para el puntero.

Las primeras reglas para la transformación son:

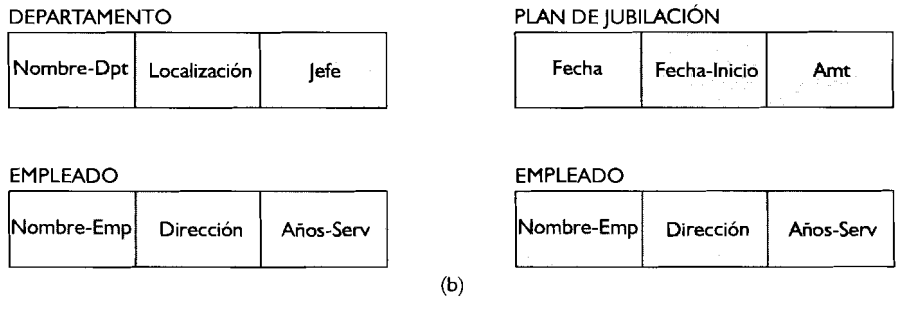
Regla 1. Para cada conjunto de objeto *O* en un modelo conceptual, cree un tipo de segmento *S* en el modelo jerárquico. Todos los atributos de *O* se representan como campos de *S*.

Regla 2. Para las interrelaciones uno-muchos entre dos conjuntos de objetos, cree diagramas, los diagramas correspondientes de árboles, haciendo cada conjunto de objetos un segmento y haciendo de la interrelación uno-muchos una interrelación padre-hijo. El segmento en la parte "mucho" de la interrelación se convierte en el segmento hijo y el segmento en la parte "uno" de la interrelación se convierte en el segmento padre.





(a)



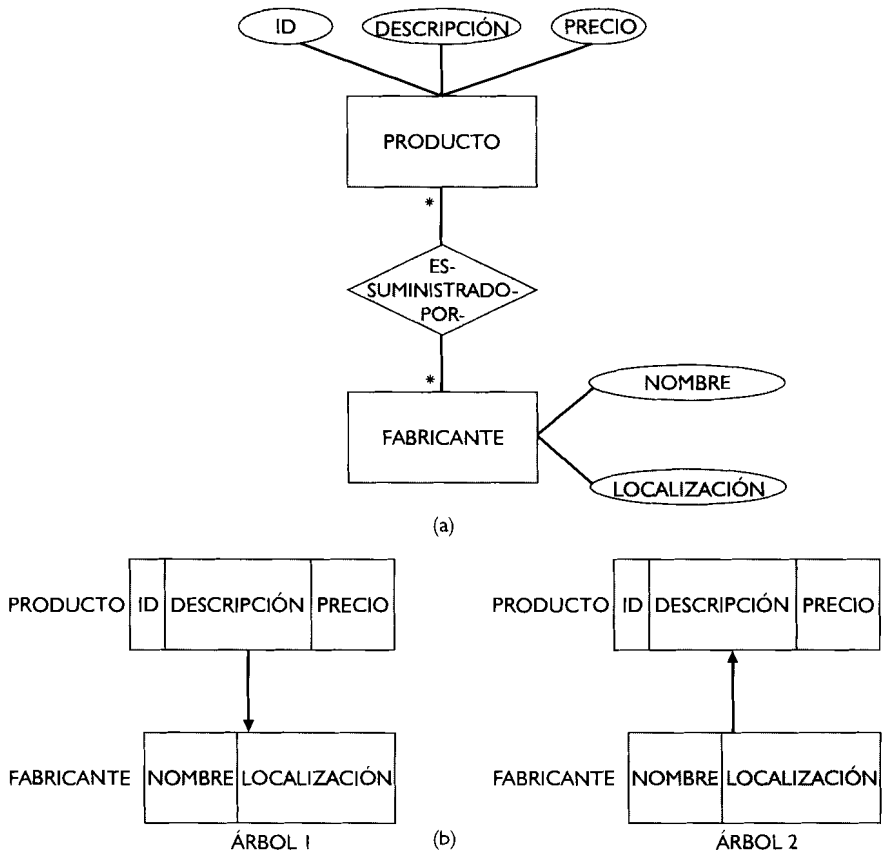
(b)

Transformar interrelaciones muchos-muchos

Las reglas 1 y 2 cubren un gran número de requisitos de transformaciones. Sin embargo, se puede generar un modelo orientado a objetos que requiera la transformación de interrelaciones binarias muchos-muchos. Recuerde que hubo situaciones similares para hacer corresponder el modelo orientado a objetos con el modelo en red.

La Figura 16.8(a) ilustra un fragmento de un modelo orientado a objetos de una interrelación producto-fabricante. La interrelación es de muchos-muchos, debido a que un producto dado puede suministrarse por muchos fabricantes, y un fabricante dado suministra un número de productos. El fragmento orientado a objeto se corresponde con la estructura de datos jerárquica, tal como muestra la Figura 16.8(b). Se han creado dos árboles, uno dando PRODUCTO como la raíz y otro dando FABRICANTE como su raíz. Esto proporciona la interrelación uno-muchos deseada en cada árbol. La regla que se ha seguido es:

Regla 3. Para los conjuntos de objetos, O1 y O2, que tienen interrelaciones binarias muchos-muchos y de los cuales se han definido los segmentos S1 y S2, construya dos IPHs

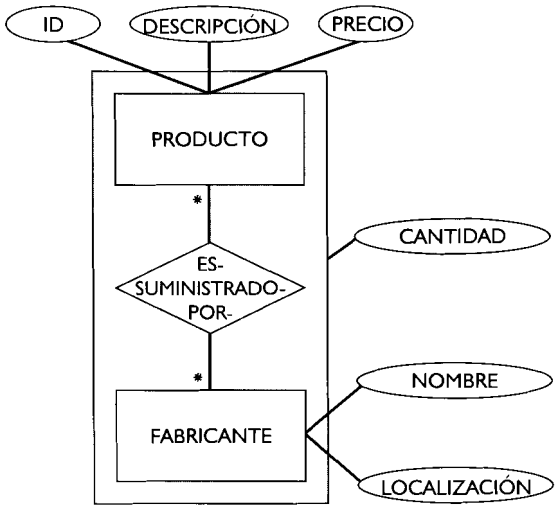


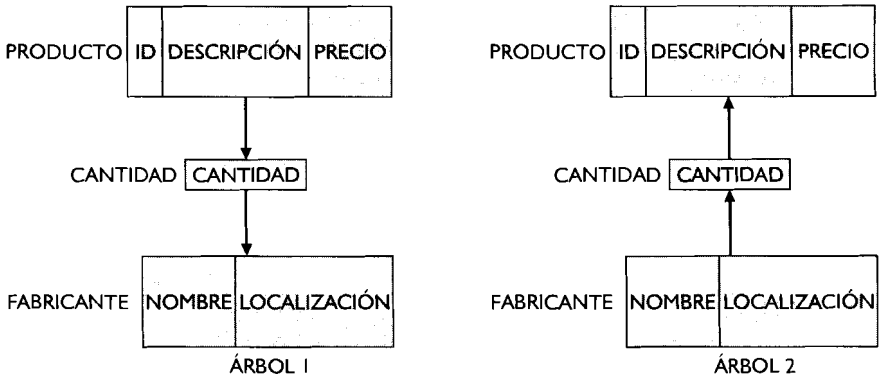
uno-muchos diferentes: de S1 a S2 y de S2 a S1. En uno de los dos IPHs, los segmentos de datos actuales para ambos tipos de segmentos se reemplazarán por punteros.

Una situación un tanto más compleja surge cuando la interrelación orientada a objetos tiene un atributo, como se muestra en la Figura 16.9. El atributo CANTIDAD se ha añadido al agregado de la interrelación ES-SUMINISTRADO-POR para indicar la cantidad máxima de un producto que puede suministrarse en un envío de un fabricante. En casos similares a éste, se crea un tipo de segmento adicional en correspondencia a la estructura de datos jerárquica.

Esto se muestra en la Figura 16.10. Ambos conjuntos de objetos involucrados en la interrelación binaria muchos-muchos funcionarán como un tipo de segmento padre en árboles separados. Un nuevo tipo de segmento CANTIDAD se inserta entre los segmentos PRODUCTO y FABRICANTE para indicar la cantidad máxima que un fabricante particular enviará de un determinado producto. En un árbol, el enlace uno-muchos se establece de PRODUCTO a CANTIDAD y de CANTIDAD a FABRICANTE. Este proceso se invierte en el otro árbol. Se tienen dos interrelaciones padre-hijo en cada estructura de árbol de tres-niveles. Este procedimiento se captura en la siguiente regla:

Regla 4. Si una interrelación binaria muchos-muchos tiene un dato de atributo, cree un nuevo segmento de intersección I, que contenga ese dato. Cada uno de los tipos de seg-





mento creados a partir de conjuntos de objetos funcionará como la raíz de un árbol diferente. Inserte el nuevo segmento entre los segmentos padre-hijo. Si cualquiera de estas interrelaciones padre-hijo son exactamente uno-uno, el dato de atributo podrá combinarse en los segmentos creados a partir de conjuntos de objetos.

▼ La arquitectura IMS

Debido a que el modelo de datos jerárquico no tiene estándar, se continúa usando su implementación más ampliamente utilizada, el IMS, como una base para la exposición. En este caso, si bien no es verdaderamente un estándar, el IMS se presenta como un modelo jerárquico básico.

Los registros de IMS deben ordenarse jerárquicamente, tal como se sugiere en la Figura 16.1(c). En la misma se ve un par de árboles de dos-niveles. En la Figura 16.2 se ve

un árbol de tres-niveles. IMS permite a lo sumo 15 niveles en un árbol, aunque es raro una implementación que contenga más de cuatro niveles en un árbol. IMS también limita el número de tipos de segmentos a 225.

DL/1. (Data language 1). lenguaje de manipulación de datos del IMS.

El **DL/1** (Data Language 1) es el componente del IMS que se proporciona para el almacenamiento y la recuperación de base de datos. Una equivalencia genérica para el esquema usado en el modelo de datos en red es la **descripción de la base de datos (DBD)**[2]. Esto da el armazón para la estructura interna de la base de datos. Se establece la manera en la cual los datos se almacenan para su uso por IMS. DBD define el formato, la longitud y la localización de cada dato para ser accedido por el DL/1. Se define también la posición de cada segmento en la estructura del árbol.

descripción de la base de datos (DBD). La manera en que los datos son almacenados físicamente en el IMS (estructura interna de la base de datos).

La **base de datos externa**, o la vista de usuario, que se expresó por el subesquema en el modelo en red, se describe en IMS por el **bloque de especificación de programa (BEP).** BEP especifica los nombres de cada segmento que un programa accederá. BEP, a su vez, se subdivide en **bloques de comunicación de programas (BCPs)** que definen la visualización de la base de datos que se requiere para cada aplicación del usuario. BCP contiene el nombre de BEP, del cual él es un subconjunto.

base de datos externa. La vista del usuario de los datos en el IMS.

Ejemplos del BCP y del BEP ilustrarán la arquitectura básica del IMS.

bloque de especificación de programa (BEP). Especifica los nombres de cada segmento en una aplicación de programa que accederá, corresponde a la vista de usuario o subesquema.

Definir la base de datos física - El DBD

bloque de comunicación de programa (BCP). Una componente del BEP.

Como se dijo anteriormente, el DBD es similar al esquema en red y el BEP es similar al subesquema de red. Aclaremos primero las funciones del DBD con un ejemplo. La Figura 16.12 presenta cuatro tipos de segmentos: DEPARTAMENTO, EMPLEADO, OFICIO y ASIGNADO A. El DBD para este árbol se presenta en la Figura 16.11. Las instrucciones se han numerado para su explicación.

La instrucción 1 es la instrucción DBD. Esta instrucción identifica el nombre de la base de datos, en este caso NAME = DEPTPERS. La instrucción también especifica el método de acceso a la base de datos que se usará, en este caso HISAM, que denota el método de acceso secuencial indexado jerárquico. (Los métodos de acceso de IMS se estudiarán más adelante en una sección posterior de este capítulo.)

```
 1   DBD     NAME = DEPTPERS, ACCESS = HISAM
 2   SEGM    NAME = DEPARTAMENTO, PARENT = 0, BYTES = 20
 3   FIELD   NAME = (NOMBRE-DPTO, SEQ, U), BYTES = 10, START = 1, TYPE = C
 4   FIELD   NAME = JEFE, BYTES = 10, START = 11, TYPE = C

 5   SEGM    NAME = EMPLEADO, PARENT = DEPARTAMENTO, BYTES = 22
 6   FIELD   NAME = (NOMBRE-EMP, SEQ), BYTES = 20, START = 1, TYPE = C
 7   FIELD   NAME = AÑOS-SERV, BYTES = 2, START = 21, TYPE = P

 8   SEGM    NAME = OFICIO, PARENT = EMPLEADO, BYTES = 17
 9   FIELD   NAME = (COD-OFICIO, SEQ), BYTES = 2, START = 1, TYPE = P
10   FIELD   NAME = OFICIO-NOMBRE, BYTES = 15, START = 3, TYPE = C

11   SEGM    NAME = ASIGNADO A , PARENT = EMPLEADO, BYTES = 4
12   FIELD   NAME = (PROY-NOMBRE, SEQ), BYTES = 2, START = 1, TYPE = P
13   FIELD   NAME = DIVISIÓN, BYTES = 2, START = 5, TYPE = C

14   DBDEN
```

[2] La siglas en inglés son también DBD (*Data Base Description*) (N. del T.).

SEGM. Una instrucción del IMS que define los campos que se incluyen en un segmento para usarse por un programa.

La instrucción DBD va seguida por una serie de instrucciones de segmentos (SEGM). La instrucción **SEGM** define el grupo de elementos de datos (campos en el IMS) que abarca ese segmento y cómo se ordenan. El primer segmento en la Figura 16.11 se identifica con la instrucción 2. El segmento se nombra DEPARTAMENTO. La asignación PARENT = 0 significa que DEPARTAMENTO es un segmento raíz, "0" significa que no tiene segmento padre. BYTES = 20 establece la longitud del segmento. En la instrucción 3, NOMBRE-DPTO es el primer campo en el segmento DEPARTAMENTO. Se identifica como un campo secuencial para el segmento DEPARTAMENTO por la asignación NAME = (NOMBRE-DPTO, SEQ, U). Eso significa que cuando se almacenan las nuevas ocurrencias de los segmentos, se ordenarán en secuencia de acuerdo con el valor de este campo en el segmento. La "U" significa que NOMBRE-DPTO debe ser único —dos segmentos no pueden tener el mismo NOMBRE-DPTO—. Como puede verse, las especificaciones que quedan definen la localización de inicio del campo en el segmento, la longitud del campo y el tipo de dato. Los tipos de datos más comunes son *P* (packed decimal) y *C* (character).

Se necesita describir al menos un segmento más para ver cómo se establecen las interrelaciones padre-hijo. La instrucción 5 comienza con la definición del segmento EMPLEADO. Después de darle un nombre, se le asigna el parámetro PARENT = DEPARTAMENTO. Esto establece que la interrelación padre-hijo es entre los segmentos DEPARTAMENTO y EMPLEADO. Las ocurrencias de los segmentos se añaden en orden por el nombre del empleado [NAME = (NOMBRE-EMP, SEQ)], pero estos nombres no necesitan ser únicos como con NOMBRE-DPTO en el segmento anterior.

Con estas dos descripciones, el resto de la Figura 16.11 se interpreta fácilmente.

Definir la base de datos lógica - El BEP

Recuerde que BEP es algo similar al subesquema de red, porque se especifica la vista de los datos que se usan en una aplicación. El BEP contiene uno o más bloques de comunicación de programas (BCPs). El BCP especifica los segmentos que una aplicación puede acceder. Los programas no pueden acceder a segmentos que no se definen en el BCP. Por consiguiente, el BCP identifica la base de datos adecuada y especifica cualquier **segmento susceptible** a ser incluido (SENSEG)[3].

segmento susceptible. Un segmento que es accesible para un programa.

Considere el ejemplo mostrado en la Figura 16.12. Suponga que la Corporación Zeus tiene una aplicación que requiere exactamente los nombres de los DEPARTAMENTOS y los nombres de los EMPLEADOS de estos dos segmentos. Comenzando con la línea 1, se requiere TYPE = DB para cada BCP a definirse. La base de datos de la cual se extrae este BCP se llama DEPTPERS (DBNAME = DEPTPERS). PROCOPT se usa para indicar opciones de procesos, que significa las operaciones que la aplicación puede ejecutar en un BCP. Estas opciones incluyen:

```
G = GET
I = INSERT
R = REPLACE
D = DELETE
A = ALL
```

GET especifica accesos de sólo lectura a los segmentos en el BCP. INSERT permite añadir nuevos segmentos al BCP. REPLACE permite que los segmentos sean recuperados y modificados. DELETE permite que los segmentos sean recuperados y borrados. PROCOPT = A, como se muestra en la Figura 16.12, significa que todas estas opciones de procesos se permiten para esta aplicación. Cuando un segmento se usa por el programa, IMS identifica su localización en la base de datos registrando una **clave concatenada totalmente.** KEYLEN = 18 [4] indica la mayor clave concatenada que el programa puede acceder.

clave concatenada totalmente. Significado de la identificación de la localización de un segmento en la base de datos.

---

[3] Del inglés *SENsitive SEGment* (N. del T.).

[4] Siglas del inglés *KEY LENgth* (longitud de la llave) (N. del T.).

Básicamente, esto se refiere a la mayor clave que se obtendría añadiendo los campos claves que permiten cualquier camino a través de la jerarquía.

```
1    BCP        TYPE = DB, DBNAME = DEPTPERS, PROCOPT = A, KEYLEN = 18
2    SENSEG     NAME = DEPARTAMENTO, PARENT = 0
3    SENFLD     NAME = NOMBRE-DPTO, START = 1
4    SENSEG     NAME = EMPLEADO, PARENT = DEPARTAMENTO
5    SENFLD     NAME = NOMBRE-EMP, START = 1
```

Las instrucciones SENSEG y SENFLD (líneas 2-5) identifican aquellos segmentos y campos que son accesibles a la aplicación En la Figura 16.12 hay dos segmentos susceptibles (accesibles al programa): DEPARTAMENTO y EMPLEADO. Los campos susceptibles para los respectivos segmentos se denotan por la instrucción SENFLD, con el mismo significado (accesibilidad). Observe que los segmentos dependientes deben identificar explícitamente sus segmentos padres. Por ejemplo, el padre de EMPLEADO es DEPARTAMENTO, tal como se indica en la línea 4 por la cláusula PARENT = DEPARTAMENTO. START = 1 significa que el campo comienza en el primer byte del segmento.

