

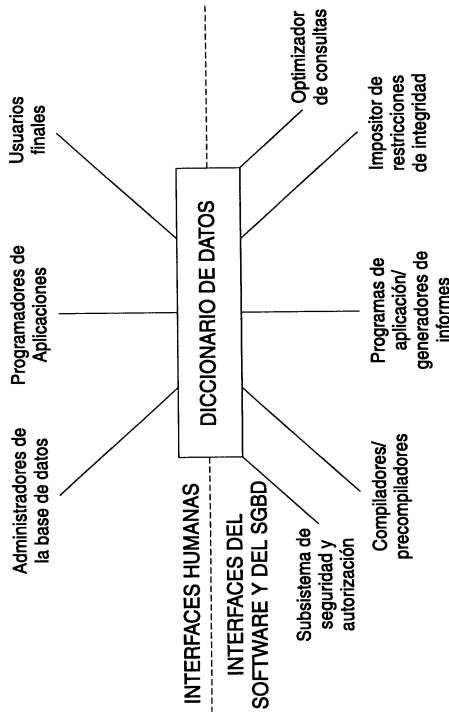


Figura 15.1 Interfaces humanas y de software con un diccionario de datos.

la descripción de la base de datos se cambia el contenido del diccionario de datos. Si se usa el diccionario de datos durante la fase de diseño, al terminar ésta los metadatos ya estarán en el diccionario.

En este capítulo nos concentraremos en estudiar del catálogo del sistema, y no los sistemas de diccionario de datos generales. En la sección 15.1 veremos los catálogos para los SGBD relacionales; luego, en la sección 15.2, hablaremos de los catálogos de red. En la sección 15.3 examinaremos cómo diversos módulos de un SGBD usan el catálogo, y describiremos otros tipos de información que puede estar almacenada en un catálogo. Como por sí mismo el catálogo es una minibase de datos, podemos describir su estructura haciendo referencia a un esquema en algún modelo de datos. Daremos una descripción de esquema conceptual para cada catálogo en el modelo EER (véase el Cap. 21) a fin de aclarar la estructura conceptual de los catálogos.

15.1 Catálogos para SGBD relacionales

La información almacenada en el catálogo de un SGBD relacional incluye descripciones de los nombres de las relaciones, nombres de los atributos, dominios (tipos de datos) de los atributos, claves primarias, atributos de clave secundaria, claves externas y otros tipos de restricciones, así como descripciones de nivel externo de las vistas y descripciones de nivel interno de las estructuras de almacenamiento e índices. También contiene información de seguridad y autorización, que especifica los permisos que tienen los usuarios para tener acceso a las relaciones y vistas de la base de datos, y quiénes son los creadores o propietarios de cada relación (véase el Cap. 20).

En los sistemas relacionales se acostumbra almacenar el catálogo mismo como relaciones y usar el software del SGBD para consultar, actualizar y mantener el catálogo. Esto permite a las rutinas del SGBD (y a los usuarios) tener acceso a la información almacenada en el catálogo siempre que cuenten con la autorización necesaria, empleando el lenguaje de consulta del SGBD.

CAPÍTULO 15

El catálogo del sistema

El catálogo del sistema constituye el núcleo de todo SGBD de aplicación general. Es una "minibase de datos" por sí misma, cuya función es almacenar los **esquemas**, o *descripciones*, de las bases de datos que el SGBD mantiene. Cada una de las bases de datos se describe por los datos almacenados en el catálogo, los cuales a menudo se denominan **metadatos**. El catálogo contiene una descripción del esquema conceptual de la base de datos, del esquema interno, de cualesquier esquemas externos y de las correspondencias entre los esquemas en los diferentes niveles. Además, contiene información que utilizan módulos específicos del SGBD; por ejemplo, el módulo de optimización de consultas o el módulo de seguridad y autorización.

Con el término **diccionario de datos** suele designarse una utilidad de software más general que un catálogo. Un **catálogo** está acoplado íntimamente al software del SGBD; proporciona la información que contiene a los usuarios y al DBA, pero lo utilizan *principalmente* los diversos módulos de software del SGBD mismo, como son los compiladores de DDL y DML, el optimizador de consultas, el procesador de transacciones, los generadores de informes y el módulo encargado de hacer que se cumplan las restricciones. Por otro lado, un paquete de software *autónomo* de **diccionario de datos** puede interactuar con los módulos del SGBD, pero lo utilizan *principalmente* los diseñadores, usuarios y administradores de un sistema para la gestión de los recursos de información. Los sistemas de diccionario de datos también sirven para mantener información relativa al hardware y software, la documentación y los usuarios del sistema, así como otra información pertinente para la administración del sistema. Si *sólo* los diseñadores, usuarios y administradores usan un diccionario de datos, y no el software del SGBD, se llama **diccionario de datos pasivo**; en caso contrario se denomina **diccionario de datos activo** o **directorio de datos**. La figura 15.1 ilustra los tipos de interfaces de los diccionarios de datos activos.

Estos diccionarios también se usan para documentar el proceso mismo de diseño de bases de datos, pues almacenan información sobre los resultados de todas las fases de diseño y sobre las decisiones de diseño. Esto ayuda a automatizar el proceso de diseño poniendo las decisiones y las modificaciones a disposición de todos los diseñadores. Para modificar

CATÁLOGO_DE_REL_Y_ATR

NOMBRE_REL	NOMBRE_ATR	TIPO_ATR	MIEMBRO_CLP	MIEMBRO_CLE	RELACIÓN_CLE
EMPLEADO	NOMBREP	VSTR15	no	no	
EMPLEADO	INIC	CHAR	no	no	
EMPLEADO	APELLIDO	VSTR15	no	no	
EMPLEADO	NSS	STR9	si	no	
EMPLEADO	FECHAN	VSTR30	no	no	
EMPLEADO	DIRECCIÓN	CHAR	no	no	
EMPLEADO	SEXO	CHAR	no	no	
EMPLEADO	SALARIO	INTEGER	no	no	
EMPLEADO	NSSUPER	STR9	no	no	
EMPLEADO	NUMD	INTEGER	no	si	EMPLEADO
DEPARTAMENTO	NOMBREP	VSTR10	no	si	DEPARTAMENTO
DEPARTAMENTO	NUMEROD	INTEGER	no	no	
DEPARTAMENTO	NSSGTE	STR9	si	no	
DEPARTAMENTO	FECHANICGTE	STR10	no	si	EMPLEADO
LUGARES_DEPTOS	NUMEROD	INTEGER	no	no	
LUGARES_DEPTOS	LUGARD	VSTR15	si	si	DEPARTAMENTO
PROYECTO	NOMBREPR	VSTR10	no	no	
PROYECTO	NUMEROP	INTEGER	si	no	
PROYECTO	LUGARP	VSTR15	no	no	
PROYECTO	NUMD	INTEGER	no	no	
TRABAJA_EN	NSSE	STR9	si	si	DEPARTAMENTO
TRABAJA_EN	NUMP	INTEGER	si	si	EMPLEADO
TRABAJA_EN	HORAS	REAL	no	no	
DEPENDIENTE	NSSE	STR9	si	si	PROYECTO
DEPENDIENTE	NOMBRE_DEPENDIENTE	VSTR15	no	no	
DEPENDIENTE	SEXO	CHAR	no	no	
DEPENDIENTE	FECHAN	STR9	no	no	
DEPENDIENTE	PARENTESCO	VSTR8	no	no	

Figura 15.2 Catálogo básico para el esquema relacional de la figura 6.5.

En la figura 15.2 se muestra una posible estructura de catálogo para almacenar información de las relaciones base, que incluye nombres de las relaciones, nombres de los atributos, tipos de los atributos e información relativa a las claves primarias. Esa figura también muestra cómo podrían incluirse las restricciones de clave externa. En la figura 6.5 vemos la descripción del esquema de base de datos relacional en forma de tuplas (contenido) del archivo de catálogo que se muestra en la figura 15.2, al cual llamamos CATÁLOGO_DE_REL_Y_ATR. La clave primaria de CATÁLOGO_DE_REL_Y_ATR es la combinación de atributos {NOMBRE_REL, NOMBRE_ATR}, porque todos los nombres de relación deben ser únicos y todos los nombres de atributo dentro de un esquema de relación particular también deben ser únicos. Otro archivo de catálogo puede almacenar información sobre cada relación, como el tamaño y el número de las tuplas, el número de índices y el nombre del creador.

Si queremos incluir información sobre los atributos de clave secundaria de una relación, basta con extender el catálogo anterior si suponemos que un atributo puede ser miembro de sólo una clave. En este caso podemos reemplazar el atributo MIEMBRO_CLP de CATÁLOGO_DE_REL_Y_ATR por un atributo NÚMERO_CLAVE; el valor de este atributo es 0 si el atributo no es miembro de ninguna clave, 1 si es miembro de la clave primaria, e $i > 1$ si es miembro de la i -ésima clave secundaria, donde las claves secundarias de una relación están numeradas 2, 3, ..., n . Sin embargo, si un atributo puede ser miembro de más de una clave, lo cual es el caso general, la representación anterior no basta. Una posibilidad es almacenar información sobre los atributos de clave por separado en una segunda relación de catálogo CLAVES_RELACIÓN con los atributos {NOMBRE_REL, NÚMERO_CLAVE, ATR_MIEMBRO} que, juntos, constituyen la clave de CLAVES_RELACIÓN. Esto se muestra en la figura 15.3(a). El

(a) CLAVES_RELACIÓN

NOMBRE_REL	NÚMERO_CLAVE	ATR_MIEMBRO
------------	--------------	-------------

(b) INDICES_RELACIÓN

NOMBRE_REL	NOMBRE_INDICE	ATR_MIEMBRO	TIPO_INDICE	NÚM_ATR	ASC_DESC
------------	---------------	-------------	-------------	---------	----------

(c) CONSULTAS_VISTA

NOMBRE_VISTA	CONSULTA
--------------	----------

ATRIBUTOS_VISTA

NOMBRE_VISTA	NOMBRE_ATR	NÚM_ATR
--------------	------------	---------

Figura 15.3 Otras posibles relaciones de catálogo para un sistema relacional. (a) Relación de catálogo para almacenar información general sobre claves. (b) Relación de catálogo para almacenar información sobre índices. (c) Relaciones de catálogo para almacenar información sobre vistas.

compilador de DDL asigna el valor 1 a NÚMERO_CLAVE para la clave primaria y los valores 2, 3, ..., n para las claves secundarias. Cada clave tendrá una tupla en CLAVES_RELACIÓN para cada atributo que sea parte de esa clave, y el valor de ATR_MIEMBRO dará el nombre de tal atributo. Se puede usar una estructura similar para almacenar información relativa a las claves externas.

Consideremos ahora la información relativa a los índices. En el caso general, en el que un atributo puede ser miembro de más de un índice, puede usarse la relación de catálogo INDICES_RELACIÓN que se muestra en la figura 15.3(b). La clave de INDICES_RELACIÓN es la combinación {NOMBRE_REL, NOMBRE_INDICE, ATR_MIEMBRO} (suponiendo que los nombres de los índices son únicos). ATR_MIEMBRO es el nombre de un atributo incluido en el índice. Por ejemplo, si especificamos tres índices para la relación TRABAJA_EN de la figura 6.5 —un índice de agrupamiento según NSSE, un índice secundario según NÚMP y otro índice secundario según la combinación {NSSE, NÚMP}—, los atributos NSSE y NÚMP son miembros de dos índices cada uno. Los campos NÚM_ATR y ASC_DESC de INDICES_RELACIÓN especifican el orden de cada atributo en el índice y especifican si el atributo se ordena en forma ascendente o descendente dentro del índice.

También hay que almacenar las definiciones de vistas en el catálogo. Una vista se especifica con una consulta, posiblemente cambiando el nombre de los valores que aparecen en el resultado de la consulta (véase el Cap. 7). Con las dos relaciones de catálogo que se muestran en la figura 15.3(c) podemos almacenar definiciones de vistas. La primera, CONSULTAS_VISTA, tiene dos atributos, {NOMBRE_VISTA, CONSULTA}, y almacena la consulta (la cadena de texto completa) que corresponde a la vista. La segunda, ATRIBUTOS_VISTA, tiene los atributos {NOMBRE_VISTA, NOMBRE_ATR, NÚM_ATR} para almacenar los nombres de los atributos de la vista, donde NÚM_ATR es un número entero mayor que cero que especifica la correspondencia entre cada atributo de la vista y los atributos del resultado de la consulta. La clave de CONSULTAS_VISTA es NOMBRE_VISTA, y la de ATRIBUTOS_VISTA es la combinación {NOMBRE_VISTA, NOMBRE_ATR}.

Los ejemplos anteriores ilustran los tipos de información que se almacenan en el catálogo, el cual por lo regular contiene muchos más archivos e información. En su mayoría, los sistemas relacionales almacenan sus archivos de catálogo como relaciones del SGBD, sin embargo, como los módulos del SGBD tienen acceso muy a menudo al catálogo, es importante implementar este acceso de la manera más eficiente posible. Quizá sea más eficiente implementar el catálogo mediante un conjunto especializado de estructuras de datos y rutinas de acceso, sacrificando así la generalidad por la eficiencia.

Por último, daremos un vistazo a la información básica almacenada en un catálogo relacional desde el punto de vista conceptual. La figura 15.4 muestra un diagrama de esquema ER de alto nivel (véanse los Caps. 3 y 21) que describe parte de un catálogo relacional. Ahí el tipo de entidades RELACIÓN almacena los nombres de las relaciones que aparecen en un esquema relacional. Se crean dos subclases disjuntas, RELACIÓN_BASE y RELACIÓN_VISTA, para RELACIÓN. El tipo de entidades ATRIBUTO es un tipo de entidades débil de RELACIÓN, y su clave parcial es NombreAtr. Las RELACIÓN_BASE también tienen restricciones de clave general y de clave externa, además de índices, mientras que las RELACIÓN_VISTA tienen su consulta definidora (así como el NúmAtr) para especificar la correspondencia entre los atributos de la vista y los atributos de la consulta. Observe que una restricción adicional, no especificada en la figura 15.4, es que todos los atributos relacionados con una entidad CLAVE o ÍNDICE —a través de los vínculos ATRS_CLAVE o ATRS_ÍNDICE— deben estar relacionados con la misma entidad RELACIÓN_BASE con la que está relacionada la entidad CLAVE o ÍNDICE.

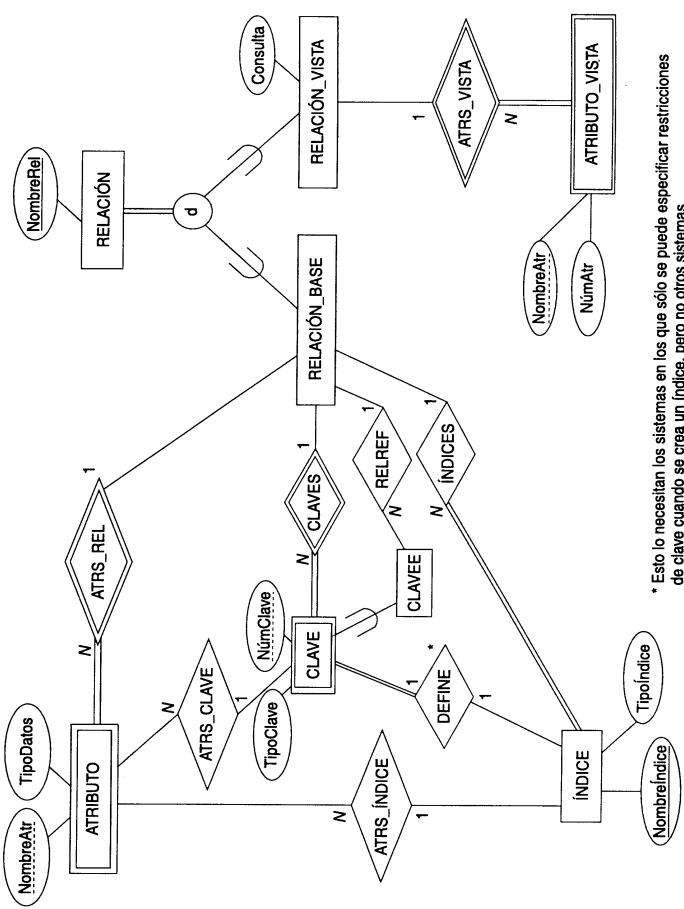


Figura 15.4 Ejemplo de diagrama ER extendido para una parte de un catálogo relacional.

TipoClave específica si la clave es externa, primaria o secundaria. CLAVE es una subclase para las claves externas y se relaciona con la relación referida a través de RELREF.

Analizaremos en la sección 15.3 la demás información que se debe almacenar en un catálogo, y trataremos la seguridad y la autorización en el capítulo 20.

15.2 Catálogos para SGBD de red

La información básica que debe almacenarse en el catálogo de un SGBD de red es una descripción de los tipos de registros y de conjuntos. La información sobre la implementación de cada tipo de conjuntos y otras elecciones de almacenamiento físico también debe incluirse en el catálogo, junto con la información de seguridad y autorización y la relativa a los subesquemas externos. La figura 15.5 muestra un posible catálogo de red para describir los tipos de

CATÁLOGOS_TIPOS_REG

NOMBRE_TIPO_REG	NOMBRE_CAMPO	TIPO_CAMPO
EMPLEADO	NOMBREP	VSTR15
EMPLEADO	INIC	CHAR
EMPLEADO	APELLIDO	VSTR15
EMPLEADO	NSS	STR9
EMPLEADO	FECHAN	STR9
EMPLEADO	DIRECCIÓN	VSTR30
EMPLEADO	SEXO	CHAR
EMPLEADO	SALARIO	INTEGER
EMPLEADO	NOMBREDEPTO	VSTR10
DEPARTAMENTO	NOMBRE	VSTR10
DEPARTAMENTO	NÚMERO	INTEGER
DEPARTAMENTO	LUGARES	VSTR15
DEPARTAMENTO	INICGTE	STR10
PROYECTO	NOMBRE	VSTR10
PROYECTO	NÚMERO	INTEGER
PROYECTO	LUGAR	VSTR15
PROYECTO	NSSE	STR9
TRABAJA_EN	NÚMEROP	INTEGER
TRABAJA_EN	HORAS	REAL
TRABAJA_EN	NSSEMP	STR9
DEPENDIENTE	NOMBRE	VSTR15
DEPENDIENTE	SEXO	CHAR
DEPENDIENTE	FECHAN	STR9
DEPENDIENTE	PARENTESCO	STR10
SUPERVISOR	NSS_SUPERVISOR	STR9

CATÁLOGO_TIPOS_CONJ

NOMBRE_TIPO_CONJ	TIPO_REGISTRO_PROPIETARIO	TIPO_REGISTRO_MIEMBRO
TODOS_DEPTOS	SISTEMA	DEPARTAMENTO
PERTENECE_A	EMPLEADO	DEPARTAMENTO
DIRIGE	DEPARTAMENTO	PROYECTO
CONTROLA	DEPARTAMENTO	SUPERVISOR
ES_SUPERVISOR	EMPLEADO	EMPLEADO
SUPERVISADOS	SUPERVISOR	EMPLEADO
E_TRABAJAEN	EMPLEADO	TRABAJA_EN
P_TRABAJAEN	PROYECTO	TRABAJA_EN
DEPENDIENTES_DE	DEPENDIENTE	DEPENDIENTE

Figura 15.5 Archivos de catálogo básicos para almacenar información sobre tipos de registros y de conjuntos.

(a) CATÁLOGOS_TIPOS_CONJUNTOS

NOMBRE_TIPO_CONJ	...	APUNT_SIGTE	APUNT_PROPIETARIO	APUNT_PREVIO	MIEMBROS_CONTIGUOS
------------------	-----	-------------	-------------------	--------------	--------------------

(b) CATÁLOGOS_TIPOS_CONJUNTOS

<u>NOMBRE_TIPO_CONJ</u>	...	OPCIÓN_RETENCIÓN	OPCIÓN_INSERTIÓN
-------------------------	-----	------------------	------------------

(c) CATÁLOGOS TIPOS REGISTROS

NOMBRE_TIPO_REG	NOMBRE_CAMPO	TIPO_CAMPO	VECTOR	GRUPO_REPET
-----------------	--------------	------------	--------	-------------

COMPONENTES CAMPO COMPLEJO

NOMBRE_TIPO_REG	NOMBRE_CAMPO	NOMBRE_CAMPO_COMPONENTE
-----------------	--------------	-------------------------

RESTRICCIONES ÚNICO (SIN DUPLICADOS)

NOMBRE_TIPO_REG	NÚMERO_CLAVE	NOMBRE_CAMPO
-----------------	--------------	--------------

Figura 15.6 Algunas extensiones del catálogo de red básico. (a) Extensión de `CATALOGO_TIPOS_CONJ` para incluir información sobre la implementación de conjuntos. (b) Inclusión de las opciones de tipo de conjuntos en el archivo `CATALOGO_TIPOS_CONJ`. (c) Representación de campos repetitivos y vectoriales y de la restricción `NO DUPLICATES ALLOWED`.

registros y de conjuntos. El catálogo consta de dos archivos —uno para describir los tipos de registros y el otro para definir los tipos de conjuntos— y contiene la información mínima necesaria para describir un esquema de base de datos de red. El contenido de estos dos archivos corresponde al esquema de red de la figura 10.9.

Es fácil extender el CATÁLOGO_TIPOS_CONI agregando información sobre cómo implementar cada tipo de conjuntos. Por ejemplo, podríamos añadir los campos indicadores APUNT_SIGTE, APUNT_PROPIETARIO, APUNT_PREVIO, MIEMBROS_CONTIGUOS, etc., cada uno de los cuales tendría el valor FALSO o VERDADERO. Esto se ilustra en la figura 15.6(a). Las opciones de conjunto para cada tipo de conjuntos también deben almacenarse en el catálogo, agregando los campos OPCIÓN_RETENCIÓN y OPCIÓN_INSERTACIÓN (véase la Fig. 15.6(b)), donde los posibles valores de OPCIÓN_RETENCIÓN son {MANDATORY, OPTIONAL, FIXED} y los de OPCIÓN_INSERTACIÓN son {AUTOMATIC, MANUAL}. También debemos especificar en el catálogo cualesquier restricciones estructurales que se apliquen a los tipos de conjuntos, y el método SET SELECTION IS AUTOMATIC para los tipos de conjuntos AUTOMATIC.

Así mismo, debemos poder representar los campos compuestos de **grupo**, como los campos vectoriales y los grupos repetitivos. Además, hay que asentar en el catálogo las restricciones sobre los campos, como la opción **NO DUPLICATES ALLOWED** que se especifica para los campos clave. La figura 15.6(c) ilustra cómo podemos representar esta restricción empleando la misma técnica que nos sirvió para representar claves generales en un catálogo relacional. Esta figura también muestra cómo representar campos de grupo en el catálogo. Los campos **VECTOR** y **GRUPO_REPET** de **CATÁLOGO_TIPOS** son campos indicadores cuyos valores son **VERDADERO** o **FALSO**. Los campos componentes de un campo de grupo (compuesto) se almacenan en el archivo **COMPONENTES_CAMPO_COMPUESTO**.