▼ Métodos de acceso de IMS

IMS proporciona cuatro métodos de acceso: HSAM, HISAM, HDAM e HIDAM. Recuerde que esta opción se refleja en la DBD por la entrada "ACCESS = <método de acceso elegido>". Las siguientes subsecciones resumen los cuatro métodos.

HSAM

HSAM. Método de acceso del IMS, muy rápido para recuperaciones de segmentos secuenciales.

HSAM indica el *método de acceso secuencial jerárquico*[5] de acceso a la base de datos. Los segmentos están contiguos físicamente en los medios de almacenamiento, además HSAM podría implementarse en cinta o disco. Los segmentos se ordenan de acuerdo con el esquema transversal pre-orden, que permite mantener la estructura de datos jerárquica.

HSAM es prácticamente solo para la lectura de los datos. Esto quiere decir que la estructura no es lo suficientemente flexible para soportar con efectividad la actualización de la base de datos. La actualización de los segmentos requiere que se cree y se almacene una nueva versión de la base de datos.

HISAM

HISAN. Método de acceso de IMS, brinda la capacidad para recuperaciones de segmentos tanto directa como secuencial.

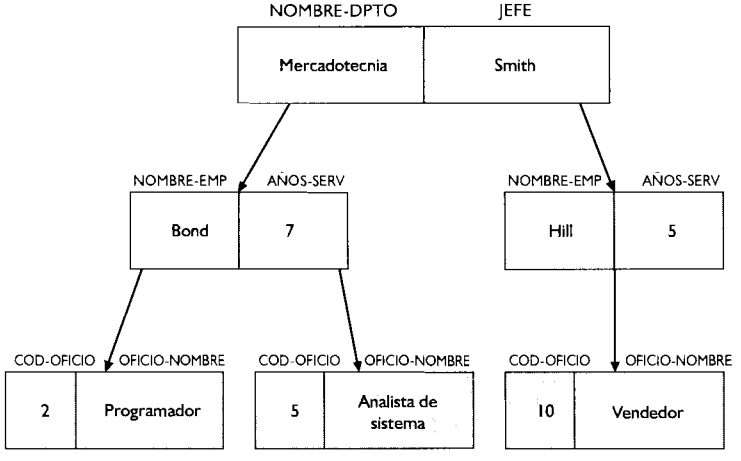
HISAM, *método de acceso secuencial-indexado jerárquico*, almacena segmentos en secuencia jerárquica, como en el HSAM, pero brinda la capacidad para mejorar el acceso directo a segmentos raíces específicos de acuerdo a un índice. Después de eso, los segmentos dependientes del segmento raíz que se ha accedido se acceden secuencialmente, como con el HSAM.

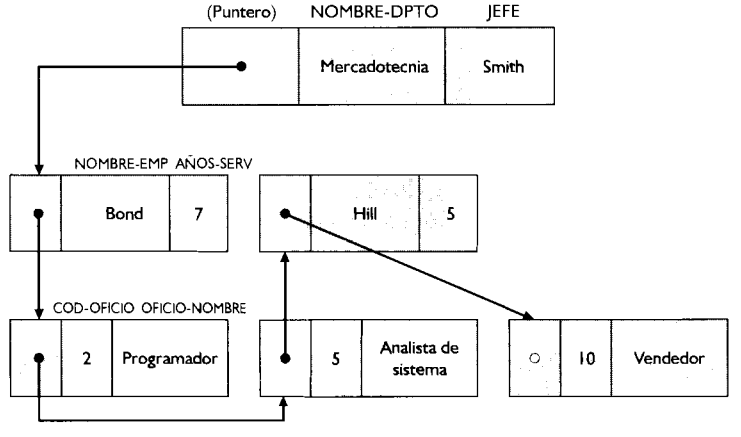
[5] Por ser siglas establecidas, se mantienen en inglés HSAM (*Hierarchic Sequential Access Method*), HISAM (*Hierarchic Indexed-Sequential Access Method*), HDAM (*Hierarchic Direct Access Method*) (N. del T.).

HDAM

HDAM. Método de acceso del IMS, brinda acceso-directo de segmentos muy rápidos, pero no la capacidad de procesamiento secuencial.

HDAM soporta el *método de acceso-directo jerárquico*. Este método no relaciona segmentos por un índice o por la proximidad física, excepto sólo a través de punteros (campos que contienen direcciones físicas a disco). El acceso a las raíces se mejora a través del uso de un algoritmo de búsqueda (ver Capítulo 10). Examine el árbol de ocurrencia de la Figura 16.13. Los punteros que enlazan segmentos de una ocurrencia pueden ser *jerárquicos*, tal como se muestra en la Figura 16.14 o *hijos y gemelos*, como se muestra en la Figura 16.15.





Cuando se usan punteros jerárquicos, cada segmento apunta a su sucesor en la secuencia transversal pre-orden ("0" indica que los segmentos no siguen). Por ejemplo, para acceder al registro del empleado Hill en la Figura 16.14, se comenzaría por el registro del departamento de Mercadotecnia, usando su puntero se llega al registro para el empleado Bond, entonces se sigue el puntero de ese registro hasta el registro de oficio Programador, desde allí hasta el registro de oficio Analista y finalmente hasta el registro de Hill.

Cuando se usan punteros hijos-y-gemelos, cada segmento contiene un puntero al segmento inmediato en el mismo nivel (gemelo) y un puntero al segmento hijo inmediato. En la

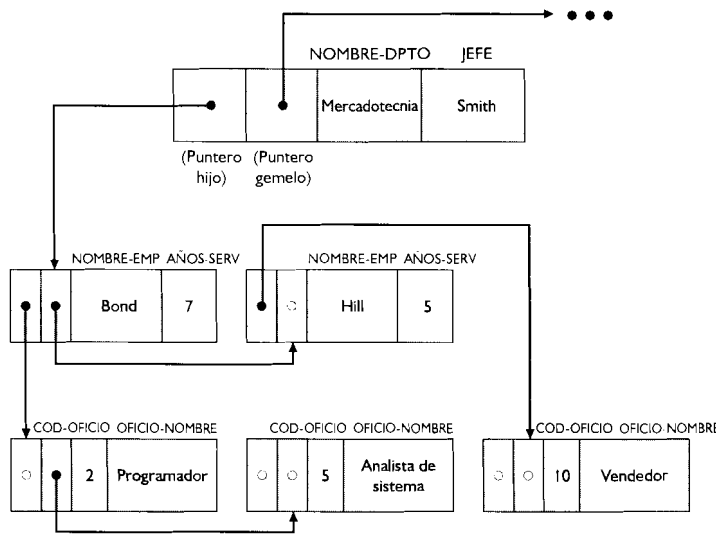


Figura 16.15, el puntero más a la izquierda es el puntero hijo y el puntero a su derecha es el puntero gemelo. Los punteros gemelos son simples de implementar debido a que cada segmento contiene exactamente dos punteros.

Para acceder al registro de Hill en la Figura 16.15, se comienza una vez más por el registro del departamento de Mercadotecnia, se sigue el puntero hijo al registro del primer empleado (Bond) y se sigue el puntero *gemelo* de ese registro hasta el registro de Hill. Para acceder al registro de oficio Vendedor de Hill, se sigue simplemente el puntero hijo en el registro de Hill directamente al registro deseado.

HIDAM

HIDAM. Método de acceso del IMS, brinda tanto acceso directo a los segmentos raíces como recuperación secuencial.

HIDAM, *método de acceso-directo indexado jerárquico*[6], es básicamente similar al HDAM, pero permite tanto acceso indizado a raíces como acceso por punteros a segmentos dependientes.

▼ El lenguaje de la manipulación de datos del IMS

En esta sección se definen e ilustran las características del lenguaje de manipulación de datos, DL/1. Se necesita entender el acceso y la manipulación del programador en una base de datos de IMS para interactuar correctamente con ellos.

El área de trabajo del programa

Para el DL/1 ejecutar operaciones en una base de datos de IMS, el sistema mantiene un área de trabajo del programa que contiene las siguientes variables:

plantilla de segmento. El formato del área de trabajo del programa para un segmento.

- Las **plantillas de segmentos** (*segment templates*), que dan la composición del segmento para cada tipo de segmento en la base de datos.

[6] Por ser siglas establecidas, se mantienen en inglés HIDAM (*Hierarchic Indexed Direct Access Method*) (N. del T.).

puntero actual. Contiene la dirección del último segmento que se accedió en el árbol.

bandera de estado. Campo cuyo valor indica el resultado de la última operación de la base de datos (por ejemplo, si tuvo éxito o no).

- Los **punteros actuales** para cada árbol de la base de datos. Éstos contienen la dirección del segmento en el árbol que se ha accedido más recientemente.
- Las **banderas de estado** que indican el resultado de la última operación de la base de datos. Por ejemplo, podría asignarse un "0" si la última operación tuvo éxito y otro símbolo en caso contrario. Siguiendo una operación, el programa puede referirse a la bandera de estado para determinar qué hacer a continuación.

DL/1: Una panorámica

El lenguaje de manipulación de datos para IMS se denomina DL/1. A continuación se dan ejemplos de las órdenes más comúnmente usados del DL/1:

Órdenes	*Significado*
GET UNIQUE (GU)	Recuperación del primer segmento de datos que satisface una condición dada.
GET NEXT (GN)	Recuperación del próximo segmento.
GET NEXT WITHIN PARENT (GNP)	Recuperación del próximo segmento, pero sin el segmento padre actual.
GHU, GHN, GHNP	Cerrar la base de datos para GU, GN, GNP.
INSERT (ISRT)	Añadir un nuevo segmento a la base de datos.
REPLACE (REPL)	Modificar el valor de un campo del segmento.
DELETE (DLET)	Borrar un segmento.

La sintaxis común para el *DL/1* es:

Orden <nombre del segmento> <WHERE requisito>

Veamos algunos ejemplos del uso de estas órdenes.

Get Unique (GU)

La orden GET se usa para seleccionar una ocurrencia del segmento. GET UNIQUE selecciona un segmento, dando un valor particular al área de trabajo. El segmento deseado se define entre paréntesis por un requisito eliminatorio llamado un argumento de búsqueda del segmento (ABS).

Ejemplo 1: Una simple recuperación de un segmento

```
GU DEPARTAMENTO (NOMBRE-DPTO = 'Mercadotecnia')
```

En este ejemplo, el argumento de búsqueda del segmento es NOMBRE-DPTO = = 'Mercadotecnia'. La orden GU recuperará el primer segmento que satisface el ABS. En el próximo ejemplo, observe la diferencia cuando se desea recuperar el segmento para el empleado Steve Smith, quien fue asignado al Departamento de Mercadotecnia.

Ejemplo 2: Recuperación de un segmento dependiente

```
GU DEPARTAMENTO (NOMBRE-DPTO = 'Mercadotecnia')
   EMPLEADO (NOMBRE-EMP = 'Steve Smith')
```

Debido a que el segmento EMPLEADO es dependiente, se especifica un camino jerárquico. El operador GU recuperará sólo el último segmento del camino. Eso significa que el segmento Mercadotecnia no se recuperará.

Suponga que no se conoce el departamento al cual Steve Smith fue asignado. La consulta podría tratarse como se muestra en el próximo ejemplo.

Ejemplo 3: Recuperación del segmento dependiente cuando no se conoce la ocurrencia padre

```
GU DEPARTAMENTO
   EMPLEADO (NOMBRE-EMP = 'Steve Smith')
```

Esta consulta examinará secuencialmente los segmentos de DEPARTAMENTO hasta que sea encontrado el segmento dependiente para Steve Smith.

Get Next (GN)

Si en lugar de simplemente recuperar la primera ocurrencia de un segmento que tiene un valor específico, se desea recuperar todos los segmentos que tienen ese valor, se puede usar GU en coordinación con la orden GET NEXT (GN). Por ejemplo, suponga que se quiere recuperar todos los segmentos de EMPLEADOS para el Departamento de Mercadotecnia. Esto se haría de la siguiente manera:

Ejemplo 4: Recuperación de un conjunto de segmentos

```
GU DEPARTAMENTO (NOMBRE-DPTO = 'Mercadotecnia')
   EMPLEADO
GN EMPLEADO
```

El operador GU efectuará la recuperación del primer segmento de EMPLEADO para el Departamento de Mercadotecnia. El operador GN controla entonces la recuperación del próximo segmento de empleado para el Departamento de Mercadotecnia. Mientras exista una segunda ocurrencia de empleado para el departamento de Mercadotecnia, esta consulta hace lo que se quiere. Sin embargo, si no existieran los demás segmentos de empleados cuando se ejecuta la orden GN, el sistema iría a encontrar el próximo segmento de EMPLEADO indiferente al departamento al cual el empleado fue asignado. Esta posibilidad puede prevenirse usando la orden GNP, como se demostrará en el Ejemplo 6, pero antes se verá una modificación simple a la consulta del Ejemplo 4, que permite la recuperación de todos los segmentos de empleados en la base de datos.

Ejemplo 5: Recuperar todos los segmentos de un tipo particular

```
GU DEPARTAMENTO
   EMPLEADO
MAS GN EMPLEADO
   GOTO MAS
```

Mientras exista otra ocurrencia de un segmento EMPLEADO, se ejecutará el ciclo identificado por la etiqueta de instrucción MAS.

Get Next Within Parent (GNP)

El GNP difiere del GN en que IMS sólo recupera los segmentos que son dependientes en una simple ocurrencia padre. Volviendo nuevamente al Ejemplo 4, esa consulta podría modificarse usando el GNP, como en el Ejemplo 6.

Ejemplo 6: Recuperar segmentos para un solo padre

```
GU DEPARTAMENTO (NOMBRE-DPTO = 'Mercadotecnia')
   EMPLEADO
GNP EMPLEADO
```

En este ejemplo, si no hubiera otro segmento de EMPLEADO en el Departamento de Mercadotecnia, la ejecución pararía y el usuario tendría la restricción deseada para los empleados del Departamento de Mercadotecnia.

Como otro ejemplo, suponga que se buscan todos los segmentos de OFICIO para el empleado Steve Smith, que trabaja en el Departamento de Mercadotecnia. Se aplicaría la siguiente orden:

Ejemplo 7: Recuperar segmentos para un solo padre

```
GU DEPARTAMENTO (NOMBRE-DPTO = 'Mercadotecnia')
   EMPLEADO (NOMBRE-EMP = 'Steve Smith')
   OFICIOS
PROXIMO GNP OFICIOS
   GOTO PROXIMO
```

La orden GU recupera el primer segmento de OFICIOS para Steve Smith. La orden GNP entonces recupera secuencialmente los restantes segmentos de OFICIO para Steve Smith.

Get Hold

La orden GET HOLD puede ocurrir en una de las tres formas: GET HOLD UNIQUE (GHU), GET HOLD NEXT (GHN) y GET HOLD WITHIN PARENT (GHNP). El programador usa estas órdenes de la misma forma que GU, GN y GNP, excepto que las órdenes GET HOLD deben usarse para informar a la base de datos que un cambio o una eliminación se realizará en el segmento recuperado. Esto significa que, GHU, GHN y GHNP se usan conjuntamente con las órdenes REPLACE (reemplazar) o DELETE (borrar).

Replace (REPL)

Para modificar un segmento existente, éste debe transferirse al área de trabajo, donde se realizan los cambios deseados a los campos de los segmentos. Usando el lenguaje *DL/1*, el segmento objetivo debe recuperarse usando primero una de las órdenes GET HOLD. El segmento se modifica entonces y la orden REPL escribe el segmento actualizado. El siguiente ejemplo ilustra cómo se procedería si se desea cambiar el salario de Irving Schatz, quien es empleado del Departamento de Mercadotecnia, de $20.000 a $25.000.

Ejemplo 8: Modificar valores de campos de segmentos

```
GU DEPARTAMENTO (NOMBRE-DPTO = 'Mercadotecnia')
   EMPLEADO (NOMBRE-EMP = 'Irving Schatz')
   MOVE 25000 TO SALARIO
   REPL
```

Delete (DLET)

Un segmento se borra primero del segmento objetivo usando una instrucción GET HOLD y luego usando la instrucción DLET. Suponga que Irving Schatz abandona la firma. Se aplicaría el Ejemplo 9.

Ejemplo 9: Borrar un segmento. Las órdenes

```
GU DEPARTAMENTO (NOMBRE-DPTO = 'Mercadotecnia')
   EMPLEADO (NOMBRE-EMP = 'Irving Schatz')
DLET
```

borrarán el segmento EMPLEADO, para Irving Schatz, de la base de datos. Cuando un segmento se borra, cualquier segmento dependiente también se borra, por lo que los segmentos OFICIO y ASIGNADO A de Schatz también se borrarían.

Insert (INSRT)

Los nuevos segmentos se añaden a la base de datos usando la orden INSRT. Los valores de los campos asociados primero se mueven a un área de trabajo. Entonces éstos se enlazan con los nombres de los segmentos padres relacionados con ellos. Así, si el segmento que se inserta es un segmento dependiente, el segmento padre ya debe existir.

Bob Lee ha terminado su currículum de Ingeniería en un colegio local. Se desea añadir un nuevo segmento OFICIO para Bob Lee en la base de datos.

Ejemplo 10: Añadir un segmento

```
MOVE 598 TO COD-OFICIO
MOVE 'INGENIERO DE REDACCIÓN' TO OFICIO-NOMBRE
MOVE 0 TO AÑOS-SERV
INSRT DEPARTAMENTO (NOMBRE-DPTO = 'Ingeniería')
EMPLEADO (NOMBRE-EMP = "Bob Lee')
OFICIO
```

El segmento se insertará como el último segmento de oficio debajo del nombre de Bob Lee.