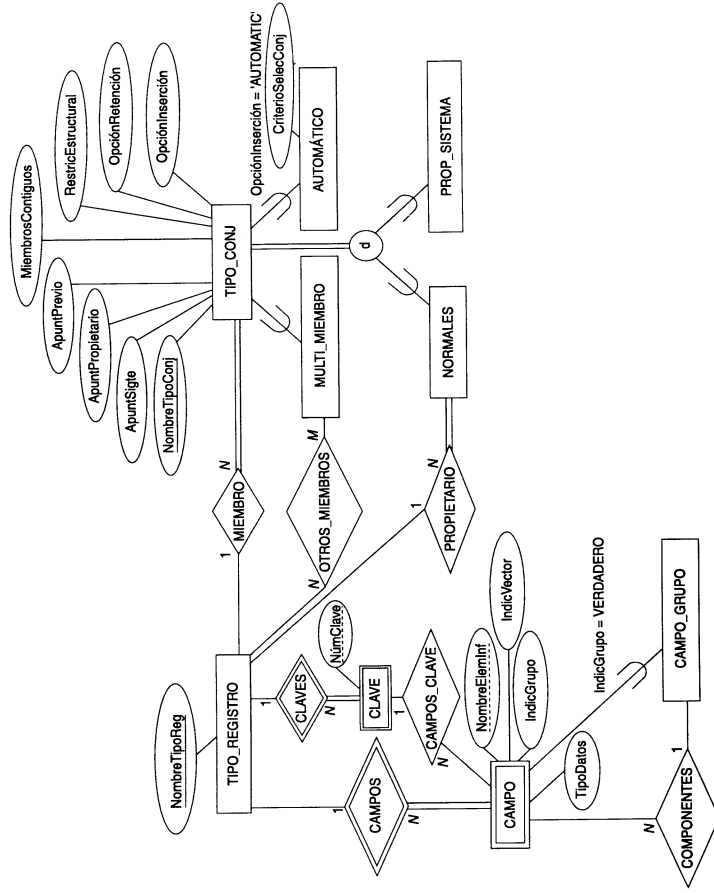


Figura 15.7 Diagrama EER para una parte de un catálogo de red.

Como hicimos con el modelo relacional, en la figura 15.7 mostramos una descripción conceptual de parte de un catálogo de sistema de red como esquema EER (véanse los Caps. 3 y 21). El conjunto de vínculo MIEMBRO relaciona una entidad TIPO_CONJ con su entidad miembro TIPO_REGISTRO. Los tipos de conjuntos MULTI_MIEMBRO tienen tipos de registros miembro adicionales. Los tipos de conjuntos PROP_SISTEMA no tienen un tipo de registros propietario, pero otros tipos de conjuntos, que en la figura 15.7 llamamos NORMALES, están relacionados a través del conjunto de vínculo PROPIETARIO con su entidad propietaria TIPO_REGISTRO. Los tipos de conjuntos AUTOMATIC son una subclase de TIPO_CONJ con la condición de pertenencia OpciónInserción = 'AUTOMATIC', y tienen el atributo específico adicional CriterioSelección. Los CAMPO GRUPO son una subclase de CAMPO que constan de COMPONENTES.

15.3 Otra información de catálogo utilizada por módulos de software del SGBD

Los módulos del SGBD usan y leen el catálogo con mucha frecuencia; por ello es importante implementar el acceso al catálogo de la forma más eficiente posible. En esta sección estudiaremos las diferentes formas en que algunos módulos de software del SGBD usan y leen el catálogo. Dichos módulos incluyen:

1. Compiladores de DDL (y SDL): Estos módulos del SCBD procesan y verifican la especificación de un esquema de base de datos escrita en el lenguaje de definición de datos y almacenan esa definición en el catálogo. Las construcciones y restricciones del esquema en todos los niveles —conceptual, interno y externo— se extraen de las especificaciones de DDL (lenguaje de definición de datos) y de SDL (lenguaje de definición de almacenamiento) y se introducen en el catálogo, lo mismo que cualquier información de correspondencia entre los niveles, si es necesaria. Así pues, estos módulos de software realmente *poseen* la minibase de datos del catálogo (o *meta-base de datos*) con datos que son las descripciones de los esquemas de la base de datos.
2. Analizador sintáctico y verificador de consultas y DML: Estos módulos analizan sintácticamente las consultas, las instrucciones de obtención en DML y las instrucciones de actualización de la base de datos, y examinan el catálogo para verificar que todos los nombres de esquema a los que se hace referencia en todas estas instrucciones sean válidos. Por ejemplo, en un sistema relacional, un analizador de consultas verificaría que todos los nombres de relación especificados en la consulta existan en el catálogo y que los atributos especificados pertenezcan a las relaciones apropiadas. De manera similar, en un sistema de red, cualesquier tipos de registros o de conjuntos a los que se haga referencia en órdenes de DML se extraerán del catálogo, y se verificarán dichas órdenes.
3. Compilador de consultas y de DML: Estos compiladores convierten las consultas y órdenes DML de alto nivel en órdenes de bajo nivel de acceso a archivos. Durante este proceso, se tiene acceso a la correspondencia entre el esquema conceptual y las estructuras de archivos del esquema interno, la cual está almacenada en el catálogo. Por ejemplo, el catálogo debe incluir una descripción de cada archivo y de sus campos, y de las correspondencias entre los campos y los atributos del nivel conceptual.
4. Optimizador de consultas y de DML: El optimizador de consultas tiene acceso al catálogo para obtener información sobre caminos e implementación, a fin de determinar la mejor manera de ejecutar una consulta u orden DML (véase el cap. 16). Por ejemplo, el optimizador tiene acceso al catálogo para verificar cuáles campos de una relación tienen acceso por dispersión o índices, antes de decidir cómo ejecutar una condición de selección o de reunión sobre esa relación. De manera similar, una orden DML de procesamiento de conjuntos en una base de datos de red, como FIND OWNER, hace que se examine el catálogo para determinar si existe un apuntador al propietario para el tipo de registros miembro o si se debe seguir los apuntadores al siguiente hasta llegar al registro propietario.
5. Comprobación de autorización y seguridad: El DBA cuenta con órdenes privilegiadas para actualizar la porción de autorización y seguridad del catálogo (véase el Cap. 20). Cada vez que un usuario trata de tener acceso a una relación o tipo de registros, el SCBD examina el catálogo para verificar que tenga los permisos necesarios.
6. Correspondencia externa-interna de las consultas y órdenes de DML: Las consultas y órdenes de DML que se especifican haciendo referencia a una vista o esquema externo deben transformarse de modo que hagan referencia al esquema conceptual, antes de que el SCBD las pueda procesar. Esto se logra leyendo la descripción de la vista en el catálogo para poder efectuar la transformación.

Prácticamente todos los módulos de software del SCBD tienen acceso con mucha frecuencia a la información almacenada en el catálogo. La información que analizamos en las secciones 15.1 y 15.2 es apenas la información básica. Los SCBD más complejos necesitan almacenar información adicional en el catálogo, como el número de registros que hay en cada relación base o tipo de registros, la selectividad media de los diferentes campos de una relación base, el número de niveles que tiene un índice, y el número promedio de miembros en un ejemplar de conjunto de un tipo de conjuntos. El SCBD debe actualizar automáticamente esta información. Además, un sistema de catálogo/diccionario de datos expandido podría incluir información útil para los usuarios del sistema de base de datos, como son las decisiones de diseño y sus justificaciones. Es evidente que el catálogo es un componente muy importante de todo SCBD generalizado.

15.4 Resumen

En este capítulo estudiamos el tipo de información que se incluye en un catálogo de SCBD. En la sección 15.1 vimos la estructura del catálogo de un SCBD relacional y mostramos cómo puede almacenar las construcciones del modelo relacional, incluida información sobre restricciones de clave, índices y vistas. También presentamos una descripción conceptual —en forma de diagrama de esquema EER— de las construcciones del modelo relacional y las relaciones que hay entre ellas. En la sección 15.2 tratamos las construcciones de los catálogos de los sistemas de red y presentamos un diagrama conceptual EER para describir los conceptos del modelo de red. El modelo jerárquico se analiza en los ejercicios. En la sección 15.3 vimos cómo es que los distintos módulos del SCBD tienen acceso a la información almacenada en el catálogo. También mencionamos otros tipos de información almacenada en el catálogo.

Preguntas de repaso

- 15.1. ¿Qué significa el término *metadatos*?
- 15.2. ¿Cómo suelen implementarse los catálogos de SCBD relacionales?
- 15.3. Analice los tipos de información incluidos en un catálogo relacional en los niveles conceptual, interno y externo.
- 15.4. Analice los tipos de información incluidos en un catálogo de red en los niveles conceptual, interno y externo.
- 15.5. Explique la forma en que algunos de los diferentes módulos del SCBD tienen acceso al catálogo e indique la clase de información que utilizan.
- 15.6. ¿Por qué es importante que el acceso a un catálogo de SCBD sea eficiente?

Ejercicios

- 15.7. Expanda el catálogo relacional de la figura 15.3 de modo que incluya una descripción más completa de un esquema relacional, más descripciones internas de los archivos de almacenamiento y cualquier información de correspondencia que se necesite.

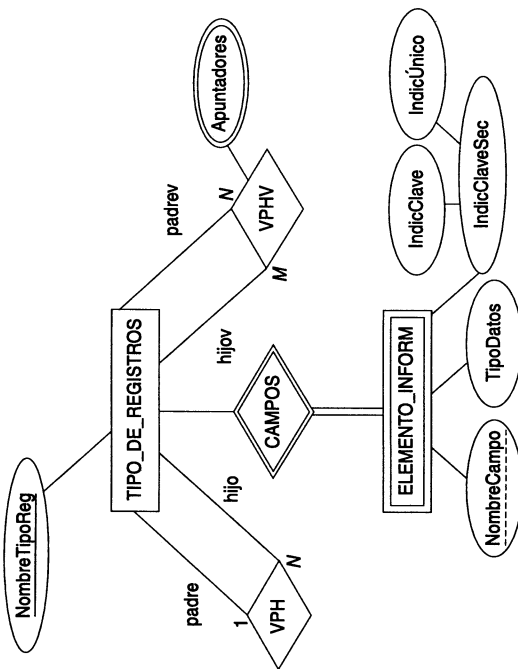


Figura 15.8 Esquema ER para un catálogo jerárquico básico.

15.8. Utilice los algoritmos de transformación que vimos en los capítulos 6 y 10 para crear un esquema relacional equivalente para las figuras 15.4 y 15.7. ¿Para qué pueden servir estos esquemas relacionales?

15.9. Escriba (en español) ejemplos de consultas de los esquemas EER de las figuras 15.4 y 15.7 que obtengan del catálogo información importante sobre los esquemas de base de datos.

15.10. A partir de los esquemas relacionales del ejercicio 15.8, escriba en algún lenguaje de consulta relacional (SQL, QUEL, álgebra relacional) las consultas que especificó en el ejercicio 15.9.

15.11. La figura 15.8 muestra un diagrama de esquema ER que representa la información básica contenida en un catálogo de un sistema de bases de datos jerárquico. Transforme este esquema a un conjunto de relaciones, y repita los ejercicios 15.9 y 15.10 con este esquema.

15.12. Suponga que tiene un SGBD "generalizado" que usa el modelo EER en el nivel de esquema conceptual y archivos similares a relaciones en el nivel interno. Dibuje un diagrama EER que represente la información fundamental de un catálogo para un sistema de bases de datos EER de ese tipo. Describa primero los conceptos de EER como un esquema EER (I), y luego añada información de correspondencia del esquema conceptual al esquema interno dentro del catálogo.

CAPÍTULO 16

Procesamiento y optimización de consultas

En este capítulo examinaremos las técnicas con que los SGBD procesan, optimizan y ejecutan las consultas de alto nivel. Para expresar una consulta en un lenguaje de alto nivel como SQL, primero debe pasar por un examen o análisis léxico, un análisis sintáctico y una validación.¹ El **analizador léxico** identifica los componentes del lenguaje (componentes léxicos) en el texto de la consulta, y el **analizador sintáctico** revisa la sintaxis de la consulta para determinar si está formulada de acuerdo con las reglas sintácticas (reglas gramaticales) del lenguaje de consulta. Además la consulta se debe **validar**, para lo cual ha de comprobarse que todos los nombres de atributos y de relaciones sean válidos y tengan sentido desde el punto de vista semántico. A continuación se crea una representación interna de la consulta, por lo regular en forma de estructura de datos de árbol o de grafo; esto se denomina **árbol (o grafo) de consulta**. En seguida, el SGBD debe crear una **estrategia de ejecución** para obtener el resultado de la consulta a partir de los archivos internos de la base de datos. Por lo regular, una consulta tiene muchas posibles estrategias de ejecución, y el proceso de elegir la más adecuada para procesar una consulta se conoce como **optimización de consultas**.

La figura 16.1 muestra los diferentes pasos para procesar una consulta de alto nivel. El módulo **optimizador de consultas** se encarga de producir un plan de ejecución, y el **generador de código** genera el código necesario para ejecutarlo. El **procesador de base de datos en tiempo de ejecución** se encarga de ejecutar el código de la consulta, sea compilado o interpretado, para producir el resultado de la consulta. Si se presenta un error durante la ejecución, este procesador genera un mensaje de error.

En realidad, el término *optimización* no es correcto porque, en algunos casos, el plan de ejecución elegido no es la estrategia óptima (la mejor); es tan sólo una **estrategia**

¹No estudiaremos aquí la fase de análisis y comprobación de la sintaxis dentro del proceso de consultas; este tema se trata en casi todos los textos sobre compiladores.

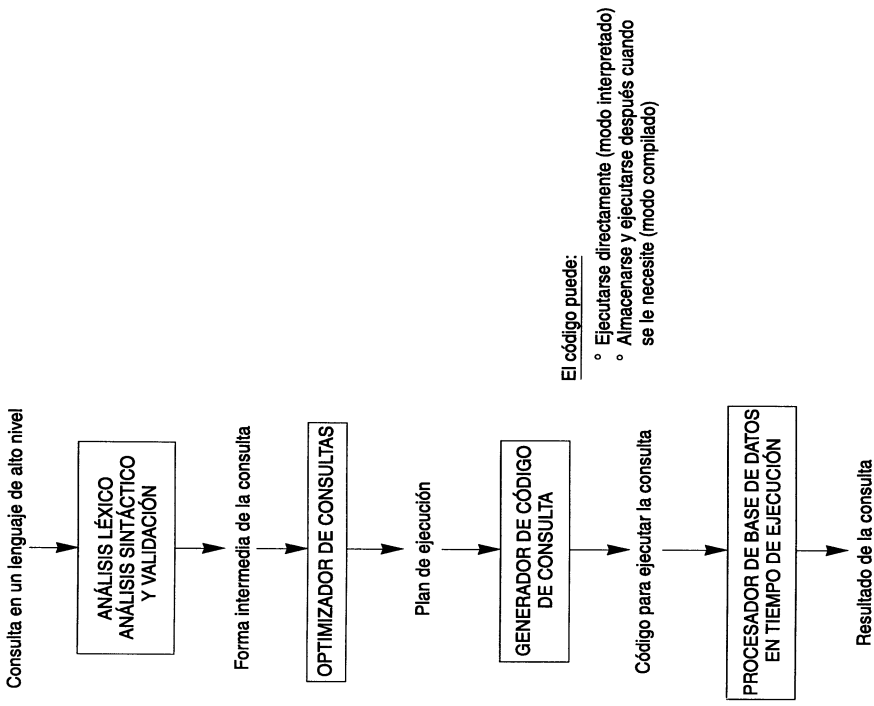


Figura 16.1 Pasos del procesamiento de una consulta de alto nivel.

razonablemente eficiente para ejecutar la consulta. En general, encontrar la estrategia óptima consume demasiado tiempo si la consulta no es muy simple, y puede requerir información sobre cómo están implementados los archivos o incluso sobre el contenido de éstos; información que tal vez no esté disponible en el catálogo del SCBD. Así pues, tal vez sea más correcto el término “planificar una estrategia de ejecución” que “optimizar una consulta” para el asunto que nos ocupará en este capítulo.

En los lenguajes de bajo nivel para navegar la base de datos, como el DML de red o el HDML jerárquico (véanse los Caps. 10 y 11), el programador debe elegir una estrategia de ejecución para su consulta en el momento de escribir un programa de base de datos. Si un SCBD sólo cuenta con un lenguaje de recorrido (de navegación), *la necesidad y las oportunidades* de optimización extensa de las consultas por parte del SCBD son *limitadas*; más bien, al programador se le ofrece la posibilidad de elegir la estrategia de ejecución “óptima”. Por

otro lado, por su naturaleza una consulta relacional de alto nivel es más declarativa: específica cuál es el resultado que se desea obtener de la consulta, más que detallar cómo debe obtenerse. Por ello, la optimización es necesaria en las consultas relacionales de alto nivel. Un SCBD relacional debe evaluar sistemáticamente estrategias alternativas de ejecución de una consulta y escoger la óptima.

Por lo regular, todo SCBD cuenta con varios algoritmos generales de acceso a la base de datos que implementan operaciones relacionales como SELECCIONAR o REUNIÓN o sus combinaciones. El módulo de optimización de consultas sólo puede considerar las estrategias de ejecución que puedan poner en práctica los algoritmos de acceso del SCBD y que se apliquen al diseño particular de la base de datos. En la sección 16.1 analizaremos los algoritmos para implementar operaciones relacionales.

Son dos las técnicas principales para optimizar consultas. La primera se basa en **reglas heurísticas** para ordenar las operaciones en una estrategia de ejecución de consulta. Por lo regular, las reglas reordenan las operaciones en un árbol de consulta (que se define en la sección 16.2.1) o determinan un orden para ejecutar las operaciones con base en un grafo de consulta (que se define en la sección 16.2.2). La segunda técnica implica **estimar sistemáticamente** el costo de diferentes estrategias de ejecución y elegir el plan con el más bajo costo estimado. Las dos estrategias suelen combinarse en el optimizador de consultas. Hablaremos de la optimización heurística en la sección 16.2 y de la estimación de costos en la sección 16.3.

16.1 Algoritmos básicos para ejecutar operaciones de consulta

Un SCBD relacional, o un SCBD no relacional con una interfaz de lenguaje de consulta relacional de alto nivel, debe contar con **algoritmos** para implementar los tipos de operaciones relacionales que pueden aparecer en una estrategia de ejecución de consulta. Éstas incluyen las operaciones básicas del álgebra relacional que vimos en el capítulo 6 y, en muchos casos, combinaciones de ellas. El SCBD debe contar además con algoritmos para procesar operaciones especiales, como las funciones agregadas y la agrupación. Para cada una de estas operaciones o combinación de operaciones, se dispone de uno o más algoritmos para ejecutarla. Un algoritmo puede aplicarse a estructuras de almacenamiento y caminos de acceso específicos; en tal caso, sólo podrá usarse si los archivos que intervienen en la operación incluyen estos caminos de acceso (véanse los Caps. 4 y 5). En esta sección examinaremos algoritmos representativos con los que se implementan SELECCIONAR, REUNIÓN y otras operaciones relacionales.

16.1.1 Implementación de la operación SELECCIONAR

Hay muchas opciones para ejecutar una operación SELECCIONAR; algunas dependen de que el archivo tenga caminos de acceso específicos y posiblemente sólo se apliquen a ciertos tipos de condiciones de selección. Aquí examinaremos algunos de los algoritmos para implementar SELECCIONAR. Para ilustrar nuestro análisis usaremos las siguientes operaciones, especificadas sobre la base de datos relacional de la figura 6.5:

(OP1): $\sigma_{NSS=123456789}$ (EMPLEADO)
 (OP2): $\sigma_{NUMEROD=5}$ (DEPARTAMENTO)
 (OP3): $\sigma_{ND=5}$ (EMPLEADO)
 (OP4): $\sigma_{ND=5 \text{ AND SALARIO} > 30000 \text{ AND SEXO} = F}$ (EMPLEADO)
 (OP5): $\sigma_{NSSE=123456789 \text{ AND NUMP}=10}$ (TRABAJA_EN)

Métodos de búsqueda para seleccionar

Hay varios algoritmos de búsqueda para seleccionar registros de un archivo, a los cuales se les conoce también como **examinadores de archivos** porque examinan los registros de un archivo para encontrar y leer registros que satisfacen una condición de selección. Si el algoritmo de búsqueda implica el empleo de un índice, la búsqueda en el índice se denomina **examinador de índice**. Los siguientes métodos de búsqueda (S1 a S6) son ejemplos de los algoritmos de búsqueda con que se puede implantar una operación de selección:

- S1. Búsqueda lineal (fuerza bruta): Leer *todos* los registros del archivo, y probar si los valores de sus atributos satisfacen la condición de selección.
- S2. Búsqueda binaria: Si la condición de selección implica una comparación de igualdad sobre un atributo clave según el cual está ordenado el archivo, se puede usar la búsqueda binaria (más eficiente que la lineal). Un ejemplo es OP1 si NSS es el atributo de ordenamiento del archivo EMPLEADO.
- S3. Empleo de un índice primario o una clave de dispersión para obtener un solo registro: Si la condición de selección implica una comparación de igualdad sobre un atributo clave que tiene un índice primario (o es una clave de dispersión) —por ejemplo, NSS = 123456789 en OP1—, se usa el índice primario (o la clave de dispersión) para obtener el registro.
- S4. Empleo de un índice primario para leer múltiples registros: Si la condición de comparación es $>$, $\geq$, $<$ o $\leq$ sobre un campo clave que tiene un índice primario —por ejemplo, NUMEROD $>$ 5 en OP2—, se usa el índice para encontrar el registro que satisfaga la condición de igualdad correspondiente (NUMEROD = 5), y luego se leen todos los registros subsiguientes del archivo (ordenado). Si la condición es NUMEROD $<$ 5, se leen todos los registros precedentes.
- S5. Empleo de un índice de agrupamiento para leer múltiples registros: Si la condición de selección implica una comparación de igualdad sobre un atributo no clave que tiene un índice de agrupamiento —por ejemplo, ND = 5 en OP3—, se usa este índice para obtener todos los registros que satisfagan la condición de selección.
- S6. Empleo de un índice secundario (árbol B*): Si se trata de una comparación de igualdad, se puede usar este método de búsqueda para obtener un solo registro si el campo de indización tiene valores únicos (es una clave) o para obtener múltiples registros si el campo de indización no es una clave. Además, puede servir para obtener registros que satisfagan condiciones en las que intervengan $>$, $\geq$, $<$ o $\leq$.

En la sección 16.3.3 veremos cómo se pueden crear fórmulas para estimar el costo de acceso que tienen estos métodos de búsqueda en términos del número de accesos a bloques y del tiempo de acceso implicado. El método S1 se aplica a cualquier archivo, pero todos los demás métodos dependen de que exista el camino de búsqueda apropiado según los atributos que

intervienen en la condición de selección. Los métodos S4 y S6 también pueden servir para leer registros dentro de un cierto intervalo; por ejemplo, $30\,000 \leq \text{SALARIO} \leq 35\,000$. Las consultas determinadas por tales condiciones se denominan **consultas de intervalo**.

Si la condición de una operación SELECCIONAR es una **condición conjuntiva** —esto es, si se compone de varias condiciones simples unidas por el conectivo lógico AND, como OP4—, el SCBD puede usar también los métodos siguientes para efectuar la operación:

- S7. Selección conjuntiva: Si un atributo que interviene en cualquier *condición simple* de la condición conjuntiva tiene un camino de acceso que permite emplear uno de los métodos S2 a S6, utilícese esa condición para obtener los registros y después verifíquese individualmente si satisfacen las demás condiciones simples de la condición conjuntiva.
- S8. Selección conjuntiva con un índice compuesto: Si dos o más atributos intervienen en condiciones de igualdad en la condición conjuntiva y existe un índice compuesto (o estructura de dispersión) sobre los campos combinados —por ejemplo, si se ha creado un índice según la clave compuesta (NSSE, NUMP) del archivo TRABAJA_EN para OP5—, se puede usar ese índice directamente.
- S9. Selección conjuntiva por intersección de apuntadores a registros:[†] Este método es posible si se dispone de índices secundarios basados en todos (o en algunos de) los campos que intervienen en condiciones de comparación de igualdad en la condición conjuntiva y si los índices incluyen apuntadores a registros (en vez de apuntadores a bloques). Se puede usar cada índice para obtener los *apuntadores a los registros* que satisfacen la condición individual. La *intersección* de estos conjuntos de apuntadores produce los apuntadores a los registros que satisfacen la condición conjuntiva, y los registros que servirán para leer directamente dichos registros. Si sólo algunas de las condiciones tienen índices secundarios, se determinará individualmente si los registros obtenidos satisfacen las condiciones restantes.