▼ La evaluación del modelo de datos jerárquico

En esta sección se examinan las fortalezas y debilidades del modelo de datos jerárquico. Veamos la representación y la manipulación de los datos.

La representación de los datos

Existen tres características que definen la estructura del modelo de datos jerárquico: árboles, segmentos y campos de segmentos. Debido a que cualquier modelo orientado a objetos puede transformarse en un modelo de datos jerárquico, el requisito de que todos los registros de la base de datos sean árboles puede provocar duplicación de segmentos. Cualquier situación cuyos resultados naturales se enlazan en un segmento que sea un segmento hijo de dos segmentos padres distintos requiere que estos segmentos padres se encuentren en árboles separados.

Aunque tal duplicación elimina ciertas dificultades de implementación, tiene estos resultados negativos:

1. El espacio de almacenamiento se usa ineficientemente debido a que el segmento está repetido.
2. Se crea la posibilidad de inconsistencia de los datos. Si los datos se cambian en una copia del segmento, pero no en la otra, la base de datos es inconsistente.

Este problema se ha eliminado con el uso de los *segmentos virtuales* y los punteros. Un segmento virtual no contiene los datos, pero tiene un puntero a un segmento de datos físicos donde se almacenan los datos. Cuando se requiere un segmento, se reproduce en dos o más árboles de la base de datos, los datos verdaderos se almacenan en sólo uno de estos árboles. Todas las otras instancias de los datos contendrán un puntero a la localización donde se almacenan los datos verdaderos. Refiérase a la Figura 16.16. Note que el puntero mostrado por la línea discontinua apunta a la raíz del árbol que contiene los datos verdaderos en un segmento dependiente.

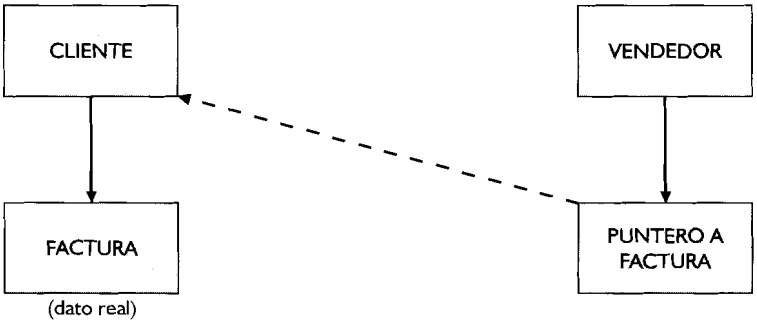
Una limitación importante del modelo jerárquico es que existen muchas aplicaciones, para las cuales un árbol no es la estructura de datos natural. Se ha visto esto, por ejemplo, en las manipulaciones que se requieren para convertir en dos estructuras de árbol una situación donde un segmento hijo pertenece de manera natural a dos padres. Estos tipos de ocurrencias pueden generar muchos árboles y una ineficiencia asociada al uso del espacio de almacenamiento.

Para las aplicaciones que son inherentemente jerárquicas por naturaleza y para las cuales las transacciones de consultas son estables, el modelo jerárquico puede ser bastante satisfactorio. El hecho que existan cerca de 7.000 instalaciones de IMS apoya esta conclusión, aunque algunas están cambiando a sistemas de bases de datos relacionales.

El lenguaje de manipulación de datos

Generalmente, la interfaz del lenguaje proporcionó al modelo jerárquico discrepancias con el vendedor. Esto significa que los programadores necesitan ser conscientes de qué interrelaciones deben estar predefinidas por el sistema y cuáles no. Esta confianza en el programador no es totalmente satisfactoria, debido a que el programador puede no ser entendido en todos los requisitos que han sido construidos en el sistema. Además, si se reorganiza la base de datos jerárquica, eso puede tener efecto negativo en la ejecución de programas, debido a que la estructura que soporta la aplicación *A* puede no ser la mejor estructura para soportar la aplicación *B*.

En el lado positivo, debido a que IMS domina las implementaciones jerárquicas, un programador que ha trabajado con IMS puede estar totalmente familiarizado con el trabajo interno del lenguaje *DL/1*. Además, para el uso estable de la base de datos, la reorganización puede ser secundaria.



La nomenclatura del modelo jerárquico puede resumirse en la siguiente forma:

1. Una estructura de datos jerárquica se construye a partir de un grupo de segmentos.
2. Cada estructura de datos jerárquica debe tomar la forma de un árbol. Un árbol se caracteriza por una o más interrelaciones padre-hijo, donde cada interrelación es uno-muchos. Cada árbol tendrá un segmento que funciona sólo como un segmento padre. Este segmento se llama raíz.
3. Todas las interrelaciones padre-hijo en un árbol se extienden a partir de la raíz hacia abajo.
4. Cualquier segmento en un árbol, exceptuando el segmento raíz, es un segmento hijo de algún segmento padre. Si se sigue un camino de interrelaciones padre-hijo desde un segmento de nivel superior, H, hasta un segmento de nivel inferior, S, todos los segmentos en ese camino, incluyendo S, son segmentos dependientes (también descendientes) de *H*.
5. El segmento no puede aparecer en más de un camino a partir del segmento raíz. Si en el modelo natural resulta en más de un camino, debe crearse un árbol separado para cada camino adicional.

El modelo de datos jerárquico continúa jugando un importante papel en las implementaciones de bases de datos. Es especialmente corriente en grandes centros de procesamiento de datos que son soportados por grandes computadores de IBM.

Una base de datos jerárquica se construye a partir de colecciones de segmentos que se conectan unos con otros por punteros. Cada segmento es una colección de campos, cada uno de los cuales contiene un solo valor de dato. Un puntero establece los enlaces lógicos necesarios entre dos segmentos. En este sentido, el modelo jerárquico es muy similar al modelo en red, donde los datos y las interrelaciones se representan por tipos de registros y punteros.

El modelo jerárquico difiere en que los segmentos se organizan como colecciones de árboles (un solo segmento padre permitido) más bien que como grafos arbitrarios (donde se permite más de un segmento padre).

Algunas limitaciones del modelo jerárquico son:

1. Las características lógicas y físicas del modelo no están claramente separadas.
2. Se requieren manipulaciones para representar interrelaciones de datos no jerárquicos.
3. Los requisitos de consultas *ad hod* pueden requerir la reorganización de la base de datos.

A partir del punto de vista del procesamiento eficiente de una base de datos, el modelo de datos jerárquico es competitivo. Sin embargo, como líderes de negocios que utilizan crecientemente la información como una herramienta estratégica, el modelo de datos jerárquico al parecer no proporciona toda la capacidad necesaria.

(Las preguntas marcadas con un asterisco son opcionales.)

1. Defina cada uno de los siguientes términos con sus propias palabras:
   a. IMS
   b. TDMS

c. MARK IV
d. System 2000
e. modelo de datos jerárquico
f. árbol
g. ocurrencia del segmento
h. ancestro
i. registro de la base de datos
j. segmentos gemelos
k. tipo IPH
l. árbol de ocurrencia jerárquico
m. recorrido pre-orden
n. HSAM
o. HISAM
p. HDAM
q. HIDAM
r. DL/1
s. plantilla de registro
t. puntero actual
u. bandera de estado

2. Defina la función que realizan cada una de las siguientes órdenes del DL/1
GU
GN
GNP
GHU
GHN
GHNP
ISRT
REPL
DLET

3. Brevemente analice el DBD, su estructura y su función.

4. Describa la composición y la función del BEP.

5. ¿Cuál es el propósito de crear un registro de intersección para corresponder los modelos orientados a objetos con los modelos jerárquicos?

6. En IMS, ¿cuántos subárboles pueden existir en un árbol?

7. * Compare los métodos de representación de los datos de los modelos de datos relacional, en red y jerárquico.

8. *¿Piensa que el lenguaje DL/1 es más simple que el DBTG DML? Justifique su respuesta.

9. ¿Qué ventajas ofrece IMS en comparación con el modelo DBTG y el modelo relacional?

1. Enlace los siguientes términos con sus definiciones:

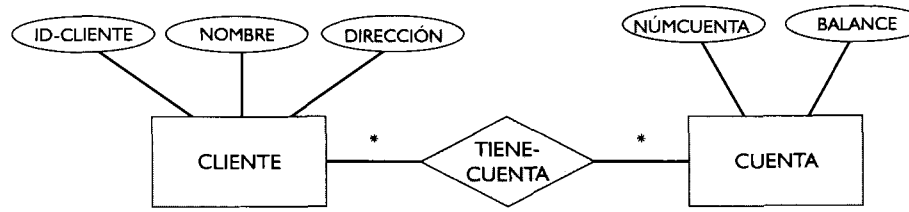
—*tipo del segmento*

a. La forma en que los datos se almacenan físicamente en el IMS.

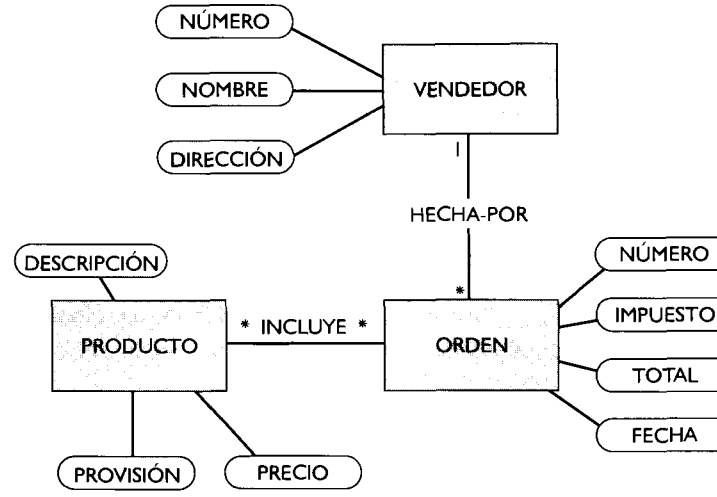
—interrelación padre-hijo
—DBD
—BEP
—BCP
—segmento susceptible
—segmento raíz
—segmento hoja

b. Una componente del BEP.
c. Una colección de BCPs.
d. Segmento que puede ser accedido por un programa de aplicación.
e. Un tipo de segmento que no participa como un segmento hijo en ningún IPH.
f. Un tipo de segmentos que no tiene tipos de segmentos hijos.
g. Corresponde a un objeto en el modelo orientado a objetos.
h. Interrelación lógica entre un tipo de segmento padre y un tipo de segmento hijo.

2. Exponga cualquier regla del capítulo que usaría en la transformación del diagrama orientado a objetos de la figura 16.1E en una estructura de datos jerárquica. Muestre la transformación resultante.



3. Haga corresponder el diagrama orientado a objetos de la Figura 16.2E con una estructura de datos jerárquica.



4. Escriba una DBD que se corresponda con la estructura de datos del problema 3.

5. Escriba un BEP para permitir que la DBD sea accedida por una aplicación que use sólo los segmentos VENDEDOR y ORDEN (ver problema 4).

6. Para la estructura de datos jerárquica del problema 3, escriba las órdenes del DL/1 para hacer lo siguiente:
   a. Recuperar el registro para el vendedor 13.
   b. Recuperar todas las órdenes con la cantidad de $100,00.
   c. Borrar la orden 256.
   d. Cambiar la dirección para el vendedor 13 a 'Adams Street 912. Gainesville,FL'.
   e. Añada el vendedor 15 a la base de datos. El nombre del vendedor 15 es "Mike Otteson' y la dirección es Buster Building, Suite 95, Toronto, Canadá.
   f. Añada la factura #285 a la base de datos.
   g. Borre la factura #842

1. Escriba un pequeño informe comparando las ventajas y desventajas del modelo de datos jerárquico. Compárelo y contrástelo con los modelos relacional y en red.

2. Encuentre una firma comercial que use el IMS. ¿Los usuarios están satisfechos con sus capacidades? ¿Cómo se usa? ¿Está la firma planificando cambiar?

3. Si fuera responsable con las recomendaciones desarrolladas para mejorar el modelo jerárquico, ¿qué sugerencias incluiría?

4. ¿Puede refinar y mejorar cualquiera de las reglas sugeridas para la transformación de los modelos orientados a objetos en estructuras de datos jerárquicas? ¿Cuáles? ¿Cuándo se aplican?

GLOSARIO

A

acceso sólo para recuperar. Acceso a la base de datos pero sin poder hacer actualización

administración de la base de datos. Personal con la responsabilidad de controlar y proteger la base de datos.

administrador de archivos. Software que administra la reservación de localizaciones de almacenamiento y estructuras de datos.

administrador de buffer. Software que controla el movimiento de datos entre la memoria principal y el almacenamiento en disco.

administrador de la base de datos. Personas cuyas responsabilidades se centran en la gestión de los aspectos técnicos del sistema de base de datos.

administrador de los datos. Persona cuyas responsabilidades se centran en el estableeimiento de las políticas y los procedimientos para el sistema de información de la organización.

agente. Un proceso que coopera para ejecutar una transacción.

agregado. Una interrelación vista como un conjunto de objetos.

agrupamiento (*clustering*). Ubicación en un mismo bloque de relaciones que se reúnen con frecuencia.

álgebra relacional. Un lenguaje procedimental para la manipulación de relaciones.

alias. Un nombre alternativo que se le da a una relación.

ancestro. En una jerarquía, un tipo de segmento que se encuentra en el mismo camino en un nivel superior en el árbol.

anomalías de actualización. Inconsistencia de los datos como resultado de datos redundantes y actualizaciones parciales.

anomalías de borrado. Pérdida no intencionada de datos debido a que se han borrado otros datos.

anomalías de inserción. Imposibilidad de adicionar datos en la base de datos debido a la ausencia de otros datos.

Application painter. Un painter que se usa para definir los aspectos generales de una aplicación, tales como el nombre y la biblioteca de la aplicación.

árbol. Una estructura de datos jerárquica, esto es, una estructura en red donde un tipo de segmento hijo es enlazado a sólo un tipo de segmento padre.

árbol de la base de datos. Un árbol que tiene una raíz.

árbol de oeurrencia jerárquico. Representación de segmentos de ocurrencias en una estructura de árbol que refleja todos los tipos IPH.

árbol enraizado. Una jerarquía de registros índices que tiene un único registro índice en el nivel más alto; este registro se llama raíz.

área. En IDMS/R es una localización en el almacenamiento que contiene uno o más tipos de registro.

área de trabajo del usuario. Porción de la memoria primaria que se usa para contener las variables de programa que apuntan y contienen los registros de los diferentes tipos de registro, mientras que sus contenidos se operan por el programa anfitrión o principal.

arista. Parte de una estructura en red representada por una flecha.

arquitectura tres-niveles. Estructura estándar de base de datos consistente en los niveles conceptual, externo e interno.

asignación. Operación del álgebra relacional que da un nombre a una relación.

asociar (*map*). Asociar elementos de una esfera con elementos en otra esfera.

atributo. Una interrelación funcional de un conjunto de objetos con otro conjunto de objetos.

atributo de la relación. Una columna en una relación.

B

banderas de estado. Variables que se usan para denotar el éxito o el fracaso de la última operación ejecutada por el programa de aplicación.

base de datos. Una colección de datos interrelacionados que se puede utilizar por uno o más programas de aplicación.

base de datos centralizada. Una base de datos que está físicamente confinada en una única localización.

base de datos distribuida. Una base de datos que está distribuida entre una red de localidades separadas geográficamente.

base de datos externa. La vista del usuario de los datos en el IMS.

bloque de comunicación de programa (BCP). Un componente del BEP.

bloque de especificación de programa (BEP). Especifica los nombres de cada segmento en una aplicación de programa que accederá, corresponde a la vista de usuario o subesquema.

bloqueo. Impide el acceso a un registro de la base de datos por una segunda transacción hasta que la primera transacción haya completado todas sus acciones.

bloqueo de escritura. El usuario tiene derecho a leer y a actualizar el registro dado.

bloqueo de lectura. El usuario tiene derecho a leer el registro dado.

bloqueo en dos fases. Un método de control del procesamiento concurrente en el cual todas las operaciones de bloqueo preceden a la primera operación de liberación.

Botón Open Table. (Abrir Tabla.)

Botón Run Query. (Ejecutar consulta.)

Boyce-Codd (FNBC). Todo determinante es una clave.

BREAK. Palabra reservada que causa la salida de la ejecución de un lazo WHILE.

C

caja de condición. En QBE es una caja en la cual se puede expresar una condición compleja de una consulta.

cálculo relacional. Un lenguaje no procedimental para la definición de soluciones a consultas.

camino. Un conjunto de punteros que llevan desde un registro índice hacia otro.

camino de expansión. El camino desde el origen a la unidad de referencia en el análisis AED.

aracteres comodines. Símbolos especiales que valen por cualquier cadena de caracteres.

cardinalidad. El número máximo de instancias de un conjunto de objetos que puede estar relacionado con una sola instancia del otro conjunto de objetos

catálogo. En SQL-92, una colección de esquemas con nombre.

centro de información. Un área en la que los usuarios tienen facilidades para hacer sus propios cómputos.

ciclo de vida del desarrollo de la base de datos (CVDBD). Un proceso para el diseño, implementación y mantenimiento de una base de datos.

ciclo de vida del desarrollo de sistemas (SVDS). Un proceso para el desarrollo de sistemas.

cifrado. Codificación de los datos para hacerlos ininteligibles a personas no autorizadas.

clase. Una representación abstracta (plantilla) de un conjunto de objetos.

clase de inserción de conjunto. Es la forma en DBTG por la cual un registro miembro se coloca en una ocurrencia de conjunto; puede ser manual o automática.

clase de retención de conjunto. Es lo que DBTG determina cómo y cuándo un registro miembro se puede quitar de un conjunto; puede ser obligatoria u opcional.

clase derivada. Una clase que hereda características de otra clase.

cláusula. En PROLOG, el medio por el cual se expresan los hechos y el conocimiento; <conclusión> :- <lista de hipótesis>.

cláusula FROM. Lista las tablas que son referidas por la consulta.

cláusula GROUP BY. Indica qué filas deben agruparse sobre un valor común de las columna(s) especificada(s).

cláusula HAVING. Una cláusula que impone condiciones a los grupos.

cláusula SELECT. Señala las columnas que se desean en la consulta.

clánsula WHERE. Nos da la condición para seleccionar las filas de las tablas indicadas.

clave. Campos de datos que identifican unívocamente un registro en un archivo.

clave. Un valor que siempre puede utilizarse para identificar unívocamente una instancia

clave candidata. Cualquier conjunto de atributos que puede ser elegido como una clave de una relación.

clave compuesta. Una clave compuesta de más de un atributo.

clave concatenada totalmente. Significado de la identificación de la localización de un segmento en la base de datos.

clave externa. Un conjunto de atributos lexicográficos cuyos valores siempre identifican una única instancia de objeto.

clave externa. Un conjunto de atributos en una relación que constituyen una clave en alguna otra (o posiblemente la misma) relación; usada para indicar enlaces lógicos entre relaciones.

clave externa recursiva. Una clave foránea que referencia a su propia relación.

clave intersección. Operación del álgebra relacional que crea un conjunto intersección de dos relaciones unión-compatible; candidata designada como uso principal para identificar unívocamente las filas de una relación.

clave secundaria. Un elemento de dato cuyo valor identifica a un conjunto de registros.