Siempre que una sola condición especifica la selección —como en OP1, OP2 u OP3—, sólo podemos determinar si existe un camino de acceso basado en el atributo que interviene en la condición. Si así es, se utiliza el método correspondiente a ese camino de acceso; en caso contrario se usa el enfoque por “fuerza bruta” de la búsqueda lineal del método S1. La optimización de consultas para una operación SELECCIONAR se requiere sobre todo para condiciones de selección conjuntivas siempre que *más de uno* de los atributos que intervienen en las condiciones tengan un camino de acceso. El optimizador deberá elegir el camino de acceso que *obtega el mínimo de registros* de la manera más eficiente.

Al elegir entre varias condiciones simples en una condición de selección conjuntiva, es importante tener en cuenta la **selectividad** de cada condición, que se define como la razón entre el número de registros (tuplas) que satisfacen la condición y el número total de registros (tuplas) del archivo (relación). Cuanto menor sea la selectividad, menor será el número de tuplas que la condición seleccionará y más deseable será usar esa condición primero para seleccionar registros. Aunque es posible que no estén disponibles las selectividades exactas de todas las condiciones, a menudo se guardan en el catálogo del SCBD

[†]Un apuntador a registro identifica de manera única un registro y contiene la dirección del registro en el disco; por ello, también se le conoce como **identificador de registro**.

estimaciones de selectividades, mismas que aprovecha el optimizador. Por ejemplo, la selectividad de una condición de igualdad sobre un atributo clave de la relación $r(R)$ es $1/|r(R)|$, donde $|r(R)|$ es el número de tuplas que contiene la relación $r(R)$. La selectividad de una condición de igualdad sobre un atributo con i valores distintos se estima con $(1/|r(R)|)/i$, o sea $1/i$, suponiendo que los registros se distribuyen uniformemente entre los distintos valores. Con esta suposición, $|r(R)|/i$ registros satisfarán una condición de igualdad sobre este atributo.

Comparada con una condición de selección conjuntiva, una **condición disyuntiva** (en la que condiciones simples se unen mediante el conectivo lógico OR en vez de AND) es mucho más difícil de procesar y de optimizar. Por ejemplo, consideremos OP4:

(OP4) $s_{ND=5 \text{ OR SALARIO} > 30000 \text{ OR SEXO} = F}$ (EMPLEADO)

Con una condición así, poco es lo que puede optimizarse, porque los registros que satisfacen la condición de disyunción son la *unión* de los registros que satisfacen las condiciones individuales. Por tanto, si *alguna* de las condiciones no tiene un camino de acceso, estaremos obligados a utilizar el enfoque por fuerza bruta de la búsqueda lineal. Sólo si existe un camino de acceso para *todas y cada una* de las condiciones podremos optimizar la selección obteniendo los registros que satisfacen cada una de las condiciones y aplicando después la operación de unión para eliminar los registros duplicados. Podemos aplicar la unión a los apuntadores a registros, y no a los registros, si existen caminos de acceso apropiados que devuelvan apuntadores a registros para todas las condiciones.

Un SCBD dispone de muchos de los métodos antes mencionados. El optimizador de consultas deberá elegir el más apropiado para ejecutar cada operación de SELECCIONAR en una consulta. La optimización se vale de fórmulas que estiman los costos de cada uno de los métodos de acceso disponibles, como veremos en la sección 16.3. El optimizador elige el método de acceso con el más bajo costo estimado.

16.1.2 Implementación de la operación de REUNIÓN

La operación de REUNIÓN es una de las que más tiempo consumen durante el procesamiento de las consultas. En su mayoría, las operaciones de REUNIÓN utilizadas en las consultas son de los tipos EQUIRREUNIÓN y REUNIÓN NATURAL, por lo que aquí sólo consideraremos estas dos. En el resto del capítulo, con el término **reunión** nos referiremos a una EQUIRREUNIÓN (o REUNIÓN NATURAL). Hay muchas formas posibles de implementar una **reunión bidireccional**, que es una reunión de dos archivos. Las reuniones en las que intervienen más de dos archivos se denominan **reuniones multidireccionales**. El número de formas posibles de ejecutar las reuniones multidireccionales crece con gran rapidez. En esta sección veremos algunas de las técnicas para realizar reuniones bidireccionales. Para ilustrar nuestro análisis, nos referiremos una vez más al esquema relacional de la figura 6.5; en particular, a las relaciones EMPLEADO, DEPARTAMENTO y PROYECTO. Los algoritmos que consideraremos serán para operaciones de reunión de la forma

$$R \bowtie_{A=B} S$$

donde A y B son atributos con dominio compatible de R y S, respectivamente. Los métodos que veremos se pueden extender a formas de reunión más generales. Ilustraremos cuatro de las técnicas más comunes para efectuar este tipo de reuniones con las siguientes operaciones como ejemplos:

(OP6): EMPLEADO $\bowtie_{ND=NUMERO}$ DEPARTAMENTO
(OP7): DEPARTAMENTO $\bowtie_{ASSGTE=NSS}$ EMPLEADO

Métodos para implementar reuniones

- R1. Enfoque de ciclo anidado (**interior-exterior**) (por fuerza bruta): Para cada registro t de R (ciclo exterior), obtener todos los registros s de S (ciclo interior) y probar si los dos registros satisfacen la condición de reunión $t[A] = s[B]$.
- R2. Empleo de una **estructura de acceso para obtener los registros coincidentes**: Si existe un índice (o clave de dispersión) para uno de los dos atributos de reunión —digamos, B de S —, leer los registros t de R , uno por uno, y usar la estructura de acceso para obtener directamente todos los registros coincidentes s de S (los que satisfagan $s[B] = t[A]$).
- R3. **Reunión por ordenamiento-combinación**: Si los registros de R y S están ordenados físicamente según el valor de los atributos de reunión A y B , respectivamente, podemos implementar la reunión de la forma más eficiente posible. Se examinan ambos archivos en orden según los atributos de reunión, buscando los registros que tienen los mismos valores de A y B . En este método, los registros de cada archivo se examinan una sola vez para compararlos con los del otro, a menos que tanto A como B sean atributos no clave, en cuyo caso será preciso modificar ligeramente el método. La figura 16.2(a) muestra un bosquejo del algoritmo de reunión por ordenamiento-combinación. Con $R(i)$ nos referimos al i -ésimo registro de R . Podemos usar una variación de la reunión por ordenamiento-combinación cuando hay índices secundarios para ambos atributos de reunión. Los índices permiten tener acceso a los registros (examinarlos) en orden según los atributos de reunión, pero los registros mismos están dispersos físicamente entre muchos bloques del archivo, por lo que este método puede ser bastante ineficiente si cada acceso a un registro implica un acceso a un bloque de disco.
- R4. **Dispersión-reunión**: Los registros tanto de los archivos R como de S se dispersan al mismo *archivo de dispersión*, mediante la misma *función de dispersión* con los atributos de reunión A de R y B de S como claves de dispersión. Una sola pasada por el archivo con menor número de registros (digamos, R) dispersa sus registros a las cubetas del archivo de dispersión. Una sola pasada por el otro archivo (S) dispersa entonces sus registros a la cubeta apropiada, donde el registro se combina con todos los registros coincidentes de R .

En la práctica, las técnicas de R1 a R4 se implementan teniendo acceso a *bloques de disco completos* de un archivo, y no a registros individuales. Dependiendo de qué tanto espacio de almacenamiento intermedio esté disponible en la memoria, se puede ajustar el número de bloques que se lean del archivo. En el enfoque de ciclo anidado (R1), es importante cuál es el archivo que se elija para el ciclo exterior y cuál para el interior. Para ilustrar esto, consideremos OP6 y supongamos que el archivo DEPARTAMENTO contiene $r_D = 50$ registros almacenados en $b_D = 10$ bloques de disco, y que el archivo EMPLEADO contiene $r_E = 5000$ registros almacenados en $b_E = 2000$ bloques de disco. Supongamos, además, que se dispone de $n_B = 6$ bloques de almacenamiento intermedio (*buffers*) en memoria principal para llevar a cabo la reunión. Nos conviene leer en una sola operación tantos bloques como puedan caber en

la memoria desde el archivo cuyos registros se empleen en el ciclo exterior ($n_b - 1$ bloques) y un bloque a la vez del archivo del ciclo interior. Esto reducirá el número total de accesos a bloque.

Si se usa EMPLEADO para el ciclo exterior, cada bloque de EMPLEADO se leerá una vez, y el archivo DEPARTAMENTO completo (cada uno de sus bloques) se leerá una vez *cada* ocasión que leamos $n_b - 1$ bloques del archivo EMPLEADO. Tenemos:

Número total de bloques del archivo exterior leídos = b_e

Número de veces que se leen $(n_b - 1)$ bloques del archivo exterior = $\lceil b_e / (n_b - 1) \rceil$

Número total de bloques del archivo interior leídos = $b_d * \lceil b_e / (n_b - 1) \rceil$

Así pues, tenemos el siguiente total de accesos a bloque:

$$b_e + (\lceil b_e / (n_b - 1) \rceil * b_d) = 2000 + (\lceil (2000/5) \rceil * 10) = 6000 \text{ accesos a bloque}$$

Por otro lado, si usamos los registros DEPARTAMENTO en el ciclo exterior, por simetría obtenemos el siguiente total de accesos a bloque:

$$b_d + (\lceil b_d / (n_b - 1) \rceil * b_e) = 10 + (\lceil (10/5) \rceil * 2000) = 4010 \text{ accesos a bloque}$$

Además de los costos anteriores, el algoritmo de reunión necesita un *buffer* para tener los registros reunidos del *archivo resultado*. Una vez lleno el *buffer*, se escribe en disco. Con almacenamiento intermedio doble se puede acelerar el algoritmo (véase la Sec. 4.3). Si el archivo resultado de la operación de reunión tiene b_r bloques de disco, como cada bloque se escribe una vez se añadirán b_r accesos a bloque adicionales a las fórmulas anteriores para estimar el costo de la operación de reunión. Lo mismo pasa con las fórmulas que deduciremos más adelante para otros algoritmos de reunión. Como lo demuestra este ejemplo, conviene usar el archivo *con menos bloques* como archivo del ciclo exterior en el método R1, si hay más de dos *buffers* en memoria para llevar a cabo la reunión. Así pues, el *tamaño* de los archivos por reunir afecta directamente el rendimiento de las diversas técnicas de reunión. Otro factor que influye en el rendimiento de una reunión, sobre todo con el método R2, es el porcentaje de registros de un archivo que se reunirán con los registros del otro. Llamamos a esto el **factor de selección de reunión** de un archivo respecto a una condición de equirreunión con otro archivo. Este factor depende de la condición de equirreunión específica entre los dos archivos.