CODASYL (Conference on Data Systems Languages). Una organización compuesta de representantes de los fabricantes de hardware y software y de los usuarios; conocida principalmente por el desarrollo del lenguaje COBOL.

conjunto. En el modelo de DBTG, una interrelación uno-muchos entre dos tipos de registros.

completo relacionalmente. Que tiene el mismo poder lógico que el álgebra o el cálculo.

conectores booleanos. AND, OR, NOT.

conjunto de objetos. Un conjunto de cosas de la misma clase.

conjunto de objetos abstractos. Un conjunto de objetos que consiste de las instancias que no se pueden imprimir.

conjunto de objetos agregado. Una interrelación vista como un conjunto de objetos.

conjunto de objetos físicos. Un conjunto de objetos cuyas instancias son objetos físicos.

conjunto de objetos léxico. Un conjunto de objetos que consiste de las instancias que se pueden imprimir.

conjunto solución. Un conjunto de valores de datos de la base de datos que satisface las condiciones de una consulta.

conocimiento de direcciou-empresa. Conocimiento que ayuda a una empresa a tomar decisiones.

conocimiento específico de la aplicación. Conocimiento que se determina por las reglas y las convenciones que se aplican a un dominio específico de problema.

conocimiento estructural. Conocimiento sobre dependencias y restricciones entre los datos.

conocimiento en extenso. Hechos que se almacenan en las relaciones de la base de datos.

conocimiento general procedimental. Conocimiento que sólo se puede describir por un procedimiento.

conocimiento en intenso. Conocimiento que se deduce del conocimiento en extenso por la aplicación de reglas.

consulta disyuntiva. Una consulta cuyas condiciones están conectadas por un "or", o algún símbolo equivalente.

consulta externa. La consulta principal que contiene a las subconsultas.

consulta multi-tabla. Una consulta que involucra a más de una tabla.

consulta simple. Una consulta que involucra a una sola tabla de la base de datos.

CONTINUE. Palabra reservada que causa que el control de ejecución de un bucle WHILE retorne a la primera instrucción del bucle.

control. Un dispositivo gráfico por el cual el usuario interactúa con el sistema.

control de integridad (restricción). Una restricción aplicada a un conjunto determinado de datos; utilizado para minimizar los errores en la entrada de los datos.

controles de acceso. Controles que limitan el acceso del usuario a los programas y a los datos.

controles de concurrencia. Controles que mantienen la integridad de la base de datos cuando dos o más usuarios simultáneamente solicitan un registro de la base de datos.

controles de respaldo y recuperación. Aquellos controles que proporcionan medios para restaurar la base de datos en caso de fallos del sistema.

coutroles de vista. Aquellos controles que restringen el acceso a las vistas (subconjuntos de las relaciones de la base).

cuantificador existencial. Expresión del cálculo relacional que afirma la existencia de al menos una fila a la cual se le aplica la condición.

cuantificador universal. Expresión del cálculo relacional que establece que una cierta condición se le aplica a *toda* fila de algún tipo.

cuarta forma normal (4FN). Una relación que está en 3FN y no tiene dependencias multievaluadas.

cursor. Una facilidad del SQL incrustado en la que se almacenan los resultados de una consulta SQL para su ulterior procesamiento.

D

DataWindow painter. Un *painter* que construye objetos DataWindow que acceden a la base de datos y ponen los resultados en ventanas.

datos. Hechos aislados.

datos globales. Datos que se mantienen en una base de datos cuya ubicación es diferente para al menos uno de los usuarios.

datos locales. Datos en mismo sitio o localidad con su propia base de datos.

DBGT (Database Task Group). Un subgrupo de CODASYL que tiene la responsabilidad de desarrollar los estándares de los sistemas de gestión de bases de datos.

definición de dominio. Un tipo de datos especializado definido dentro de un esquema y usado en las definieiones de columnas.

definición de esquema. Descripción de una base de datos para el SGBD.

definición de requisitos. Determinación de los requisitos de información de las áreas administrativas y funcionales.

DELETE. Operación que quita filas de una relación.

dependencia funcional. El valor de un atributo en una tupla determina el valor de otro atributo en la tupla.

dependencia multievaluada (DMV). Una restricción que garantiza la independencia mutua de atributos multievaluados.

dependencia transitiva. Aparece cuando un atributo no clave es funcionalmente dependiente de uno o más atributos no claves.

descomposición de relaciones. División de una relación en múltiples relaciones.

descripción de la base de datos (DBD). La manera en que los datos son almacenados físicamente en IMS (estructura interna de la base de datos).

determina funcionalmente. Determinar unívocamente un valor.

determinante. El atributo de la parte izquierda de una DF determina el valor de los otros atributos en una tupla.

diario de operaciones. Un registro de todas las transacciones y los cambios correspondientes en la base de datos.

diccionario de datos. Parte del SGDB que define la estructura de los datos del usuario y cómo éstos son usados.

diferencia. Operación del álgebra relacional que crea un conjunto diferencia de dos relaciones unión-compatible.

detección de interbloqueo. Una comprobación periódica del SGBD para determinar si la cola de espera para algunos recursos excede un límite predeterminado.

diseño conceptual. Creación del esquema de nivel conceptnal para la base de datos.

diseño conceptual de bases de datos. Identificación de los elementos de datos, interrelaciones y restricciones para una base de datos.

diseño físico de la base de datos. Determinación de los dispositivos de almacenamiento, métodos de acceso e índices para usar la base de datos.

disparador. Un programa que se ejecuta automáticamente cuando se intenta hacer una determinada actualización sobre una tabla específica.

DISTINCT. Operador que elimina las filas duplicadas.

Distributed INGRESS. Un SGBDD comercializado por Relational Technology.

dirigida-por-eventos. Una caracteristica de la aplicación que significa que se escriben *scripts* para responder a los eventos que tienen lugar.

división. Operación del álgebra relacional que crea una nueva relación seleccionando las filas en una relación que se corresponden con *todas* las fila en otra relación.

división de correspondencia (*mapping division*). La porción del subesquema del DBTG que permite cambiar los nombres que se usan en el esquema por los nombres escogidos para el subesquema.

división de estructura. La porción del subesquema del DBTG donde se definen los registros, los elementos de datos y los conjuntos del esquema.

división de título. La porción del subesquema del DBTG que permite dar nombre a los subesquemas y sus esquemas asociados.

L/I (Data language I). Lenguaje de manipulación de datos del IMS.

dominio. El término PROLOG para la especificación de un tipo de dato.

dominio del atributo. El conjunto de valores que puede tomar un atributo.

dueño de un esquema. Persona que tiene autoridad y responsabilidad para otorgar acceso a tablas, columnas y vistas en un esquema de base de datos.

E

ejecución en serie. Acciones que se ejecutan una después de otra; no hay acciones en paralelo.

elemento ejemplo. Una variable en QBE que representa un valor no especificado en una columna de una tabla.

encadenamiento hacia delante. Una cadena de reglas que van de una hipótesis a una conclusión.

encadenamiento hacia detrás. Una cadena lógica de reglas que van de una conclusión a una hipótesis.

enfoque orientado a datos. Centra la atención en el análisis de los datos utilizados por las funciones.

enfoque orientado a funciones. Ver a un sistema desde la perspectiva de las funciones que debe realizar.

enlace. Canales de comunicación entre dos sitios de la red y que proveen las capacidades de transferencia de datos.

enlace físico. Un medio de conectar los registros usando las direcciones de registros en disco.

equirreunión. Reunión theta basada en la igualdad de columnas específicas.

esquema. Una definición de la estructura lógica de la base de datos completa.

esquema de información. Esquema en un catálogo que contiene metadatos.

especialización. Un conjunto de objetos que es un subconjunto de otro conjunto de objetos

especificación de consulta. Definición de una consulta que se usa en una definición de vista, declaración de cursor u otra instrucción.

esquema de base de datos relacional. Un listado que muestra los nombres de las relaciones, los nombres de los atributos y las claves foráneas.

estudio de viabilidad. Parte del CVDBD que determina la viabilidad económica, tecnológica y operacional de la base de datos.

EXCEPT. Operación que crea el conjunto diferencia entre dos relaciones.

expresión de calificación. Una condición verdadera o falsa que se refiere a la lista resultado y que debe cumplirse para los elementos en el conjunto solución.

F

Forma Normal Dominio Clave (FNDLL). Requiere que cada restricción sea resultado de definiciones de dominios y claves.

formas normales. Reglas para relaciones estructuradas que eliminan anomalías.

formato cuenta-dato. Un formato de datos para pistas que no usan claves externas.

formato cuenta-clave. Un formato de datos para pistas que usan claves externas.

fragmentación de datos. La partición de una relación en un SBDD.

fragmentación horizontal. Partición de una relación en subconjuntos de sus tuplas.

fragmentación vertical. Particionar una relación por la proyección del subconjunto de sus atributos.

frontera de la eficiencia. La frontera exterior en el análisis AED.

función integrada. Una función estadística que opera sobre un conjunto de filas - SUM, AVG, COUNT, MAX, MIN.

función de conjuntos. Una función integrada.

G

generalización. Un conjunto de objetos que es un superconjunto de (o que contiene a) otro conjunto de objetos.

grado de la relación. El número de atributos en una relación.

grafo dirigido. Una estructura matemática en la que los puntos o nodos están conectados por flechas o aristas.

H

hereda. La propiedad de un conjunto de especialización que causa que éste tenga todos los atributos del conjunto generalizado.

hijo. Un registro "subordinado" en una interrelación jerárquica.

homónimo. Un término que tiene diferentes significados en diferentes contextos.

HDAM. Método de acceso del IMS, brinda acceso-directo de segmentos muy rápidos, pero no la capacidad de procesamiento secuencial.

HIDAM. Método de acceso del IMS, brinda tanto acceso directo a los segmentos raíces como recuperación secuencial.

HISAM. Método de acceso del IMS, brinda la capacidad para recuperaciones de segmentos tanto directa como secuencial.

hoja. El registro índice en el nivel más bajo de un árbol enraizado.

HSAM. Método de acceso del IMS, muy rápido para recuperaciones de segmentos secuenciales.

I

identificador. Una clave externa.

IDS (Integrated Data Store). Uno de los primeros sistemas de gestión de bases de datos; su arquitectura estuvo muy influenciada por las recomendaciones del DBTG para un modelo de base de datos en red.

IMS (IBM's Information Management System), SGBD basado en el modelo de datos jerárquico.

indicadores de actualización. Marcas de posición de los registros que han sido encontrados.

información. Datos organizados o resumidos.

información para la gestión. Información de apoyo a los que toman decisiones y a las operaciones dentro de una compañía.

instancia de un objeto. Un miembro particular de un conjunto de objetos.

integridad de los datos. La exactitud y la consistencia de los datos almacenados en el sistema de base de datos.

interbloqueo (*deadlock*). Dos transacciones se excluyen mutuamente del acceso al siguiente registro requerido para completar cada una de sus

transacciones; también se denomina "abrazo mortal".

instrucción FETCH. Una instrucción que toma una fila a partir de un cursor que ya ha sido abierto.

instrucción bloque. O una sola instrucción SQL sin delimitadores o un conjunto de dos o más instrucciones SQL delimitadas por BEGIN ... END.

instrucción IF. Una instrucción cuya ejecución depende de la veracidad de una condición.

instrucción OPEN cursor. Instrucciones SQL que causan que el SGBD procese una consulta y "almacene" el resultado en el cursor.

instrucción WHILE. Una instrucción que usa una condición para controlar la ejecución iterativa de un bloque de instrucciones.

instrucción condicional. Una instrucción que pregunta por una condición y determina el procesamiento a seguir según el cumplimiento de la condición.

instrucción CASE. Se usa con el SWITCH para indicar cuál es el procesamiento a seguir cuando una condición sea verdadera.

instrucción SWITCH. Una instrucción que permite preguntar por una serie de condiciones.

instrucción de calificación. Una condición en una instrucción del cálculo relacional que restringe la membresía en una relación solución.

instrucción iterativa. Una instrucción que se puede repetir una cierta cantidad de veces.

instrucciones de señalización (*flag statements*). Instrucciones SQL que se incrustan en un programa de aplicación para señalar dónde comienza o termina un conjunto de instrucciones SQL.

integración de los datos. Combinar los datos para un uso común.

integridad de los datos. Precisión y consistencia de los valores de los datos en la base de datos.

Interfaz Gráfica de Usuario (GUI). Pantallas y funciones que proporcionan al usuario final un medio gráfico para tener acceso al sistema de computación.

interoperabilidad. El estado que caracteriza a múltiples sistemas heterogéneos que se comunican y contribuyen a la terminación de una tarea común.

interrelación. Un enlace entre instancias de dos conjuntos de objetos.

interrelación binaria. Una interrelación entre dos conjuntos de objetos.

interrelación de alto nivel. Una interrelación entre tres o más conjuntos de objetos

interrelación funcional. Una interrelación que tiene una cardinalidad máxima de valor 1 en al menos una dirección.

interrelación n-aria. Una interrelación entre *n* conjuntos de objetos.

interrelación recursiva. Una relación que relaciona un objeto consigo mismo.

INSERT. Operación que causa que se añadan filas a una relación.

Interfaz Gráfica de Usuario (GUI). Pantallas y funciones que brindan medios gráficos para que un usuario final acceda a un sistema de cómputo.

INTERSECT. Operación que crea el conjunto intersección de dos relaciones.

J

JOIN ON. Operación que conecta las relaciones cuando ocurre una condición

JOIN USING. Operación que conecta las relaciones cuando las columnas comunes designadas tienen iguales valores.

L

lenguaje anfitrión (*host language*). Lenguaje de los programas en los cuales se pueden incrustar las instrucciones SQL.

Lenguaje de Definición de Datos (LDD) (*Data Definition Language*) (*DDL*). El lenguaje que se usa para especificar el esquema de una base de datos.

Lenguaje de Manipulación de Datos (LMD) (*Data Manipulation Language*) (*DML*). El lenguaje que se usa para almacenar y manipular los datos.

lenguaje flujo-de-control. Lenguaje de manipulación de datos de SQL Server.

lenguaje gráfico. Un lenguaje de computadoras que usa representaciones pictóricas para resolver los problemas.

lenguaje textual. Un lenguaje de computadoras cuyas instrucciones consisten en cadenas de caracteres.

lista invertida. Un directorio en donde cada entrada contiene un apuntador a todos los registros físicos con un valor específico.

lista resultado. Una lista en una instrucción del cálculo relacional que define los atributos de la relación solución.

literal cadena de caracteres. Literales formados por caracteres alfanuméricos y por caracteres "especiales".

lógica de primer orden. Una estructura lógica que se caracteriza por un conjunto de objetos, un conjunto de predicados (cada uno de los cuales evalúa verdadero o falso) y un conjunto de funciones.

M

marca de tiempo (*timestamp*). Un método para identificar los mensajes con la hora de transmisión.

MARK IV (Control Data Corporation's Multi-Access Retrieval System), SGBD basado en el modelo de datos jerárquico.

memoria principal. Almacén localizado en la unidad de procesamiento central; usado para hacer disponible los datos para las operaciones de los usuarios.

metadato. Información descriptiva sobre las bases de datos.

metadatos. Datos en el diccionario de datos que describen la base de datos.

modelo. Una representaeión de la realidad que retiene sólo detalles seleccionados.

modelo de datos. Un método conceptual para estructurar los datos.

modelo de datos jerárquico. Un modelo de dato en el cual todas las interrelaciones son estructuradas como árboles.

modelo de datos relaeional. Un modelo de datos donde los datos se representan en forma de tabla.

modelo de datos en red. Representa los datos en estructuras en redes de tipos de registro conectados en interrelaciones uno-uno o muchos-muchos.

modelo jerárquico. Un modelo de datos que presume que todas las interrelaciones entre los datos pueden estructurarse como jerarquías.

modelo orientado a objetos. Un modelo que representa las entidades del

mundo real como objetos en lugar de como registros.

modelo semántico. Un modelo que captura los significados de las entidades del mundo real y sus interrelaciones.

modo de inserción manual. Es lo que en DGTB requiere que un registro miembro sea puesto en un conjunto usando una orden CONNECT (conectar) para enlazar el registro con la ocurrencia de conjunto que se quiera.

modo de inserción automático. Es cuando en DBTG se crea un nuevo registro y el sistema lo conectará automáticamente con la ocurrencia de conjunto correcta.

muchos-muchos. Una cardinalidad de la interrelación que es mucho en ambas direcciones

multiconjunto. Un conjunto que puede tener entradas duplicadas.

N

NATURAL JOIN. Operación que concecta las relaciones cuando las columnas comunes tienen iguales valores.

nivel conceptual. Nivel estructural de la base de datos que define el esquema lógico de la base de datos.

nivel externo. Nivel estructural de la base de datos que define las vistas de los usuarios.

nivel interno. Nivel estructural de la base de datos que define la vista física de la base de datos.

no procedimental. Lenguaje que proporciona un medio de establecer *qué* se desea en lugar de *cómo* hacerlo.

nodo. Parte de una estructura en red representada por un punto.

normalización. El proceso de conversión de una relación en una forma estándar.

O

objeto complejo. Corresponde a un agregado o interrelación de alto nivel.

objeto conceptual. Un objeto que representa un tipo de cosa.

objeto físico. Un objeto que representa una cosa física específica.

ocurrencia. Sinónimo de instancia.

ocurrencia del segmento. Terminología del IMS para ocurrencias de registros.

operador EXISTS. Evalúa verdadero si el conjunto resultante es no vacío.

operador NOT EXISTS. Evalúa verdadero si el conjunto resultante es vacío.

operadores de comparación. =, <>, <, >, <=, >=

órdenes de actualización de conjuntos. Órdenes del LMD del DBTG que se usan para crear, cambiar o borrar conjuntos instancias.

órdenes de actualización de registros. Órdenes del LMD del DBTG que se usan para cambiar los valores de los registros de la base de datos.

órdenes de recuperación. Órdenes del LMD del DBTG que se usan para recuperar registros de la base de datos.

órdenes navegacionales. Una orden del LMD del DBTG que se usa para encontrar registros de la base de datos.

P

padre. Un registro "dueño" en una interrelación jerárquica.

PARADOX PARA WINDOWS. Una base de datos para microcomputadoras cuyo lenguaje de consulta es QBE.

parámetro. Una variable que se usa para pasar o recibir datos de un procedimiento almacenado.

planificación de la base de datos. Esfuerzo estratégico para determinar las necesidades de información de un extenso período de tiempo.

planificación preliminar. Planificación de una base de datos que ocurre durante el proceso de planificación estratégica de la base de datos.

plantilla de segmento. El formato del área de trabajo del programa para un segmento.

plantillas de registro (*record templates*). Formatos usados por los registros que se leen en el área de trabajo del usuario.

plataforma cliente/servidor. Una red local que consiste en computadoras clientes, las cuales reciben servicios de una computadora servidor.

PL/SQL. Lenguaje de manipulación de datos de Oracle.

predicado. Una expresión que evalúa verdadero o falso.

preview (*vista preliminar*). Un recurso que permite al que desarrolla la aplicación ver cómo aparecerá la DataWindow cuando se ponga en una ventana.

primera forma normal (1FN). Todos los valores de los atributos deben ser atómicos.

procedimental. Lenguaje que provee un método paso-por-paso para la solución de problemas.

procedimiento. Instrucciones escritas que describen los pasos necesarios para realizar una tarea determinada en un sistema.

procedimiento almacenado. Un programa compilado a lenguaje de máquina y que se salva para su ejecución repetida más eficiente.

procedimientos de recuperación de la base de datos. Los medios mediante los cuales una base de datos que se ha corrompido por desperfectos puede restablecerse a un estado correcto y consistente.

procesamiento concurrente (concurrencia). Ocurre cuando dos o más transacciones piden acceso concurrentemente a un mismo registro de la base de datos aproximadamente al mismo tiempo.

procesamiento con acceso directo. Un método de acceso a un archivo que permite el acceso directo a un registro específico.

procesamiento electrónico de datos. Automatización computacional del papeleo al nivel operacional de una organización.

producto. Operación del álgebra relacional que crea el producto cartesiano de dos relaciones.

producto cartesiano. Resultado de aparear cada fila de una tabla con *todas* las filas de otra tabla

profesionales de la computación. Personas responsables del sistema de base de datos y del paquete de programas de aplicación asociado.

programa de aplicación. Un programa de computadora que realiza una tarea específica de valor práctico en una situación de negocios.

propiedad importante. Una propiedad de un SGBD que no es obligatoria pero hace al SGBD más atractivo.

propiedad indeseable. Una propiedad de un SGBD que le resta valor al SGBD para la firma.

propiedad innecesaria. Una propiedad de un SGBD que no contribuye en nada al valor del SGBD para la firma.

propiedad obligatoria. Una propiedad de un SGBD que tiene que ser proporcionada.

propiedad opcional. Una propiedad de un SGBD que tiene una importancia secundaria; puede ayudar a distinguir entre otros SGBD igualmente atractivos.

protocolo de cierre de dos-fases. Un protocolo que consiste en una fase de preparado-para-el-cierre y una fase de votación (cerrar o abortar.

proyección. Relación que resulta de una operación de proyectar.

proyección de una relación. Una relación compuesta de atributos seleccionados de otra relación.

proyectar. Operación del álgebra relacional que crea una relación borrando columnas de una relación existente.

pruebas de evaluación. Un método de comparación del rendimiento de un SGBD mediante pruebas sobre aplicaciones reales.

puntero. Una dirección física que identifica dónde puede encontrarse un registro sobre el disco.

puntero actual. Contiene la dirección del último segmento que se accedió en el árbol.

punto de expansión. Cualquier punto en el camino de expansión en el análisis AED.

punto de frontera. Cualquier punto sobre la frontera en el análisis AED.

puntos de comprobación. Copias de la base de datos en momentos predeterminados durante el procesamiento; la recuperación de la base de datos comienza o termina en el punto de chequeo más reciente.

Q

quinta forma normal (5FN). Una forma normal que elimina las dependencias de unión.

R

recorrido pre-orden. Un método de conversión de una estructura de árbol a un archivo plano que contiene la información necesaria sobre las interrelaciones jerárquicas.

red. Una interrelación de datos en la cual un registro puede estar subordinado a registros de más de un archivo.

red de área amplia (WAN). Una red de computadoras en la que los sitios están bastante dispersos geográficamente.

red de área local (LAN). Una red de computadoras en la que los sitios están ubicados a corta distancia (por lo general menos de una milla) uno de otro.

redundancia de datos. Repetición de datos en una base de datos.

registro físico. Un bloque físico de datos.

reunión. Operación del álgebra relacional que conecta relaciones.

reunión externa. Expansión de la reunión natural que incluye *todas* las filas de ambas relaciones.

reunión natural. Operación de reunión que conecta relaciones cuando las columnas comunes tienen iguales valores.

reunión theta. Operación de reunión que conecta relaciones cuando los valores de determinadas columnas tienen una interrelación específica.

registro de la base de datos. Una ocurrencia de una raíz y todas sus tipos de segmentos dependientes.

registro lógico. Un tipo de registro como se ve desde la perspectiva del usuario.

regla. Una restricción sobre el valor que se permite en una columna, establecida por una expresión condicional.

regla de integridad de la entidad. El atributo que es clave de una fila no puede ser nulo.

regla de integridad referencial. El valor no nulo de una clave foránea debe ser un valor real de la clave de otra relación.

relación. Una tabla de dos dimensiones que contiene filas y columnas de datos

relación de intersección. Una relación que representa instancia donde otras dos relaciones se encuentran en una interrelacion muchos-muchos

restricción (*constraint*). Una regla que restringe los valores en una base de datos.

restricción CHECK. Una restricción general, basada en una expresión condicional, que se impone sobre una columna o sobre una tabla.

restricción de columna. Una restricción que se establece en la definición de una columna en una tabla.

restricción de tabla. Una restricción que se aplica simultáneamente a múltiples columnas de una tabla.

restricción UNIQUE. Una restricción que garantiza que dos tuplas de una misma relación no tengan el mismo valor en una columna.

restricción de valor. Una regla que define los valores permisibles para un dato en específico.

retención fija. Es cuando en DBTG, una vez que un registro miembro ha sido asignado a una ocurrencia de un conjunto, éste debe siempre estar en alguna ocurrencia de dicho conjunto.

retención opcional. Es cuando en DBTG no se han impuesto restricciones sobre las conexiones o reconexiones a tipos de conjunto.

retención obligatoria. Es cuando en DBTG, una vez que un registro miembro ha sido asignado a una ocurrencia de conjunto, éste no se puede quitar del conjunto a menos que el registro se borre de la base de datos.

retículo complejo. Una estructura de datos en la cual las interrelaciones binarias son muchos-muchos.

retículo simple. Una estructura de datos en la cual todas las interrelaciones binarias son uno-muchos.

R*. Un SGBDD comercializado por IBM.

S

sección de conjuntos. La sección del esquema del DBTG que define los conjuntos e incluye el tipo de registro dueño y los tipos de registros miembros.

sección de esquema. La sección del esquema del DBGT que nombra al esquema.

sección de registro. La sección del DBGT que define cada registro, sus elementos de datos y sus localizaciones.

sección de subesquema de conjuntos. La sección de la división de la estructura que define los conjuntos que se van incluir.

sección de subesquema de registros. La sección de la división de la estructura que especifica los registros de los subesquemas, los elementos de datos y los tipos de datos.

SEGM. Una instrucción del IMS que define los campos que se incluyen en un segmento para usarse por un programa.

segmento hoja. En un árbol, cualquier tipo de segmento que no tiene tipos de segmentos hijos.

segmento raíz. En un árbol, un tipo de segmento que no participa como un segmento hijo en ningún IPH.

segmento susceptible. Un segmento que es accesible para un programa.

segmentos gemelos. Ocurrencias de tipo de segmentos que tienen la misma ocurrencia de tipo de segmento padre (más de dos ocurrencias de segmentos pueden calificarse como gemelas).

segunda forma normal (2FN). Los atributos no claves no pueden ser funcionalmente dependientes de una parte de la clave.

seguridad de los datos. Se refiere a la protección del sistema de base de datos ante usos mal intencionados o no autorizados.

selección. Operación del álgebra relacional que usa una condición para seleccionar filas de una relación.

selector de estrategia. Software que transforma una consulta del usuario en una forma efectiva para la ejecución.

servidor de bases de datos. Un programa que corre en una computadora servidor para dar servicios de bases de datos a las máquinas clientes.

símbolos predicados. Nombres que se aplican a los argumentos para expresar un predicado.

sinónimo. Términos que significan lo mismo.

sistema basado en el conocimiento. Otra forma de nombrar a un sistema de base de conocimientos.

sistema de base de conocimientos. Un sistema que provee el rango completo de capacidades para almacenamiento y manipulación de datos, así como facilidades para la creación, almacenamiento y ejecución de reglas de inferencia en tablas de datos almacenados.

sistema de base de datos. Una base de datos, un sistema de gestión de bases de datos, con el *hardware* y el personal apropiados.

sistema de base de datos distribuida. Un sistema de base de datos compuesto de varios sistemas en sitios locales, conectados por líneas de comunicación.

sistema de base de datos orientado a objetos. Sistemas de bases de datos que pueden implementar modelos conceptuales directamente y que pueden representar complejidades que van más allá de las capacidades de los sistemas relacionales.

sistema de gestión de base de conocimientos. Sistema de software que soporta el rango usual de las funciones de los SGBD, así como que gestiona el proceso deductivo de la base de reglas que opera sobre la base de hechos.

sistema de gestión de bases de datos (SGBD). Un sistema computacional que facilita la gestión de las bases de datos.

sistema de gestión de bases de datos distribuida (SGBDD). El software que gestiona el sistema de bases de datos distribuida.

sistema de información. Sistema automatizado que organiza los datos para producir información.

sistemas de información de gestión. Sistema automatizado cuyo foco de atención es la información para el nivel de dirección intermedio.

sistema de información para la gestión. Sistema que provee información para la gestión o la administración.

sistema de procesamiento de datos. Un sistema automatizado para procesar los datos de los registros de una organización.

sistemas abiertos. El concepto de conectar una variedad de computadoras con diferentes *hardware* y *software* para trabajar coordinadamente con el fin de lograr los objetivos del usuario.

sistemas de soporte para la toma de decisiones. Sistema automatizado que provee información estratégica para los altos directivos.

sistemas expertos. Un sistema que modela el proceso de toma de decisión de los expertos en varios dominios de problema, tales como diagnóstico médico, decisiones de auditoría, etc.; un tipo especial de desarrollo de IA.

solicitud de propuesta (SDP). Un documento formal que delinea los requisitos de rendimiento y le pide a los vendedores que respondan con una propuesta que satisfaga aquellos requisitos.

SQL incrustado (*embedded SQL*). Un conjunto de instrucciones que permite que SQL sea utilizado con lenguajes de programación tradicionales.

subconsulta. Una consulta dentro de una consulta.

subconsulta no correlacionada. Una subconsulta cuyos valores no dependen de ninguna consulta más externa.

subesquema. Subconjuntos del esquema que son definidos por las vistas de los usuarios de la base de datos.

superclave. Un conjunto de atributos que identifica unívocamente cada fila en una relación.

sustracción. La operación diferencia del álgebra relacional.

System-2000 (del SAS Institute). SGBD basado en el modelo de datos jerárquico.

System for Distributed Databases (SSD). SGBDD comercializado por Computer Corporation of America.

T

tabla ejemplo. En QBE es una tabla esqueleto que muestra el nombre de la tabla y los nombres de las columnas arriba de espacios en blanco que se usan para la entrada de las condiciones de consulta.

tabla resultado. En QBE una tabla sin encabezamientos por columna. Se usa para definir la salida de una consulta.

tablas de verificación de disparadores. Tablas actualizadas con las tuplas insertadas y borradas cada vez que ocurre una actualización en una tabla correspondiente de la base de datos.

painter. Un subprograma interactivo que lleva a cabo un tipo específico de funcionalidad necesaria para el desarrollo completo de una aplicación.

TDMS (System Development Corporation's Time-Shared Data Management System). SGBD basado en el modelo de datos jerárquico.

teoría de la serialización. Establece que un algoritmo de control de concurrencia es correcto cuando sus resultados son los mismos que si se hubiese procesado serialmente.

tercera forma normal (3FN). Cada determinante es una clave.

texto cifrado. Texto legible cifrado.

texto legible. Texto que se puede leer.

tiempo de activación de la cabeza. El tiempo necesario para activar una cabeza de lectura/escritura.

tipo de dato definido por el usuario. Un subtipo de uno de los tipos de datos suministrados por el sistema, el

cual es adaptado a las necesidades del esquema de la base de datos.

tipo de interrelación padre-hijo (tipo IPH). Interrelación lógica entre un tipo de segmento padre y un tipo de segmento hijo.

tipo de segmento. Corresponde a un objeto en el modelo de dato orientado a objetos, también llamado un segmento.

tipo de segmento dependiente. Todos los tipos de registros menos que el tipo de segmento raíz.

tipo registro. Una colección de elementos de datos relacionados lógicamente.

tipo registro enlace. Un registro que se crea para convertir un retículo complejo en un retículo simple equivalente.

tipo registro miembro. El tipo de registro de la parte " muchos" de una interrelación muchos-muchos de un conjunto DBTG.

tipo registro propietario. El tipo registro de la parte "uno" de una interrelación uno-muchos de un conjunto DBTG.

transacción. Una unidad de programa cuya ejecución conserva la consistencia de la base de datos.

transacción abortada. Transacción que se cancela antes de aplicar los cambios a la base de datos.

transacción atómica. Una transacción en la cual todas las acciones asociadas con la transacción se ejecutan completamente o ninguna se ejecuta.

transacción local. Una transacción que requiere un único agente.

transacción global. Una transacción que requiere varios agentes.

tupla. Una fila en una relación.

U

unidad de referencia. El SGBD que está siendo evaluado en el análisis AED.

unión. Operación del álgebra relacional que crea un conjunto unión de dos relaciones unión-compatible.

unión-compatible. Dos o más relaciones que tienen columnas equivalentes en número y dominios.

uno-uno. Una cardinalidad de la interrelación que es 1 en ambas direcciones.

uno-muchos. Una cardinalidad de la interrelación que es 1 en una dirección y muchos en la otra.

UPDATE. Operación que cambia los valores de las columnas en las filas.

usuarios. Personas que necesitan información de la base de datos para desarrollar su responsabilidad primaria en el negocio.

V

valor atómico. Un valor que no es un conjunto de valores o un grupo repetitivo.

valor nulo. El valor dado a un atributo en una tupla si el atributo es inaplicable o su valor es desconocido.

valor nulo de un atributo. Un valor de atributo que no existe en una instancia específica.

valor por defecto. Un valor que se inserta automáticamente si el usuario no lo especifica en una entrada.

valor por defecto del parámetro. El valor de un parámetro que suministra el sistema si el programa que llama lo omite.

variable. Un nombre simbólico que representa una instancia no especificada en un conjunto de objetos.

variable local. Una variable definida para usar dentro de un procedimiento y almacenar valores temporales de trabajo.

velocidad de transferencia de datos. La velocidad a la cual los datos se leen del disco hacia la memoria principal, o equivalentemente, la razón en la cual los datos se escriben desde la memoria principal al disco.

viabilidad económica. Estudio de costo-beneficio del sistema de base de datos propuesto.

viabilidad operacional. Determinación de la disponibilidad de experticidad y del personal que se necesita para el sistema de base de datos.

viabilidad tecnológica. Determinación de la disponibilidad de hardware y el software para el sistema de base de datos.

vista. Una definición de una porción restringida de la base de datos; también llamada vista de datos.

vista de los datos. Una definición de una porción restringida de la base de datos; también llamada una vista.

W

Window painter. Un *painter* que se usa para construir ventanas de una aplicación.

BIBLIOGRAFÍA

Las siguientes abreviaturas se usaron en esta bibliografía:

ACM	Association for Computing Machinery.
AFIPS	American Federation of Information Processing Societies.
CACM	Communications of the ACM.
DE	Proceedings of the IEEE Computer Society International Conference on Data Engineering.
EDS	Proceedings of the International Conference on Data Engineering.
ER	Proceedings of the International Conference on Entity-Relationship Approach.
ICIS	Proceedings of the International Conference on Information Systems.
IFIP	International Federation for Information Processing.
NCC	Proceedings of the National Computer Conference.
OOPSLA	Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications.
PODS	Proceedings of the ACM Symposium on Principles of Database Systems.
SIGMOD	Proceedings of the ACM SIGMOD International Conference on Management of Data.
TODS	ACM Transactions on Database Systems.
TOOIS	ACM Transaction on Office Information Systems.
TSE	IEEE Transactions on Software Engineering.
VLDB	Proceedings of the International Conference on Very Large Data Bases.

ABRIAL, J. "Data Semantics." In Klimbie and Koffeman, 1974.