Para ilustrar esto, consideremos la operación OP7, que reúne cada uno de los registros DEPARTAMENTO con el registro EMPLEADO del gerente de dicho departamento. Aquí se espera que cada registro DEPARTAMENTO (50 registros en nuestro ejemplo) se reúna con un solo registro EMPLEADO, pero no se reunirán muchos registros EMPLEADO (4950 de ellos). Supongamos que hay índices secundarios sobre el atributo NSS de empleado y sobre el atributo NSSGTE de DEPARTAMENTO, y que el número de niveles de dichos índices es $x_{NSS} = 4$ y $x_{NSSGTE} = 2$, respectivamente. Tenemos dos opciones para implementar el método R2. La primera lee cada uno de los registros EMPLEADO y luego usa el índice sobre NSSGTE de DEPARTAMENTO para encontrar un registro DEPARTAMENTO coincidente. En este caso, no se encontrará ningún registro coincidente para los empleados que no dirijan un departamento. El número de accesos a bloque en este caso es aproximadamente

$$b_e + (r_e * (x_{NSSGTE} + 1)) = 2000 + (5000 * 3) = 17\ 000 \text{ accesos a bloque}$$

- (a) ordenar las tuplas de R según el atributo A_i ; (* suponer que R tiene n tuplas (registros) *)
 ordenar las tuplas de S según el atributo B_j ; (* suponer que S tiene m tuplas (registros) *)
 sea $i \leftarrow 1, j \leftarrow 1$;
 mientras $(i \leq n) \vee (j \leq m)$
 hacer {
 si $R(i)[A_i] > S(j)[B_j]$
 entonces sea $j \leftarrow j + 1$
 de otro modo si $R(i)[A_i] < S(j)[B_j]$
 entonces sea $i \leftarrow i + 1$
 de otro modo { (* $R(i)[A_i] = S(j)[B_j]$ así que devolvemos una tupla coincidente *)
 enviar la tupla combinada $\langle R(i), S(j) \rangle$ a T ;
 (* devolver otras tuplas que coincidan con $R(i)$, si las hay *)
 sea $i \leftarrow i + 1$;
 }
 mientras $(i \leq m) \vee (R(i)[A_i] = S(j)[B_j])$
 hacer { enviar la tupla combinada $\langle R(i), S(j) \rangle$ a T ;
 sea $j \leftarrow j + 1$
 }
 (* devolver otras tuplas que coincidan con $S(j)$, si las hay *)
 sea $k \leftarrow i + 1$;
 mientras $(k \leq m) \vee (R(k)[A_i] = S(j)[B_j])$
 hacer { enviar la tupla combinada $\langle R(k), S(j) \rangle$ a T ;
 sea $k \leftarrow k + 1$
 }
 sea $i \leftarrow k, j \leftarrow j$
 }
 }
 (* T contiene el resultado de la proyección *antes* de la eliminación de duplicados *)
 si $\langle$ lista de atributos $\rangle$ incluye una clave de R
 entonces $T \leftarrow T'$
 de otro modo { ordenar las tuplas de T' ;
 sea $i \leftarrow 1, j \leftarrow 2$;
 mientras $i \leq n$
 hacer { enviar la tupla $T'[i]$ a T ;
 mientras $T'[i] = T'[j]$ hacer $j \leftarrow j + 1$; (* eliminar duplicados *)
 $j \leftarrow j, j \leftarrow j + 1$
 }
 }
 (* T contiene el resultado de la proyección después de eliminarse los duplicados *)

Figura 16.2 Implementación de las operaciones de REUNIÓN, PROYECTAR, UNIÓN, INTERSECCIÓN y DIFERENCIA DE CONJUNTOS por ordenamiento, donde R tiene n tuplas y S tiene m tuplas.
 (a) Implementación de la operación $T \leftarrow R \bowtie_{A=B} S$.
 (b) Implementación de la operación $T \leftarrow \pi_{\langle$ lista de atributos $\rangle} (R)$.
 (continúa en la página que sigue)

- (c) ordenar las tuplas de R y S usando los mismos atributos de ordenamiento únicos;
 sea $i \leftarrow 1, j \leftarrow 1$;
 mientras ($i \leq n$) y ($j \leq m$)
 hacer { si $R(i) > S(j)$
 entonces { enviar $S(j)$ a T ;
 sea $j \leftarrow j + 1$
 }
 de otro modo si $R(i) < S(j)$
 entonces { enviar $R(i)$ a T ;
 sea $i \leftarrow i + 1$
 }
 }
 si ($i \leq n$) entonces agregar las tuplas $R(i)$ hasta $R(n)$ a T ;
 si ($j \leq m$) entonces agregar las tuplas $S(j)$ hasta $S(m)$ a T ;
 }
- (d) ordenar las tuplas de R y S usando los mismos atributos de ordenamiento únicos;
 sea $i \leftarrow 1, j \leftarrow 1$;
 mientras ($i \leq n$) y ($j \leq m$)
 hacer { si $R(i) > S(j)$
 entonces sea $j \leftarrow j + 1$
 de otro modo si $R(i) < S(j)$
 entonces sea $i \leftarrow i + 1$
 de otro modo { enviar $R(i)$ a T ; (* $R(i) = S(j)$), así que devolvemos la tupla *)
 sea $i \leftarrow i + 1, j \leftarrow j + 1$
 }
 }
- (e) ordenar las tuplas de R y S usando los mismos atributos de ordenamiento únicos;
 sea $i \leftarrow 1, j \leftarrow 1$;
 mientras ($i \leq n$) y ($j \leq m$)
 hacer { si $R(i) > S(j)$
 entonces sea $j \leftarrow j + 1$
 de otro modo si $R(i) < S(j)$
 entonces { enviar $R(i)$ a T ; (* $R(i)$ no tiene $S(j)$ coincidente, así que devolvemos $R(i)$ *)
 sea $i \leftarrow i + 1$
 }
 de otro modo sea $i \leftarrow i + 1, j \leftarrow j + 1$
 }
 si ($i \leq n$) entonces añadir las tuplas $R(i)$ hasta $R(n)$ a T ;

Figura 16.2 (continuación) (c) Implementación de la operación $T \leftarrow R \cup S$.

- (d) Implementación de la operación $T \leftarrow R \cap S$.
 (e) Implementación de la operación $T \leftarrow R - S$.

La segunda opción lee cada uno de los registros DEPARTAMENTO y luego usa el índice sobre NSS de EMPLEADO para encontrar un registro EMPLEADO coincidente. En tal caso, todos los registros DEPARTAMENTO tendrán un registro EMPLEADO coincidente. El número de accesos a bloques es ahora aproximadamente

$$b_D + (r_D * (x_{NSS} + 1)) = 10 + (50 * 5) = 260 \text{ accesos a bloque}$$

La segunda opción es más eficiente porque el factor de selección de reunión de DEPARTAMENTO respecto a la condición de reunión NSS = NSSGTE es 1, en tanto que el factor de selección de EMPLEADO respecto a la misma condición de reunión es (50/5000). Con el método R2, se debe usar el archivo más pequeño o bien el archivo que tenga una coincidencia con cada registro (factor de selección de reunión alto) en el ciclo exterior. En algunos casos puede crearse un índice específicamente para realizar la operación de reunión si es que no existe ya uno.

El método por ordenamiento-combinación R3 es muy eficiente, pues sólo se leen una vez ambos archivos. Así pues, el número de bloques leídos es igual a la suma de la cantidad de bloques en ambos archivos. Con este método, tanto OP6 como OP7 requerirían $b_E + b_D = 2000 + 10 = 2010$ accesos a bloque. Sin embargo, es necesario que ambos archivos estén ordenados según los atributos de reunión. Si uno de ellos, o los dos, no lo está, pueden ordenarse a propósito para efectuar la operación de reunión. Si estimamos el costo de ordenar un archivo externo en $(b \log_2 b)$ accesos a bloque, y si es necesario ordenar ambos archivos, el costo total de una reunión por ordenamiento-combinación se puede estimar en $(b_E + b_D + b_E \log_2 b_E + b_D \log_2 b_D)$.

El método por dispersión-reunión, R4, también es muy eficiente. En este caso sólo se leen una vez ambos archivos, estén o no ordenados, para producir las entradas del archivo de dispersión. Si este archivo puede mantenerse en la memoria principal, la implementación es sencilla. Si es preciso almacenar partes del archivo de dispersión en disco, el método se vuelve menos eficiente; se han propuesto diversas variaciones para elevar la eficiencia en este caso. Aquí examinaremos la técnica conocida como *reunión por dispersión híbrida*, cuya alta eficiencia se ha demostrado.

En el algoritmo de **reunión por dispersión híbrida** se cuenta con varios *buffers* en memoria para contener los registros dispersos; los registros que no caben en la memoria se guardan en bloques de disco. Supongamos que el tamaño de un *buffer* en memoria es de un bloque de disco, que disponemos de N *buffers* y que la función de dispersión empleada es $h(K) = K \bmod M$, de modo que se necesitan M cubetas de dispersión. La reunión por dispersión híbrida se basa en la idea de dispersar primero el *más pequeño* de los dos archivos por reunir a las cubetas. Si el espacio de los *buffers* en memoria es suficiente para contener todas las cubetas, todos los registros del archivo más pequeño estarán en la memoria después de dispersarse en la primera fase del algoritmo. En este caso, la segunda fase dispersará los registros en cada bloque del segundo archivo —uno por uno— y los combinará con los registros coincidentes del primero. Los registros reunidos resultantes se escribirán en el archivo resultado. En nuestro ejemplo, el archivo más pequeño es el de DEPARTAMENTO; por tanto, si el número de *buffers* disponible en memoria $N > b_D$, todo el archivo DEPARTAMENTO se guardará en la memoria principal. En seguida se leerá y colocará en un *buffer* cada bloque de EMPLEADO, y cada uno de sus registros se dispersará a una cubeta reuniéndolo con el registro o registros de DEPARTAMENTO correspondiente(s). Los registros reunidos se almacenarán en un *buffer* y de ahí se enviarán al archivo resultado en disco. El costo en términos de accesos a bloque es, pues, $(b_D + b_E)$ más b_R (el costo de escribir el archivo resultado).

Si el espacio de almacenamiento intermedio en la memoria no es lo bastante grande para contener todas las cubetas para el más pequeño de los archivos por reunir, parte de cada cubeta se deberá almacenar en disco. Este algoritmo divide el espacio de *buffers* entre las cubetas de dispersión de modo que todos los bloques de la *primera cubeta* residan por completo en la memoria principal. Para cada una de las otras cubetas sólo se asigna un *buffer* en memoria, cuyo tamaño es igual al de un bloque de disco; los demás bloques de la cubeta se almacenan en disco. Cada vez que se llena el *buffer* en memoria de una cubeta, su contenido se escribe en un bloque de disco de manera que todos los bloques de disco de una cubeta estén contiguos (o enlazados). Al final de la primera fase, el más pequeño de los archivos por reunir estará disperso en las cubetas; la primera cubeta residirá por completo en la memoria principal, y las demás residirán en disco con excepción del primer bloque de cada cubeta.

En la segunda fase, los registros del segundo archivo por reunir —el de mayor tamaño— se dispersan empleando la misma función de dispersión con el o los atributos de reunión. Si un registro se dispersa a la primera cubeta, se reúne con el registro coincidente y el resultado de la reunión se escribe en el *buffer* del resultado (y más adelante en disco). Si un registro se dispersa a una cubeta *distinta de la primera*, se escribe en disco en cubetas para los registros del segundo archivo. Al concluir la segunda fase, todos los registros que se dispersen a la primera cubeta se habrán reunido; ahora habrá $M - 1$ cubetas en disco que contengan los registros restantes del primer archivo, y otras $M - 1$ cubetas que contienen los registros restantes del segundo archivo. Por tanto, tenemos $M - 1$ fases adicionales, una para cada par de cubetas. Para las cubetas i , primero se copian en almacenamiento intermedio los registros dispersos del primer archivo; luego se leen uno por uno los registros del segundo archivo que se dispersaron a la misma cubeta, se comparan, reúnen y escriben en el archivo resultado.

Podemos aproximar el costo de esta reunión como $3 * (b_d + b_e)$ para nuestro ejemplo, ya que cada registro —con excepción de los que se dispersan a la primera cubeta— se lee una vez, se escribe otra vez en disco en una de las cubetas de dispersión, y luego se lee una segunda vez para efectuar la reunión. El costo real es menor, porque los registros de ambos archivos que se dispersan a la primera cubeta sólo se leen una vez. En el ejercicio 16.16 veremos cómo pueden deducirse las fórmulas para la reunión por dispersión híbrida.