AGRAWAL, R., GHOSH, S., IMIELINSKI, T., IYER, B., AND SWAMI, A. "An Interval Classifier for Database Mining Applications." In VLDB 1992.

AHO, A., BEERI, C., AND ULLMAN, J. "The Theory of Joins in Relational Databases." TODS, 4:3, September 1979.

AHO, A.V., HOPCROFT, E., & ULLMAN, J.D. *The Design and Analysis of Computer Programs*, Addison-Wesley, 1975.

ALASHQUR, A.M., SU, S.Y.W., AND LAM, H. "OQL: A Query Language for Manipulating Object-Oriented Databases." In VLDB 1989.

ALBANO, A., DE ANTONELLIS, V., AND DE LEYA, A. (editors). *Computer-Aided Database Design: The DATAID Project*, North-Holland, 1985.

ALBANO, A., BERGAMINI, R., GHELLI, G., AND ORSINI, R. "An Object Data Model with Roles." In VLDB 1993.

ALLEN, F., LOOMIS, M., AND MANNINO, M. "The Integrated Dictionary/Directory System." *Computing Surveys*, 14:2, June 1982.

ANDANY, J., LEONARD, M., AND PALISSER, C. "Management of Schema Evolution in Databases." In VLDB 1991.

ANDREWS, T. AND HARRIS, C. "Combining Language and Database Advances in an Object-Oriented Development Environment." In *OOPSLA 1987*.

ANDRIOLE, S. J., ED. 1985. *Applications in Artificial Intelligence*, Princeton, NJ: Petrocelli Books, 1985.

ANSI 1981. Proposed American National Standard for a Data Definition Language for Network Structured Databases. American National Standards Institute, Document ANSI X3H2, 1981.

ANSI 1986A. American National Standards Institute: The Database Language NDL, Document ANSI X3.133, 1986.

ANSI 1986B. American National Standards Institute: The Database Language SQL, Document ANSI X3.135, 1986.

ANSI 1989. American National Standards Institute: Information Resource Dictionary Systems, Document ANSI X3.138, 1989.

ASHTON-TATE CORPORATION. *dBASE IV Language Reference*, Ashton-Tate, 1988.

ASTRAHAN, M. ET AL. "System R: A Relational Approach to Data Base Management." TODS, 1:2, June 1976.

ASTRAHAN, M. ET AL. "A History and Evaluation of System R." *IBM Research Report RJ2843*, June 1980.

ATRE, S. *Data Base: Structured Techniques for Design, Performance, and Management*, New York: Wiley, 1980.

ATRE, S. *Data Base Management Systems for the Eighties*, QED Information Sciences, 1983.

BABAD, Y. M., AND J. A. HOFFER. "Even No Data Has a Value." CACM 27:8, August, 1984.

BACHMAN, C. "Data Structure Diagrams." *Data Base* (Bulletin of ACM SIGFIDET), 1:2, March 1969.

BACHMAN, C. "The Programmer as a Navigator." CACM, 16:11, November 1973.

BACHMAN, C. "The Data Structure Set Model." In Rustin 1974.

BANERJEE, J., CHOU, H., GARZA, J., KIM, W., WOELK, D., BALLOU, N., AND KIM, H. "Data Model Issues for Object-Oriented Applications." *ACM Transactions on Database Systems* 5, 197-208, 1987.

BAROODY, A. AND DEWITT, D. "An Object-Oriented Approach to Database System Implementation." TODS, 6:4, December 1981.

BATINI, C., LENZERINI, M., AND NAVATHE, S. "A Comparative Analysis of Methodologies for Database Schema Integration." *Computing Surveys*, 18:4, December 1986.

BATORY, D. ET AL. "GENESIS: An Extensible Database Management System." TSE, 14:11, November 1988.

BATRA, D., J. A. HOFFER, AND R. P. BOSTROM. "A Comparison of the Representations Developed Using the Relational and Entity-Relationship Data Models." CACM, 33:12, December 1990.

BAYER, R. AND MCCREIGHT, E. "Organization and Maintenance of Large Ordered Indexes." *Acta Informatica*, 1:3, February 1972.

BEECH, DAVID. "Collections of Objects in SQL3." In VLDB 1993.

BEERI, C. AND MILO, T. "A Model for Active Object Oriented Database." In VLDB 1991.

BELL, D. AND GRIMSON, J. *Distributed Database Systems*. Reading, MA: Addison-Wesley, 1992.

BERNSTEIN, P. AND GOODMAN, N. "The Power of Natural Semijoins." *SIAM Journal of Computing*, 10:4, December 1981.

BERNSTEIN, P. AND GOODMAN, N. "Concurrency Control in Distributed Database Systems." *Computing Surveys*, 13:2, June 1981.

BERNSTEIN, P., HADZILACOS, V., AND GOODMAN, N. *Concurrency Control and Recovery in Database Systems*, Reading, MA: Addison-Wesley, 1988.

BERNSTEIN, PHILIP A., GYLLSTROM, PER O., AND WIMBERG, TOM. "STDL-A Portable Language for Transaction Processing." In VLDB 1993.

BERNSTEIN, P. "Synthesizing Third Normal Form Relations from Functional Dependencies." TODS, 1:4, December 1976.

BETRA, D., J. A. HOFFER, AND R. B. BOSTROM. "A Comparison of User Performance Between the Relational and Extended Entity Relationship Model in the Discovery Phase of Database Design." In ICIS 1988.

BHARGAVA, B. (editor). *Concurrency and Reliability in Distributed Systems*, New York: Van Nostrand-Reinhold, 1987.

BILLER, H. "On the Equivalence of Data Base Schemas—A Semantic Approach to Data Translation." *Information Systems*, 4:1, 1979.

BJORNER, D. AND LOVENGREN, H. "Formalization of Database Systems and a Formal Definition of IMS." In VLDB 1982.

BLASGEN, M. AND ESWARAN, K. "On the Evaluation of Queries in a Relational Database System." *IBM Systems Journal*, 16:1, January 1976.

BLASGEN, M., ET AL. "System R: An Architectural Overview." *IBM Systems Journal*, 20:1, January 1981.

BLOOMBECKER, J. J. "Short-Circuiting Computer Crime." *Datamation*, October 1, 1989.

BOAR, B. H. *Application Prototyping*, New York: Wiley, 1984.

BOCCA, J. "EDUCE—A Marriage of Convenience: Prolog and a Relational DBMS." *Proceedings of the Third International Conference on Logic Programming*, New York: Springer-Verlag, 1986.

BOEHM, B.W. *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall, 1981.

BOHL, M. *Introduction to IBM Direct Access Storage Devices*, Chicago: Science Research Associates, 1981.

BORLAND INTERNATIONAL. *Paradox 3.0 User's Guide*, Scotts Valley, CA: Borland, 1988.

BOUZEGHOUB, M. AND METAIS, E. "Semantic Modeling of Object Oriented Databases." In VLDB 1991.

BRACCHI, G. AND PERNICI, B. "The Design Requirements of Office Systems." TOOIS, 2:2, April 1984.

BRACCHI, G. AND PERNICI, B. "Decision Support in Office Information Systems." In Holsapple and Whinston 1987.

BRACHMAN, R. AND LEVESQUE, H. "What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level." In EDS 1984.

BRADLEY, J. *File and Data Base Techniques*, New York: Holt, Rinehart & Winston, 1982.

BRANT, D.A., GROSE, T., LOFASO, B., AND MIRANKER, D.P. "Effects of Database Size on Rule System Performance: Five Case Studies." In VLDB 1991.

BRAY, O. *Distributed Database Management Systems*, Lexington, MA: Lexington Books, 1982.

BRODIE, M., MYLOPOULOS, J., AND SCHMIDT, J. (editors). *On Conceptual Modeling*, New York: Springer-Verlag, 1984.

BRODSKY, ALEXANDER, JAFFAR, JOXAN, AND MAHER, MICHAEL J. "Toward Practical Constraint Databases." In VLDB 1993.

BROSEY, M. AND SHNEIDERMAN, B. "Two Experimental Comparisons of Relational and Hierarchical Database Models." *International Journal of Man-Machine Studies*, 1978.

BROWN, R. "Data Integrity and SQL." *Database Programming and Design*, March 1988.

BROWNING, D. "Data Managers and LANs." *PC Tech Journal*, 5:5, May 1987.

BRUCE, T., J. FULLER, AND T. MORIARTY. "So You Want a Repository." *Database Programming and Design*, May 1989.

BUBENKO, J., BERILD, S., LINDERCRONA-OHLIN, E., AND NACHMENS, S. "From Information Requirements to DBTG Data Structures." *Proceedings of the ACM SIGMOD/SIGPLAN Conference on Data Abstraction*, 1976.

BUNEMAN, P. AND FRANKEL, R. "FQL: A Functional Query Language." In SIGMOD 1979.

CAMPBELL, D., EMBLEY, D., AND CZEJDO, B. "A Relationally Complete Query Language for the Entity-Relationship Model." In ER 1985.

CAMPBELL, D., EMBLEY, D., AND CZEJDO, B. "Graphical Query Formulation for an Entity-Relationship Model." *Data and Knowledge Engineering*, 2 (1987), 89-121.

CARDENAS, A. *Data Base Management Systems*, Second Edition, Newton, MA: Allyn and Bacon, 1985.

CAREY, M., DEWITT, D., AND VANDENBERG, S. "A Data Model and Query Language for Exodus." In SIGMOD 1988.

CAREY, M., DEWITT, D., RICHARDSON, J. AND SHEKITA, E. "Object and File Management in the EXODUS Extensible Database System." In VLDB 1986.

CAREY, M. ET AL. "The Architecture of the EXODUS Extensible DBMS." In Dittrich and Dayal 1986.

CARLIS, J. AND MARCH, S. "A Descriptive Model of Physical Database Design Problems and Solutions." In DE 1984.

CASANOVA, M. AND VIDAL, V. "Toward a Sound View Integration Method." PODS, 1982.

CERI, S. (editor). *Methodology and Tools for Database Design*, North-Holland, 1983.

CERI, S. AND PELAGATTI, G. *Distributed Databases: Principles and Systems*, New York: McGraw-Hill, 1984.

CERI, S., NAVATHE, S., AND WIEDERHOLD, G. "Distribution Design of Logical Database Schemas." TSE, 9:4, July 1983.

CHA, S.K. AND WIEDERHOLD, G. "Kaleidoscope Data Model for an English-like Query Language." In VLDB 1991.

CHAMBERLIN, D. AND BOYCE, R. "SEQUEL: A Structured English Query Language." In SIGMOD 1984.

CHAMBERLIN, D., ET AL. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control." *IBM Journal of Research and Development*, 20:6, November 1976.

CHAMBERLIN, D., ET AL. "A History and Evaluation of System R." CACM, 24:10, October 1981.

CHAMPINE, G. A. "Six Approaches to Distributed Data Bases." *Datamation*, May 1977.

CHAN, A., AND H. K. T. WONG. "Serving Up dBASE." *Data Base Programming & Design*, February 1990.

CHANG, C. AND WALKER, A. "PROSQL: A Prolog Programming Interface with SQL/DS." In EDS 1984.

CHEN AND ASSOCIATES. *E-R Designer Reference Manual*, 1988.

CHEN, I.A., AND MCLEOD, D. "Derived Data Update in Semantic Databases." In VLDB 1989.

CHEN, LING TONY AND ROTEM, DORON. "Declustering Objects for Visualization." In VLDB 1993.

CHEN, P. "The Entity Relationship Model—Toward a Unified View of Data." TODS, 1:1, March 1976.

CHEN, P. *The Entity-Relationship Approach to Logical Data Base Design*, Q.E.D. Information Sciences, Data Base Monograph Series no. 6, 1977.

CHIMENTI, D., GAMBOA, R., KRISHNAMURTHY, R., NAQVI, S., TSUR, S., AND ZANIOLO, C. "The LDL System Prototype." *IEEE Transactions on Knowledge and Data Engineering*, 2:1, March 1990.

CHOUINARD, P. "Supertypes, Subtypes, and DB2." *Database Programming and Design*, October 1989.

CHRISTODOULAKIS, S. ET AL. "Development of a Multimedia Information System for an Office Environment." In VLDB 1984.

CLAYBROOK, B. *File Management Techniques*, New York: Wiley, 1983.

CODASYL. Data Base Task Group April 71 Report, ACM, 1971.

CODASYL. Data Description Language Journal of Development, Canadian Government Publishing Centre, 1978.

CODD, E. "A Relational Model for Large Shared Data Banks." CACM, 13:6, June 1970.

CODD, E. "Relational completeness of data base sublanguages." *Courant Computer Science Symposium 6, Data Base Systems*, Englewood Cliffs, NJ: Prentice Hall, 1971.

CODD, E. "A Data Base Sublanguage Founded on the Relational Calculus." *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control*, November 1971.

CODD, E. "Further Normalization of the Data Base Relational Model." In Rustin 1972.

CODD, E. "Recent Investigations in Relational Database Systems." *Proceedings of the IFIP Congress*, 1974.

CODD, E. "How About Recently? (English Dialog with Relational Data Bases Using Rendezvous Version 1)." In Shneiderman 1978.

CODD, E. "Extending the Database Relational Model to Capture More Meaning." TODS, 4:4, December 1979.

CODD, E. "Relational Database: A Practical Foundation for Productivity." CACM, 25:2, December 1982.

CODD, E. "Is Your DBMS Really Relational?" and "Does Your DBMS Run By the Rules." *Computerworld*, October 14 and October 21, 1985.

CODD, E. "An Evaluation Scheme for Database Management Systems That Are Claimed to be Relational." In DE 1986.

COMER, D. "The Ubiquitous B-tree." *Computing Surveys*, 11:2, June 1979.

CONTE, P. "In Search of Consistency." *Database Programming and Design*, August 1989.

CREASY, P.N. "ENIAM: A More Complete Conceptual Schema Language." In VLDB 1989.

CURTICE, R., AND CASEY, W. "Database: What's in Store." *Datamation*, December 1, 1985.

CURTICE, R. "Data Dictionaries: An Assessment of Current Practice and Problems." In VLDB 1981.

CZEJDO, B., ELMASRI, R., RUSINKIEWICZ, M., AND EMBLEY, D. "An Algebraic Language for Graphical Query Formulation Using an Extended Entity-Relationship Model." *Proceedings of the ACM Computer Science Conference*, 1987.

DATE, C. AND WHITE, C. *A Guide to SQL/DS*, Reading, MA: Addison-Wesley, 1988.

DATE, C. AND WHITE, C. *A Guide to DB2*, Second Edition, Reading, MA: Addison-Wesley, 1988.

DATE, C. *An Introduction to Database Systems, Volume 2*, Reading, MA: Addison-Wesley, 1983.

DATE, C. "The Outer Join." *Proceedings of the Second International Conference on Databases*, 1983.

DATE, C. "A Critique of the SQL Database Language." *ACM SIGMOD Record*, 14:3, November 1984.

DATE, C. *An Introduction to Database Systems, Vol. 1* (4th ed.), Reading, MA: Addison-Wesley, 1986.

DATE, C. "Where SQL Falls Short." *Datamation*, May 1, 1987.

DATE, C. AND DARWEN, H. *A Guide to the SQL Standard, 3rd Edition*, Reading, MA: Addison-Wesley, 1994.

DAVIES, C. "Recovery Semantics for a DB/DC System." *Proceedings of the ACM National Conference*, 1973.

DAYAL, U., HSU, M., AND LADIN, R. "A Transactional Model for Long-Running Activities." In VLDB 1991.

DBTG. Report of the CODASYL Data Base Task Group, ACM, April 1971.

DELIS, A. AND ROUSSOPOULOS, N. "Performance and Scalability of Client-Server Database Architectures." In VLDB 1992.

DEMARCO, T. *Structured Analysis and System Specification*, Prentice-Hall Yourdan, Inc., 1979.

DENNING, D. AND DENNING, P. "Data Security." *Computing Surveys*, 11:3, September 1979.

DIAZ, O., PATON, N. AND GRAY, P. "Rule Management in Object-Oriented Databases: A Uniform Approach." In VLDB 1991.

DI BATTISTA, G. AND LENZERINI, M. "A Deductive Method for Entity-Relationship Modeling." In VLDB 1989.

DIFFIE, W. AND HELLMAN, M. "Privacy and Authentication." *Proceedings of the IEEE*, 67:3, March 1979.

DITTRICH, K. AND DAYAL, U. (editors). *Proceedings of the International Workshop on Object-Oriented Database Systems*, IEEE CS, September 1986.

DITTRICH, K. "Object-Oriented Database Systems: The Notion and the Issues." In Dittrich and Dayal 1986.

DODD, G. "Elements of Data Management Systems." *Computing Surveys*, 1:2, June 1969.

DOS SANTOS, C., NEUHOLD, E., AND FURTADO, A. "A Data Type Approach to the Entity-Relationship Model." In ER 1979.

DUMPALA, S. AND ARORA, S. "Schema Translation Using the Entity-Relationship Approach." In ER 1983.

EICK, C.F. "A Methodology for the Design and Transformation of Conceptual Schemas." In VLDB 1991.

ELLIS, C. AND NUTT, G. "Office Information Systems and Computer Science." *Computing Surveys*, 12:1, March 1980.

ELLZEY, R.S. *Data Structures for Computer Information Systems*, Science Research Associates, 1982.

ELMASRI, R. AND LARSON, J. "A Graphical Query Facility for ER Databases." In ER 1985.

ELMASRI, R. AND NAVATHE, S. "Object Integration in Logical Database Design." In DE 1984.

ELMASRI, R. AND NAVATHE, S. *Fundamentals of Database Systems*, Menlo Park, CA: Benjamin/Cummings, 1989.

ELMASRI, R. AND WIEDERHOLD, G. "Data Model Integration Using the Structural Model." In SIGMOD 1979.

ELMASRI, R. AND WIEDERHOLD, G. "Structural Properties of Relationships and Their Representation." NCC, AFIPS, 49, 1980.

ELMASRI, R. AND WIEDERHOLD, G. "GORDAS: A Formal, High-Level Query Language for the Entity-Relationship Model." In ER 1981.

ELMASRI, R., WEELDREYER, J., AND HEVNER, A. "The Category Concept: An Extension to the Entity-Relationship Model." *International Journal on Data and Knowledge Engineering*, 1:1, May 1985.

FAGIN, R. "Multivalued Dependencies and a New Normal Form for Relational Databases." TODS, 2:3, September 1977.

FAGIN, R. "Normal Forms and Relational Database Operators." In SIGMOD 1979.

FAGIN, R. "A Normal Form for Relational Databases That is Based on Domains and Keys." TODS, 6:3, September 1981.

FERNANDEZ, E., SUMMERS, R., AND WOOD, C. *Database Security and Integrity*, Reading, MA: Addison-Wesley, 1981.

FISHMAN, D. ET AL. "IRIS: An Object-Oriented DBMS." TOOIS, 4:2, April 1986.

FISHMAN, D., BEECH, D., CATE, H., CHOW, E., CONNORS, T., DAVIS, J., DERRETT, N., HOCH, C., KENT, W., LYNGBAEK, P., MAHBOD, B., NEIMAT, M., RYAN, T., AND SHAN, M. IRIS: An Object-Oriented Database Management System. TOOIS 5: 216-226, 1987.

FLAVIN, M. *Fundamental Concepts of Information Modeling*, Englewood Cliffs, NJ: Yourdon Press, 1981.

FLEMING, C., AND VON HALLE, B. "An Overview of Logical Data Modeling." *Data Resource Management*, Winter 1990.

FRANKLIN, M.J., CAREY, M.J., AND LIVNY, M. "Global Memory Management in Client-Server DBMS Architectures." In VLDB 1992.

FRANKLIN, MICHAEL J., CAREY, MICHAEL J., AND LIVNY, MIRON. "Local Disk Caching for Client-Server Database Systems." In VLDB 1993.

FRY, J. AND SIBLEY, E. "Evolution of Data-Base Management Systems." *Computing Surveys*, 8:1, March 1976.

FURTADO, A. "Formal Aspects of the Relational Model." *Information Systems*, 3:2, 1978.

GABBAY, D. AND MCBRIEN, P. "Temporal Logic & Historical Databases." In VLDB 1991.

GADIA, S. "A Homogeneous Relational Model and Query Language for Temporal Databases." TODS, 13:4, December 1988.

GALLAIRE, H. AND MINKER, J. (editors). *Logic and Databases*, Plenum Press, 1978.

GALLAIRE, H., MINKER, J., AND NICOLAS, J. "Logic and Databases: A Deductive Approach." *Computing Surveys*, 16:2, June 1984.

GAL-OZ, NURITH, GUDES, EHUD, AND FERNANDEZ, EDUARDO B. "A Model of Methods Access Authorization in Object-Oriented Databases." In VLDB 1993.

GARDARIN, G., CHEINEY, J-P., KIERNAN, G., PASTRE, D., AND STORA, H. "Managing Complex Objects in an Extensible Relational DBMS." In VLDB 1989.

GARDARIN, G., AND VALDURIEZ, P. *Relational Databases and Knowledge Bases*, Reading, MA: Addison-Wesley, 1989.

GAREY, M. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

GEHANI, N.H., JAGADISH, H.V., AND SHMUELI, O. "Composite Event Specification in Active Databases: Model & Implementation." In VLDB 1992.

GEHANI, N.H. AND JAGADISH, H.V. "Ode as an Active Database: Constraints and Triggers." In VLDB 1991.

GHANDEHARIZADEH, SHAHRAM, HULL, RICHARD, JACOBS, DEAN, CASTILLO, JAIME, ESCOBAR-MOLANO, MARTHA, LU, SHIH-HUI, LUO, JUNHUI, TSANG, CHIU, AND ZHOU, GANG. "On Implementing a Language for Specifying Active Database Execution Models." In VLDB 1993.

GOLDFINE, A. AND KONIG, P. A Technical Overview of the Information Resource Dictionary System (IRDS), Second Edition, NBS IR 88-3700, National Bureau of Standards, 1988.

GORMAN, K., AND CHOOBINEH, J. "An Overview of the Object-Oriented Entity Relationship Model (OOERM)." *Proceedings of the Twenty-Third Annual Hawaii International Conference on Information Systems*, 1990.

GOTLIEB, L. "Computing Joins of Relations." In SIGMOD 1975.

GRAY, J., MCJONES, P., AND BLASGEN, M. "The Recovery Manager of the System R Database Manager." *Computing Surveys*, 13:2, June 1981.

GRAY, J. "The Transaction Concept: Virtues and Limitations." In VLDB 1981.

GREENBLATT, D., AND J. WAXMAN. "A Study of Three Database Query Languages." In Shneiderman 1978.

GREFEN, PAUL W.P.J. "Combining Theory and Practice in Integrity Control: A Declarative Approach to the Specification of a Transaction Modification Subsystem." In VLDB 1993.

GUIMARAES, T. "Information Resources Management: Improving the Focus." *Information Resources Management Journal*, Fall 1988.

GUPTA, A., WEYMOUTH, T.E., AND JAIN, R. "Semantic Queries with Pictures: the VIMSYS Model." In VLDB 1991.

HAMMER, M. AND MCLEOD, D. "Semantic Integrity in a Relational Data Base System." In VLDB 1975.

HAMMER, M., AND D. MCLEOD. "The Semantic Data Model: A Modelling Mechanism for Data Base Applications." In SIGMOD 1978.

HAMMER, M. AND MCLEOD, D. "Database Descriptions with SDM: A Semantic Data Model." TODS, 6:3, September 1980.

HAN, J., CAI, Y., AND CERCONE, N. "Knowledge Discovery in Databases: An Attribute-

Oriented Approach." In VLDB 1992.

HANSEN, G. *Database Processing with Fourth Generation Languages*, Cincinnati: South-Western, 1988.

HANSEN, G. AND HANSEN, J. "Procedural and Non-procedural Languages Revisited: A Comparison of Relational Algebra and Relational Calculus." *International Journal of Man-Machine Studies*, (1987), 26, 683-694.

HANSEN, G. AND HANSEN, J. "Human Performance in Relational Algebra, Tuple Calculus, and Domain Calculus." *International Journal of Man-Machine Studies*, (1988), 29, 503-516.

HARRINGTON, J. *Relational Database Management for Microcomputer: Design and Implementation*, New York: Holt, Rinehart, and Winston, 1987.

HARRIS, L. "The ROBOT System: Natural Language Processing Applied to Data Base Query." *Proceedings of the ACM National Conference*, December 1978.

HASKIN, R. AND LORIE, R. "On Extending the Functions of a Relational Database System." In SIGMOD 1982.

HAWRYSZKIEWYCA, I.T. *Database Analysis and Design, 2nd Ed.*, New York: Macmillan, 1991.

HAYES-ROTH, R., WATERMAN, D., AND LENAT, D. (editors). *Building Expert Systems*, Reading, MA: Addison-Wesley, 1983.

HEIMBIGNER, D. "Experiences With an Object Manager for a Process-Centered Environment." In VLDB 1992.

HELD, G. AND STONEBRAKER, M. "B-Trees Reexamined." CACM, 21:2, February 1978.

HIMMELSTEIN, M. "Cooperative Database Processing." *Database Programming and Design*, October 1989.

HOFFER, J., MICHAELE, S., AND CARROLL, J. "The Pitfalls of Strategic Data and Systems Planning: A Research Agenda." *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Vol. IV*, 1989.

HOFFER, J. "An Empirical Investigation with Individual Differences in Database Models." In ICIS 1982.

HOLLAND, R. H. "Data Base Planning Entails Return to Basics." *Computerworld*, October 27, 1980.

HOLSAPPLE, C. AND WHINSTON, A. (editors). *Decision Support Theory and Application*, New York: Springer-Verlag, 1987.

HUBBARD, G.U. *Computer-Assisted Data Base Design*, New York: Van Nostrand Reinhold, 1981.

HULL, R. AND KING, R. "Semantic Database Modeling: Survey, Applications, and Research Issues." *Computing Surveys*, 19:3, September 1987.

HULL, R. AND JACOBS, D. "Language Constructs for Programming Active Databases." In VLDB 1991.

IMIELINSKI, T. AND LIPSKI, W. "On Representing Incomplete Information in a Relational Database." In VLDB 1981.

ISHIKAWA, HIROSHI AND KUBOTA, KAZUMI. "An Active Object-Oriented Database: A Multi-Paradigm Approach to Constraint Management." In VLDB 1993.

JACKSON, M.A. *Principles of Program Design*, Orlando: Academic Press, 1975.

JAQUA, D. "SQL Database Security." *Database Programming and Design*, July 1988.

JARDINE, D. (editor). *The ANSI/SPARC DBMS Model*, North-Holland, 1977.

KAPP, D. AND LEBEN, J. *IMS Programming Techniques*, New York: Van Nostrand-Reinhold, 1978.

KENT, W. *Data and Reality*, North-Holland, 1978.

KENT, W. "Limitations of Record-Based Information Models." TODS, 4:1, March 1979.

KENT, W. "A Simple Guide to Five Normal Forms in Relational Database Theory." CACM, 26:2, February 1983.

KENT, W. "Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language." In VLDB 1991.

KHOSHAFIAN, S., CHAN, A., WONG, A., WONG, H.K.T. *Client/Server SQL Applications*, Morgan Kaufmann, 1992.

KIM, W. "Relational Database Systems." *Computing Surveys*, 11:3, September 1979.

KIM, W. "On Optimizing an SQL-like Nested Query." TODS, 3:3, September 1982.

KIM, W., REINER, D., AND BATORY, D. (editors). *Query Processing in Database Systems*, New York: Springer-Verlag, 1985.

KIM, W. "A Model of Queries for Object-Oriented Databases." In VLDB 1989.

KLIMBIE, J. AND KOFFEMAN, K. (editors). *Data Base Management*, North-Holland, 1974.

KNUTH, D. *The Art of Computer Programming, Volume 3: Sorting and Searching*, Reading, MA: Addison-Wesley, 1973.

KORTH, H. AND SILBERSCHATZ, A. *Database System Concepts*, New York: McGraw-Hill, 1986.

KORTH, H. AND SILBERSCHATZ, A. *Database System Concepts, 2nd Edition*, New York: McGraw-Hill, 1991.

KROENKE, D. AND DOLAN, K. *Database Processing, Third Edition*, Chicago: Science Research Associates, 1988.

KROENKE, D. "Developing Object-Oriented Database Applications on Microcomputers." *Proceedings of the Second International Conference on Computers and Applications*, Beijing, June 1987.

KULL, D. "Anatomy of a 4GL Disaster." *Computer Decisions*, February 11, 1986.

KUNTZ, M. AND MELCHERT, R. "Pasta-3's Graphical Query Language: Direct Manipulation, Cooperative Queries, Full Expressive Power." In VLDB 1989.

LAMPSON, BUTLER AND LOMET, DAVID. "A New Presumed Commit Optimization for Two Phase Commit." In VLDB 1993.

LARSON, P. "Dynamic Hashing." *BIT*, 18, 1978.

LARSON, P. "Analysis of Index-Sequential Files with Overflow Chaining." TODS, 6:4, December 1981.

LÉCLUSE, C., RICHAR, P., AND VELEZ, F. "O_2, An Object-Oriented Data Model." *ACM International Conference on the Management of Data*. Chicago, IL, 1988.

LÉCLUSE, C. AND RICHARD, P. "The O_2 Database Programming Language." In VLDB 1989.

LEDERER, A., AND SETHIK, V. "Pitfalls in Planning." *Datamation*, June 1, 1989.

LEE, S.K. "An Extended Relational Database Model for Uncertain and Imprecise Information." In VLDB 1992.

LEFKOVITZ, H. C. *Proposed American National Standards Information Resource Dictionary System*, QED Information Sciences, 1985.

LEISS, E. *Principles of Data Security*, New York: Plenum Press, 1982.

LENZERINI, M. AND SANTUCCI, C. "Cardinality Constraints in the Entity Relationship Model." In ER 1983.

LIEN, E. AND WEINGERGER, P. "Consistency, Concurrency, and Crash Recovery." In SIGMOD 1978.

LITWIN, P. "Faking Multi-Table Forms." *Data Based Advisor*, October 1989.

LITWIN, W. "Virtual Hashing: A Dynamically Changing Hashing." In VLDB 1978.

LITWIN, W. "Linear Hashing: A New Tool for File and Table Addressing." In VLDB 1980.

LIU, K. AND SUNDERRAMAN, R. "On Representing Indefinite and Maybe Information in Relational Databases." In DE 1988.

LIVADAS, P. *File Structures: Theory and Practice*, Englewood Cliffs, NJ: Prentice Hall, 1989.

LOCKEMANN, P. AND KNUTSEN, W. "Recovery of Disk Contents after System Failure." CACM, 11:8, August 1968.

LOMET, DAVID B. "Key Range Locking Strategies for Improved Concurrency." In VLDB 1993.

LOZINSKII, E. "A Problem-Oriented Inferential Database System." TODS, 11:3, September 1986.

LYON, L. "CASE and the Database." *Database Programming and Design*, May 1989.

MAIER, D., STEIN, J., OTIS, A., AND PURDY, A. "Development of an Object-Oriented DBMS." OOPSLA, 1986.

MAIER, D. AND STEIN, J. "Development and Implementation of an Object-Oriented

DBMS". In B. Shriver and P. Wegner, eds. *Research Directions in Object-Oriented Programming*, 355-392. Cambridge, MA: MIT Press, 1987.

MAIER, D. *The Theory of Relational Databases*, Rockville, MD: Computer Science Press, 1983.

MANOLA, F. AND DAYAL, U. "PDM: An Object-Orieinted Data Model." *International Workshop on Object-Orieinted Database Systems*. Pacific Grove, CA, 1986.

MARKOWITZ, V. AND RAZ, Y. "ERROL: An Entity-Relationship, Role Oriented, Query Language." In ER 1983.

MARTIN, E., DEHAYES, D., HOFFER, J., AND PERKINS, W. *Managing Information Technology: What Managers Need to Know*, New York: Macmillan, 1991.

MARTIN, J. *Computer Data Base Organization. 2nd ed.*, Englewood Cliffs, NJ: Prentice Hall, 1977.

MARTIN, J. *An End-User's Guide to Data Base*, Englewood Cliffs, NJ: Prentice Hall, 1981.

MARTIN, J. *Strategic Data Planning Methodologies*, Englewood Cliffs, NJ: Prentice Hall, 1982.

MARTIN, J. *Managing the Data-Base Environment*, Englewood Cliffs, NJ: Prentice Hall, 1983.

MCFADDEN, F. AND HOFFER, J. *Database Management, Third Edition*, Menlo Park, CA: Benjamin/Cummings, 1991.

MCGEE, W. "The Information Management System IMS/VS, Part I: General Structure and Operation." *IBM Systems Journal*, 16:2, June 1977.

MELTON, J. AND SIMON, A. *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann, 1993.

MEYER, B. *Object-oriented Software Construction*, Englewood Cliffs, NJ: Prentice Hall, 1988.

MICRORIM, INC. *R:BASE for DOS User's Manual*. Redmond, WA: Microrim, 1987.

MILLER, R.J., IOANNIDIS, Y.E., AND RAMAKRISHNAN, R. "The Use of Information Capacity in Schema Integration and Translation." In VLDB 1993.

MISSIKOFF, M. AND WIEDERHOLD, G. "Toward a Unified Approach for Expert and Database Systems." In EDS 1984.

MITSCHANG, B. "Extending the Relational Algebra to Capture Complex Objects." In VLDB 1989.

MOTRO, A. "Using Integrity Constraints to Provide Intensional Answers to Relational Queries." In VLDB 1989.

NAFFAH, N. (editor). *Office Information Systems*, North-Holland, 1982.

NAVATHE, S. AND GADGIL, S. "A Methodology for View Integration in Logical Database Design." In VLDB 1982.

NAVATHE, S. AND KERSCHBERG, L. "Role of Data Dictionaries in Database Design." *Information and Management*, 10:1, January 1986.

NAVATHE, S. AND PILLALAMARRI, M. "Toward Making the ER Approach Object-Oriented." In ER 1988.

NAVATHE, S. AND SCHKOLNICK, M. "View Representation in Logical Database Design." In SIGMOD 1978.

NAVATHE, S., ELMASRI, R., AND LARSON, J. "Integrating User Views in Database Design." *IEEE Computer*, 19:1, January 1986.

NAVATHE, S. "An Intuitive View to Normalize Network-Structured Data." In VLDB 1980.

NG, P. "Further Analysis of the Entity-Relationship Approach to Database Design." TSE, 7:1, January 1981.

NIEVERGELT, J. "Binary Search Trees and File Organization." *Computing Surveys*, 6:3, September 1974.

NIJSSEN, G. (editor). *Modelling in Data Base Management Systems*, North-Holland, 1976.

OHSUGA, S. "Knowledge Based Systems as a New Interactive Computer System of the Next Generation." In *Computer Science and Technologies*, North-Holland, 1982.

OLIVE, A. "On the Design and Implementation of Information Systems from Deductive Conceptual Models." In VLDB, 1989.

OLLE, T. W. *The CODASYL Approach to Data Base Management*, Chichester, England: Wiley, 1980.

OZSOYOGLU, G., OZSOYOGLU, Z., AND MATROS, V. "Extending Relational Algebra and Relational Calculus with Set Valued Attributes and Aggregate Functions." TODS, 12:4, December 1985.

OZSOYOGLU, Z. AND YUAN, L. "A New Normal Form for Nested Relations." TODS, 12:1, March 1987.

PAPADIMITRIOU, C. *The Theory of Database Concurrency Control*, Rockville, MD: Computer Science Press, 1986.

PARENT, C. AND SPACCAPIETRA, S. "An Algebra for a General Entity-Relationship Model." TSE, 11:7, July 1985.

PARKER, D.S., SIMON, E., AND VALDURIEZ, P. "SVP: A Model Capturing Sets, Streams, and Parallelism." In VLDB 1992.

PERCY, T. "My Data, Right or Wrong." *Datamation*, June 1, 1986.

PONCELET, P., TESSEIRE, M., CICCHETTI, R., AND LAKHAL, L. "Towards a Formal Approach for Object Database Design." In VLDB 1993.