16.1.3 Implementación de la operación PROYECTAR

Una operación PROYECTAR $\pi_{\langle \text{lista de atributos} \rangle}(R)$ es fácil de implementar si $\langle \text{lista de atributos} \rangle$ incluye una clave de la relación R , ya que en este caso el resultado de la operación tendrá el mismo número de tuplas que R , pero sólo con los valores de los atributos de $\langle \text{lista de atributos} \rangle$ en cada tupla. Si $\langle \text{lista de atributos} \rangle$ no incluye una clave de R , será preciso eliminar las tuplas repetidas. Esto suele hacerse ordenando el resultado de la operación y eliminando después las tuplas repetidas, que aparecen juntas después de la ordenación, como en la figura 16.2(b). También se puede usar la dispersión para eliminar los duplicados: al dispersarse cada registro e insertarse en una cubeta del archivo de dispersión, se compara con los que ya están en la cubeta; si es un duplicado, no se inserta.

16.1.4 Implementación de operaciones de conjuntos

Las operaciones de conjuntos —UNIÓN, INTERSECCIÓN, DIFERENCIA DE CONJUNTOS y PRODUCTO CARTESIANO— pueden tener implementaciones costosas. En particular, la operación de

producto cartesiano $R \times S$ es muy costosa, porque su resultado incluye un registro por cada combinación de registros de R y S . Es más, los atributos del resultado incluyen todos los atributos de R y de S . Si R tiene n registros y j atributos y S tiene m registros y k atributos, la relación resultante tendrá $n * m$ registros y $j + k$ atributos. Por tanto, es importante evitar la operación de PRODUCTO CARTESIANO y sustituirla por otras operaciones equivalentes durante la optimización de consultas (véase la Sec. 16.2).

Las otras tres operaciones de conjuntos —UNIÓN, INTERSECCIÓN y DIFERENCIA DE CONJUNTOS— se aplican sólo a relaciones compatibles con la unión, que tienen los mismos atributos. La forma usual de efectuar estas operaciones es ordenar las dos relaciones según los mismos atributos. Después de la ordenación, basta una pasada por cada relación para producir el resultado. Por ejemplo, podemos implementar $R \cup S$ examinando ambos archivos ordenados de manera concurrente, siempre que se encuentre la misma tupla en ambas relaciones, sólo se conservará una. En el caso de $R \cap S$, conservaremos en el resultado sólo las tuplas que aparezcan en ambas relaciones. En la figura 16.2(c) a (e) se bosqueja la implementación de estas operaciones por ordenación y examen. Si la ordenación se efectúa según atributos clave únicos, las operaciones se simplificarán aún más.

También puede usarse la dispersión para implementar UNIÓN, INTERSECCIÓN y DIFERENCIA DE CONJUNTOS dispersando ambos archivos a las mismas cubetas del archivo de dispersión. Por ejemplo, para efectuar $R \cup S$ se dispersan primero los registros de R al archivo de dispersión; luego, al dispersar los registros de S , no se insertan los registros repetidos. Para implementar $R \cap S$, primero se dispersan los registros de R al archivo de dispersión; luego, al dispersar cada uno de los registros de S , si se encuentra un registro idéntico en la cubeta se añade el registro al archivo resultado. Para implementar $R - S$, primero se dispersan los registros de R al archivo de dispersión; luego, al dispersar los registros de S , si se encuentra un registro idéntico en la cubeta, se elimina esa tupla.

16.1.5 Combinación de operaciones para ejecutar consultas

No es recomendable tener una rutina de acceso individual para cada operación del álgebra relacional. Crear un archivo temporal para contener las tuplas del resultado de la operación casi nunca es eficiente. Por lo regular, una consulta especificada en el álgebra relacional consiste en una secuencia de operaciones. Si ejecutamos una sola operación a la vez, deberemos generar tantos archivos temporales como operaciones haya, y muchos de estos archivos servirán como archivos de entrada para operaciones subsecuentes. La generación y almacenamiento de un archivo temporal grande en el disco consume un tiempo apreciable. Con el fin de reducir el número de archivos temporales, es común crear algoritmos para combinaciones de operaciones.

Por ejemplo, en vez de implementarla por separado, una REUNIÓN se puede combinar con dos operaciones SELECCIONAR sobre los archivos de entrada y una operación PROYECTAR final sobre el archivo resultante; todo esto lo efectúa un algoritmo con dos archivos de entrada y uno solo de salida. En vez de crear cuatro archivos temporales, aplicamos el algoritmo directamente y obtenemos un solo archivo de resultado. En la sección 16.2.1 veremos cómo la optimización heurística del álgebra relacional puede agrupar operaciones para su ejecución.

Hoy día es común crear el código dinámicamente para implementar operaciones múltiples. El código que se genera para ejecutar la consulta combina varios algoritmos que corresponden a operaciones individuales. Conforme se producen las tuplas resultado de una

operación, se suministran como entrada a una operación subsecuente. Por ejemplo, si una operación de reunión sigue a dos operaciones de selección sobre relaciones base, las tuplas que resultan de cada selección se alimentan al algoritmo de reunión conforme se van produciendo.

16.2 Empleo de la heurística en la optimización de consultas

En esta sección analizaremos técnicas de optimización que aplican reglas heurísticas para modificar la representación interna de una consulta —que suele estar en forma de estructura de árbol o de grafo— a fin de mejorar su rendimiento esperado durante la ejecución. El analizador sintáctico de una consulta de alto nivel genera la representación interna, que después se optimiza de acuerdo con las reglas heurísticas. A continuación, el optimizador elige rutinas de acceso para ejecutar grupos de operaciones con base en los caminos de acceso disponibles para los archivos.

La principal **regla heurística** es aplicar las operaciones SELECCIONAR y PROYECTAR antes de aplicar la REUNIÓN u otras operaciones binarias. Esto es porque el tamaño del archivo resultante de una operación binaria es una función de los tamaños de los archivos de entrada; en algunos casos, una función multiplicativa. Las operaciones SELECCIONAR y PROYECTAR casi siempre reducen el tamaño de un archivo y nunca lo aumentan; por tanto, conviene aplicarlas antes de una reunión o de cualquier otra operación binaria.

En la sección 16.2.1 presentaremos la notación de árbol de consulta, con la que se representa una expresión del álgebra relacional. Después mostraremos cómo se aplican las reglas de optimización heurísticas para convertir el árbol en un árbol de consulta **equivalente**, que represente una expresión del álgebra relacional cuya ejecución sea más eficiente pero que dé el mismo resultado que la original. Luego, en la sección 16.2.2, presentaremos la notación de grafo de consulta, con la que se representa una expresión del cálculo relacional. Por último, mostraremos cómo se pueden aplicar reglas heurísticas al grafo de consulta para producir una estrategia de ejecución eficiente.

16.2.1 Optimización heurística de árboles de consulta (álgebra relacional)

Un **árbol de consulta** es una estructura de árbol que corresponde a una expresión del álgebra relacional por el hecho de que representa las relaciones de entrada como *nodos hoja* del árbol y las operaciones del álgebra relacional como *nodos internos*. Una **ejecución del árbol de consulta** consiste en la ejecución de una operación de un nodo interno siempre que sus operandos estén disponibles, sustituyendo después ese nodo interno por la relación que resulte de ejecutar la operación. La ejecución termina cuando se ejecuta el nodo raíz y produce la relación resultado.

La figura 16.3(a) muestra un árbol de consulta para la consulta C2 de los capítulos 6 a 8: Para cada proyecto ubicado en 'Santiago', obtener su número, el número del departamento que lo controla, y el apellido, la dirección y la fecha de nacimiento del gerente de ese departamento. Esta consulta se especifica sobre el esquema relacional de la figura 6.5 y corresponde a la siguiente expresión del álgebra relacional (recuérdese que $\bowtie$ significa REUNIÓN):

$$\pi_{\text{NÚMERO}, \text{NÚM}, \text{APELLIDO}, \text{DIRECCIÓN}, \text{FECHAN}}(((\sigma_{\text{LUGARP}=\text{'Santiago'}}(\text{PROYECTO})) \bowtie_{\text{NÚM}=\text{NÚMERO}}(\text{DEPARTAMENTO})) \bowtie_{\text{NSSGTE}=\text{NSS}}(\text{EMPLEADO}))$$

Esto corresponde a la siguiente consulta en SQL:

```
C2:  SELECT  NÚMERO, NÚM, APELLIDO, DIRECCIÓN, FECHAN
      FROM    PROYECTO, DEPARTAMENTO, EMPLEADO
      WHERE   NÚM=NÚMERO AND NSSGTE=NSS AND
             LUGARP='Santiago'
```

En la figura 16.3(a) se representan las tres relaciones PROYECTO, DEPARTAMENTO y EMPLEADO con nodos hoja, en tanto que las operaciones del álgebra relacional de la expresión se representan con nodos internos del árbol. Cuando se ejecuta este árbol de consulta, el nodo numerado con (1) en la figura 16.3(a) se ejecuta antes que el nodo (2) porque el resultado de la operación (1) debe estar disponible para poder ejecutar la operación (2). De manera similar, el nodo (2) debe ejecutarse antes que el nodo (3), y así sucesivamente.

En general, muchas expresiones diferentes del álgebra relacional —y por tanto muchos árboles de consulta distintos— pueden ser equivalentes; es decir, pueden corresponder a la misma consulta. Además, una consulta se puede expresar de varias maneras en un lenguaje de consulta de alto nivel como SQL (véase el Cap. 7). El analizador sintáctico de consultas casi siempre genera un árbol de consulta canónico estándar que corresponde a una consulta en SQL, sin efectuar optimización alguna. Por ejemplo, en el caso de una consulta de **selección-proyección-reunión**, como C2, su árbol canónico se muestra en la figura 16.3(b). Primero se aplica el producto cartesiano de las relaciones especificadas en la cláusula FROM; luego las condiciones de selección y reunión de la cláusula WHERE, seguidas de la proyección sobre los atributos de la cláusula SELECT.

Semejante árbol de consulta canónico representa una expresión del álgebra relacional que es muy ineficiente si se ejecuta directamente, debido a las operaciones de producto cartesiano (X). Por ejemplo, si las relaciones PROYECTO, DEPARTAMENTO y EMPLEADO tuvieran registros de 100, 50 y 150 bytes, y contuvieran 100, 20 y 5000 tuplas, respectivamente, el resultado del producto cartesiano contendría 10 millones de tuplas con un tamaño de registro de 300 bytes cada una. Sin embargo, el árbol de consulta está en una forma estándar simple; toca al optimizador de consultas heurístico transformar este **árbol de consulta inicial** en un **árbol de consulta final** cuya ejecución sea eficiente. El optimizador debe incluir reglas de equivalencia entre expresiones del álgebra relacional que puedan aplicarse al árbol inicial, guiadas por las reglas heurísticas de optimización de consultas, para producir el árbol de consulta final optimizado. En primer término analizaremos de manera informal cómo un árbol de consulta se transforma por medio de la heurística. Después estudiaremos algunas reglas (pautas) de transformación generales y mostraremos cómo se pueden usar en un optimizador algebraico heurístico.