POULOVASSILIS, ALEXANDRA AND SMALL, CAROL. "A Domain-theoretic Approach to Integrating Functional and Logic Database Languages." In VLDB 1993.

RAMAKRISHNAN, R., SRIVASTAVA, D., AND SUDARSHAN, S. "CORAL: Control, Relations and Logic." In VLDB 1992.

RAZ, Y. "The Principle of Commitment Ordering, or Guaranteeing Serializability in a Heterogeneous Environment of Multiple Autonomous Resource Managers Using Atomic Commitment." In VLDB 1992.

READ, R.L., FUSSELL, D.S., AND SILBERSCHATZ, A. "A Multi-Resolution Relational Data Model." In VLDB 1992.

REISNER, P. "Use of Psychological Experimentation as an Aid to Development of a Query Language." TSE, 3:3, May 1977.

REISNER, P. "Human Factors Studies of Database Query Language: A Survey and Assessment." *Computing Surveys*, 13:1, March 1981.

RETTIG, M. "Gourmet Guide to the DB2 Catalog." *Data Base Programming and Design*, February 1989.

RICHARDSON, J. "Supporting Lists in a Data Model (A Timely Approach)." In VLDB 1992.

RISCH, T. "Monitoring Database Objects." In VLDB 1989.

ROSENTHAL, A., CHAKRAVARTHY, U.S., BLAUSTEIN, B., AND BLAKELY, J. "Situation Monitoring for Active Databases." In VLDB 1989.

ROTH, M. AND KORTH, H. "The Design of Non-1NF Relational Databases into Nested Normal Form." In SIGMOD 1987.

ROTHNIE, J. ET AL. "Introduction to a System for Distributed Databases (SDD-1)." TODS, 5:1, March 1980.

RUBEL, M. C. "Keeping The Garbage Out." *Data Based Advisor*, April 1989.

RUBEL, M. C. "Entering Data into Screen Forms." *Data Based Advisor*, May 1989.

RUBEL, M. C. "Creating a Report." *Data Based Advisor*, July 1989.

RUDENSTEINER, E.A. "Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases." In VLDB 1992.

RUSTIN, R. (editor). *Data Base Systems*, Englewood Cliffs, NJ: Prentice Hall, 1972.

RUSTIN, R. (editor). *Proceedings of the ACM SIGMOD Debate on Data Models: Data Structure Set Versus Relational*, 1974.

SAYLES, J. S. "All in a row." *Data Based Advisor*, December 1989.

SCHAEFFER, H. *Data Center Operations*, Englewood Cliffs, NJ: Prentice Hall, 1981.

SCHEUERMANN, P. (editor). *Improving Database Usability and Responsiveness*, Orlando: Academic Press, 1982.

SCHKOLNICK, M. "A Survey of Physical Database Design Methodology and Techniques." In VLDB 1978.

SCHMIDT, J. AND SWENSON, J. "On the Semantics of the Relational Model." In SIGMOD 1975.

SCHREIER, U., PIRAHESH, H., AGRAWAL, R., AND MOHAN, C. "Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS." In VLDB 1991.

SCHUR, S. G. "Building an Active Distributed Database." *Database Programming and Design*, April 1989.

SEGEV, A. AND ZHAO, J.L. "Data Management for Large Rule Systems." In VLDB 1991.

SHANK, M., BOYNTON, A. AND ZMUD, R. "Critical Success Factor Analysis as a Methodology for IS Planning." *MIS Quarterly*, 9:2, June 1985.

SHETH, A., LARSON, J., CORNELIO, A., AND NAVATHE, S. "A Tool for Integrating Conceptual Schemas and User Views." In DE 1988.

SHIPMAN, D. "The Functional Data Model and the Data Language DAPLEX." TODS, 6:1, March 1981.

SHNEIDERMAN, B. (editor). *Databases: Improving Usability and Responsiveness*, Orlando: Academic Press, 1978.

SHOENS, K., LUNIEWSKI, A., SCHWARZ, P., STAMOS, J., AND THOMAS, J. "The Rufus System: Information Organization for Semi-Structured Data." In VLDB 1993.

SIBLEY, E. "The Development of Database Technology." *Computing Surveys*, 8:1, March 1976.

SIEGEL, M. AND MADNICK, S.E. "A Metadata Approach to Resolving Semantic Conflicts." In VLDB 1991.

SIEGELMANN, H.T. AND BADRINATH, B.R. "Integrating Implicit Answers with Object-Oriented Queries." In VLDB 1991.

SIMON, E., KIERNAN, J., AND DE MAINDREVILLE, C. "Implementing High-Level Active Rules on Top of Relational Databases." In VLDB 1992.

SMITH, J. AND SMITH, D. "Database Abstractions: Aggregation and Generalization." TODS, 2:2, June 1977.

SMITH, P. AND BARNES, G. *Files & Databases: An Introduction*, Reading, MA: Addison-Wesley, 1987.

SMITH, K. AND WINSLETT, M. "Entity Modeling in the MLS Relational Model." In VLDB 1992.

SNODGRASS, R. AND AHN, I. "A Taxonomy of Time in Databases." In SIGMOD 1985.

SPRAGUE, R., AND MCNURLINK, B. *Information Systems in Practice*, Englewood Cliffs, NJ: Prentice Hall, 1986.

SPRAGUE, R. AND WATSON, H. *Decision Support Systems, 2nd Edition*, Englewood Cliffs, NJ: Prentice Hall, 1989.

SRIVASTAVA, DIVESH, RAMAKRISHNAN, RAGHU, SESHADRI, PRAVEEN, AND SUDARSHAN, S. "Coral++: Adding Object-Orientation to a Logic Database Language." In VLDB 1993.

STONEBRAKER, M., WONG, E., KREPS, P. AND HELD, G. "The Design and Implementation of INGRES." TODS, 1:3, September 1976.

STOREY, V. C., AND R. C. GOLDSTEIN. "A Methodology for Creating User Views in Database Design." TODS, 13:3, September 1988.

SU, S.Y.W. AND CHEN, H-H.M. "A Temporal Knowledge Representation Model OSAM*/T and its Query Language OQL/T." In VLDB 1991.

TALENS, G., OUSSALAH, C., AND COLINAS, M.F. "Versions of Simple and Composite Objects." In VLDB 1993.

TAYLOR, R. AND FRANK, R. "CODASYL Data Base Management Systems." *Computing Surveys*, 8:1, March 1976.

TEOREY, T. AND FRY, J. *Design of Database Structures*, Englewood Cliffs, NJ: Prentice Hall, 1982.

TEORY, T., YANG, D., AND FRY, J. "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model." *Computing Surveys*, 18:2, June 1986.

THOMAS, J. AND GOULD, J. "A Psychological Study of Query By Example." NCC, AFIPS, 44, 1975.

THOMAS, J. AND DEBLOCH S. "A Plan-Operator Concept for Client-Based Knowledge Processing." In VLDB 1993.

TODD, S. "The Peterlee Relational Test Vehicle—A System Overview." *IBM Systems Journal*, 15:4, December 1976.

TSICHRITZIS, D. AND KLUG, A. (editors). *The ANSI/X3/SPARC DBMS Framework*, AFIPS Press, 1978.

TSICHRITZIS, D., AND LOCHOVSKY, F. "Hierarchical Data-base Management: A Survey." *Computing Surveys*, 8:1, March 1976.

TSICHRITZIS, D., AND LOCHOVSKY, F. *Data Base Management Systems*. New York: Academic Press, 1977.

TSICHRITZIS, D. AND LOCHOVSKY, F. *Data Models*, Englewood Cliffs: Prentice Hall, 1982.

TSICHRITZIS, D. "Forms Mangement." CACM, 25:7, July 1982.

UHROWCZIK, P. "Data Dictionary/Directories." *IBM Systems Journal*, 12:4, December 1973.

ULLMAN, J. *Principles of Database Systems, Second Edition*, Rockville, MD: Computer Science Press, 1982.

ULLMAN, J. *Principles of Database and Knowledge-Base Systems*, Rockville, MD: Computer Science Press, 1990.

UMBAUGH, R., (editor). *The Handbook of MIS Management*, Auerbach, 1985.

VALDURIEZ, P. AND GARDARIN, G. *Analysis and Comparison of Relational Database Systems*, Reading, MA: Addison-Wesley, 1989.

VETTER, M., AND MADDISON, R.N. *Database Design Methodology*, Englewood Cliffs, NJ: Prentice Hall, 1981.

VETTER, M. *Strategy for Data Modeling*, Wiley, 1987.

WELDON, J. *Data Base Administration*, New York: Plenum Press, 1981.

WELTY, C., AND STEMPLE, D. "Human Factors Comparison of a Procedural and a Nonprocedural Query Language." TODS, 6:4, December 1981.

WERTZ, C. *The Data Dictionary: Concepts and Uses*, QED Information Sciences, 1986.

WHITTEN, J., BENTLEY, L., AND HO, T. *Systems Analysis and Design Methods*, St. Louis: Times Mirror/Mosby, 1986.

WIEDERHOLD, G. *Database Design*, Second Edition, New York: McGraw-Hill, 1983.

WIEDERHOLD, G. "Knowledge and Database Management." *IEEE Software*, 8:1, January 1984.

WIEDERHOLD, G. "Views, Objects, and Databases." *Computer*, 19:12, December 1986.

WITKOWSKI, ANDREW, CARIÑO, FELIPE, AND KOSTAMAA, PEKKA. "NCR 3700 - The Next-Generation Industrial Database Computer." In VLDB 1993.

WYLIE, C. *101 Puzzles in Thought and Logic*, Mineola, NY: Dover Publications, Inc., 1957.

WINKLER-PARENTZ, H. B. "Can You Trust Your DBMS?" *Database Programming and Design*, July 1989.

WIORKOWSKI, G., AND D. KULL. "Distributed DB2." *Database Programming and Design*, April 1989.

WOOD, D. "A Primer of Features and Performance Issues of Relational DBMSs." *Data Resource Management*, 1:1, Winter 1990.

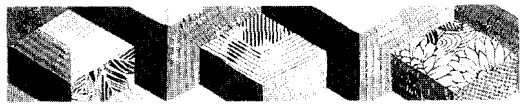
YAO, S. (editor). *Principles of Database Design, Volume 1: Logical Organizations*, Englewood Cliffs, NJ: Prentice Hall, 1985.

YOURDON, E., & CONSTANTINE, L. *Structured Design*, Englewood Cliffs, NJ: Prentice Hall, 1979.

ZANIOLO, C. ET AL. "Object-Oriented Database Systems and Knowledge Systems." In EDS 1984.

ZLOOF, M. "Query By Example." NCC, AFIPS, 44, 1975.

ZLOOF, M. M. "Query-by-Example: A Data Base Language." *IBM Systems Journal*, 16:4, 1977.



ÍNDICE ANALÍTICO

A

Acceso sólo para recuperación, 65
Administrador de archivos, 325
Agente, 394
Agregación, 445-446
Agregado, 111, 133
Agrupamiento, 350
Álgebra relacional, 178-98
Ancestro, 519
Anomalía de actualización, 148
Anomalías de borrado, 148
Anomalías de inserción, 148
Apuntador, 340-341
Árbol enraizado, 343
Archivo de acceso directo, 20
Arista, 489
Arquitectura tres-niveles, 68-69, 80
Atributo, 97-100

B

Banderas de estado, 501, 531
Base de datos centralizada, 57-58
Base de datos externa, 526
Base de datos orientada a objetos
Bloque de comunicación de programa, 526
Bloque de especificación de programa, 526, 527-528
Bloque de instrucciones, 294
Bloqueo de escritura, 373
Bloqueo de lectura, 373
Bloqueo, 371

C

Caja de condición, 260
Cálculo relacional, 198-204
Camino, 343
Campo, 38
Cardinalidad, 95-97, 102-103, 133
Catálogo, 212-213
Ciclo de vida del desarrollo de la base de datos (CVDBD), 62-63, 69, 71-72
Cifrado, 381-382, 424
Cilindro, 326-327
Clase derivada, 445
Clave candidata, 145
Clave externa, 99, 126, 133, 144
Clave foránea, 145-146
Clave primaria, 145, 353
Clave secundaria, 353
Clave subrogada, 92, 98-99, 144
Clave, 11, 98-99, 143-144
Cliente/servidor, 279-321
Conceptual, 68
Conectores booleanos, 221, 466-47
Conjunto de objetos abstractos, 92
Conjunto de objetos agregado, 94-95
Conocimiento extensional, 470
Conocimiento intensional, 470
Consulta conjuntiva, 271
Consulta externa, 227
Consultando por ejemplos (QBE), 20
Contraseña, 65, 375
Control de acceso, 423
Control de concurrencia, 370-375, 423

Controlador de disco, 23, 326
Controles de concurrencia, 370-375, 423
Controles de vistas, 423
Cuantificador universal, 202-204

D

Dato, 27
Datos globales, 394
Datos locales, 394
Dependencia funcional, 149
Dependencia transitiva, 152
Determinante, 149
Diario, 370
Diccionario/directorio de datos, 14, 25-26, 325, 422
Diseño conceptual de base de datos, 63-64, 68, 76-77, 86-138
Disparador, 301-303
Dominio, 143

E

Ejecución serial, 406
Encadenamiento hacia adelante, 472
Encadenamiento hacia detrás, 472
Enfoque orientado a datos, 70-71
Enfoque orientado a funciones, 70
Enlace físico, 494
Enlace, 394
Equirreunión, 194
Espera de la rotación, 329
Esquema de información, 246-247
Estándar, 366-367
Evaluación, 426-427

F

Factor de carga, 336
Forma Normal Boyce-Codd (FNBC), 152
Forma Normal dominio/Llave (FNDLL), 155
Fragmentación vertical, 400-401
Función de hash dinámico, 338-340

G

Grafo dirigido, 489

H

Hash extendible, 338
Hijo, 17, 516
Hoja, 343

I

Identidad del objeto, 92
Interrelación padre-hijo, 516
Instancia de objeto, 91
Instrucción de calificación, 199-200
Instrucción de iteración, 275
Instrucciones (sentencias) condicionales, 275
Integración de los datos, 55, 59
Integridad de los datos, 65, 148, 367
Integridad referencial, 147
Integridad, 367, 368-375
Interbloqueo, abrazo mortal, 372-373
Interfaz Gráfica de Usuario (GUI), 22, 281
Interoperatividad, 22
Interrelación de alto nivel, 111
Interrelación, 93-95

L

Lenguaje anfitrión, 242-243
Lenguaje de definición de datos (LDD), 497
Lenguaje de consulta estructurado (SQL), 20
Lenguaje de consulta, 27, 45-49
Lenguaje de datos lógico, 481
Lenguaje de flujo de control, 293-298
Lenguaje de manipulación de datos (LMD), 77, 497, 501-507
Lenguaje textual, 256
Lista enlazada, 340-342
Lista invertida, 342
Lista resultado, 199-200
Lógica de primer orden, 476

M

Manipulación de datos, 19-20
Marca de tiempo, 409-410
Memoria principal, 325-326
Metadatos, 26, 246
Método de acceso directo indizado jerárquico, 530
Método de acceso directo-jerárquico, 528-530
Método de acceso secuencial jerárquico, 528
Método de acceso secuencial-indizado, 528
Método del cociente cuadrático, 337
Modelado, 87-91
Modelo conceptual de datos, 90, 132
Modelo de datos en red, 487-513
Modelo de datos jerárquico, 514-531
Modelo de datos relacional, 139-173
Modelo orientado a objetos, 90
Muchos-muchos, 97, 133
Multiconjunto, 219

N

Nodo, 489
Normalización, 147-155

O

Objeto complejo, 445-446, 448
ODBC (Objetos de Conectividad de Base de Datos), 413
Orden navegacional, 540

P

Painter, 303
Pista, 326, 330
Predicado, 497
Procedimientos almacenados, 293
Procesamiento de acceso directo, 8-11
Procesamiento electrónico de datos (PED), 55
Progenitor, 17, 516
Programa de aplicación , 8
Programación orientada a objetos, 443-448
Protocolo de cierre de dos-fases, 406-407
Punto de chequeo, 386

R

Recorrido en preorden, 520-521
Red local, 58, 396
Red, 17
Registro lógico, 494
Regla, 283
Rendimiento, 65-66
Restricción de columna, 288
Restricción de tabla, 288
Restricción de valor, 63
Restricción, 214, 287-291
Retención fija, 507
Retención obligatoria, 507
Reunión externa, 190, 194
Reunión natural, 188-192, 240
Reunión theta, 192-194

S

Segmento hoja, 519
Segmento raíz, 518
Segmento sensible, 527-528
Segmentos gemelos, 519-520
Seguridad de los datos, 65, 367
Semirreunión, 404-405
Sistema de aplicación, 8
sistema de base de conocimientos (SBC), 442, 468-481
Sistema de base de datos distribuida, 58, 391-417
Sistema de base de datos orientado a objetos, 91, 441-467
Sistema de base de datos, 3-32
Sistema de bases de datos relacionales, 17-20
Sistema de gestión de base de conocimientos (SGBC), 468
Sistema de gestión de base de datos (SGBD), 12, 25-27, 59
Sistema de información para la gestión (MIS), 11, 55-56, 107
Sistema de procesamiento de datos, 7, 107
Sistema experto, 468
Sistema orientado a archivos, 6-14
Sistemas de soporte a la toma de decisiones, 55-56
Software de aplicación, 24-25
Solicitud de propuesta, 426
SQL incrustado, 242-243
Subconsulta correlacionada, 228-229

T

Tabla base, 243
Tabla de resultado, 258-259
Tabla de verificación de disparadores, 301
Teoría de la serialización, 406
Texto cifrado, 381-382
Texto legible, 381
Tiempo de activación de la cabeza, 329
Toma de decisiones, 11
Transacción abortada, 370
Transacción atómica, 369-370
Transacción cerrada, 382
Transacción local, 394
Transacción, 394
Tupla, 143

U

Uno-muchos, 97, 133
Uno-uno, 97, 133
Usuario, 27, 28

V

Valor atómico, 148
Variable local, 297-298
Vista, 26, 63, 131, 243

W

Window painter, 313-317