Ejemplo de transformación de una consulta. En relación con la base de datos de la figura 6.5, consideremos una consulta C: Buscar los apellidos de los empleados nacidos después de 1957 que trabajan en un proyecto llamado 'Acuario'. Esta consulta se puede especificar en SQL así:

```
C:  SELECT  APELLIDO
      FROM    EMPLEADO, TRABAJO EN, PROYECTO
      WHERE   NOMBREP='Acuario' AND NÚMEROP=NÚM AND NSS=NSS
             AND FECHAN>'31-DIC-57'
```

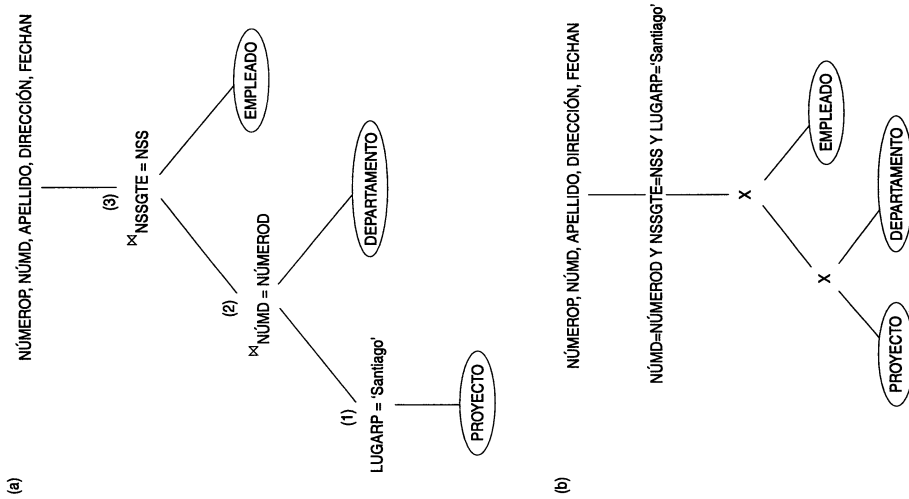


Figura 16.3 Dos árboles de consulta para la consulta C2. (a) Árbol de consulta correspondiente a la expresión del álgebra relacional para C2. (b) Árbol de consulta inicial (canónico) para la consulta C2 en SQL.

El árbol de consulta inicial para C se muestra en la figura 16.4(a). La ejecución directa de este árbol crea primero un archivo muy grande que contiene el producto cartesiano de los archivos EMPLEADO, TRABAJO_EN y PROYECTO completos. Sin embargo, sólo necesitamos un registro de PROYECTO —el del proyecto 'Acuario'— y sólo los registros EMPLEADO en los que la fecha de nacimiento sea posterior a '31-DIC-57'. La figura 16.4(b) ilustra un árbol de consulta mejorado que primero aplica las operaciones SELECCIONAR para reducir el número de tuplas que aparecen en el producto cartesiano.

Se logra una mejora adicional intercambiando las posiciones de las relaciones EMPLEADO y PROYECTO en el árbol, como se aprecia en la figura 16.4(c). Esto aprovecha la información

de que NÚMOP es un atributo clave de la relación PROYECTO, y por ello la operación SELECCIONAR sobre la relación PROYECTO obtendrá un solo registro. Podemos mejorar aún más el árbol de consulta reemplazando toda operación PRODUCTO CARTESIANO seguida de una condición de reunión con una operación REUNIÓN, como en la figura 16.4(d). Otra mejora consiste en conservar en las relaciones intermedias temporales sólo los atributos requeridos por operaciones subsiguientes, incluyendo las operaciones de PROYECTAR (π) lo antes posible en el árbol de consulta, como se aprecia en la figura 16.4(e). Esto reduce el número de atributos (columnas) de las relaciones intermedias temporales, mientras que las operaciones SELECCIONAR sólo reducen el número de tuplas (registros).

Como lo demuestra el ejemplo anterior, podemos convertir un árbol de consulta paso a paso en otro árbol cuya ejecución sea más eficiente. Sin embargo, debemos cerciorarnos de que los pasos de conversión siempre conduzcan a un árbol de consulta equivalente. Para ello, debemos saber cuáles reglas de transformación *conservan esta equivalencia*. A continuación examinaremos algunas de estas reglas de transformación.

Reglas generales de transformación para operaciones del álgebra relacional. Hay muchas reglas para transformar operaciones del álgebra relacional en otras equivalentes. Aquí nos interesa el significado de las operaciones y las relaciones resultantes. Por tanto, si dos relaciones tienen el mismo conjunto de atributos en un orden distinto, pero ambas representan la misma información, las consideraremos equivalentes. En la sección 6.1.2 dimos una definición alternativa de *relación* según la cual el orden de los atributos no es importante; utilizaremos esa definición aquí. Ahora vamos a enunciar algunas reglas de transformación, sin demostrarlas:

1. Cascada de σ : Una condición de selección conjuntiva puede descomponerse en una cascada (secuencia) de operaciones σ individuales:

$$\sigma_{c_1 \text{ AND } c_2 \text{ AND } \dots \text{ AND } c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\dots(\sigma_{c_n}(R))\dots))$$

2. Conmutatividad de σ : La operación σ es conmutativa:

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

3. Cascada de π : En una cascada (secuencia) de operaciones π , podemos ignorar todas menos la última:

$$\pi_{A_1, A_2, \dots, A_n}(\pi_{A_1, A_2, \dots, A_n}(\pi_{A_1, A_2, \dots, A_n}(\dots(\pi_{A_1, A_2, \dots, A_n}(R))\dots))) = \pi_{A_1, A_2, \dots, A_n}(R)$$

4. Conmutación de σ con π : Si en la condición de selección c intervienen sólo los atributos $A_1, \dots, A_n$ de la lista de proyección, se pueden conmutar ambas operaciones:

$$\pi_{A_1, A_2, \dots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, A_2, \dots, A_n}(R))$$

5. Conmutatividad de $\bowtie$ (o de $\times$): La operación $\bowtie$ es conmutativa:

$$R \bowtie_c S \equiv S \bowtie_c R$$

Cabe señalar que, aunque el orden de los atributos pudiera ser distinto en las relaciones que resultan de las dos reuniones, el "significado" es el mismo porque el orden de los atributos no es importante en la definición alternativa de *relación* que usamos aquí. La operación $\times$ es conmutativa en el mismo sentido que lo es la operación $\bowtie$.

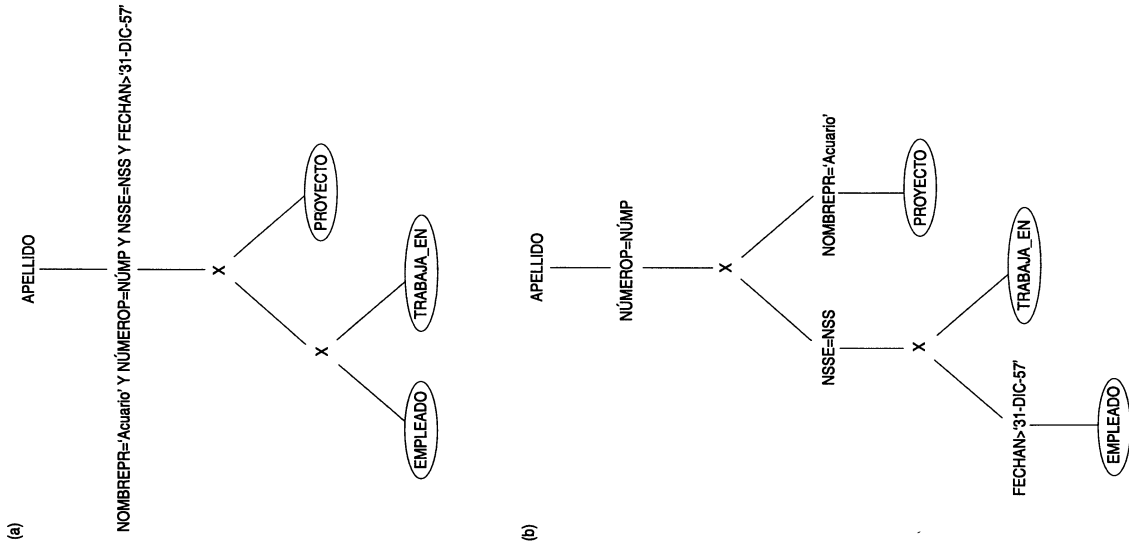


Figura 16.4 Pasos para convertir un árbol de consulta durante la optimización heurística. (a) Árbol de consulta inicial (canónico) para la consulta C en SQL. (b) Desplazamiento de las operaciones SELECCIONAR hacia abajo en el árbol de consulta. (continúa en la siguiente página)

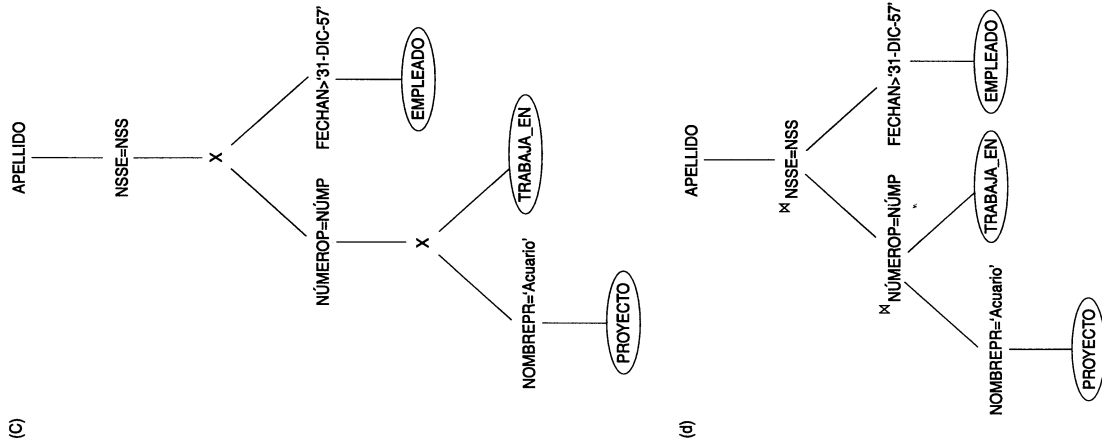


Figura 16.4 (continuación) (c) Aplicación de la operación SELECCIONAR más restrictiva primero. (d) Sustitución de PRODUCTO CARTESIANO y SELECCIONAR por operaciones de REUNIÓN. (continúa en la siguiente página)

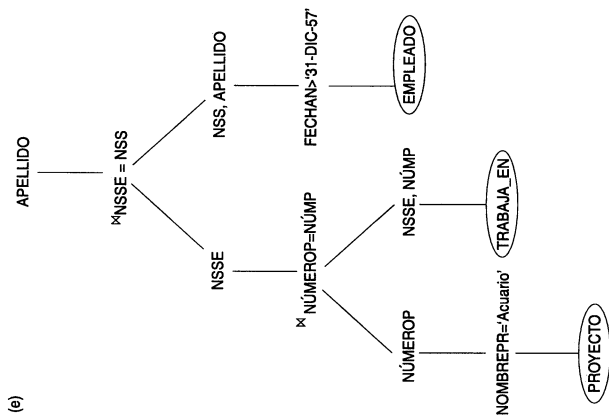


Figura 16.4 (continuación) (e) Desplazamiento de las operaciones PROYECTO y EMPLEADO hacia abajo en el árbol de consulta.

6. Conmutación de σ con $\bowtie$ ($\circ X$): Si todos los atributos de la condición de selección c pertenecen a sólo una de las relaciones por reunir —digamos, R —, las dos operaciones pueden conmutarse como sigue:

$$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$$

O bien, si la condición de selección c puede escribirse como $(c1 \text{ AND } c2)$, y en la condición $c1$ intervienen sólo atributos de R y en la condición $c2$ intervienen sólo atributos de S , las operaciones se conmutan como sigue:

$$\sigma_c(R \bowtie S) \equiv (\sigma_{c1}(R)) \bowtie (\sigma_{c2}(S))$$

Las mismas reglas se aplican si la operación $\bowtie$ se sustituye por X . Estas transformaciones son muy útiles durante la optimización heurística.

7. Conmutación de π con $\bowtie$ ($\circ X$): Supongamos que la lista de proyección es $L = \{A1, \dots, An, B1, \dots, Bm\}$, donde $A1, \dots, An$ son atributos de R y $B1, \dots, Bm$ son atributos de S . Si en la condición de reunión c intervienen sólo atributos comprendidos en L , las dos operaciones pueden conmutarse como sigue:

$$\pi_L(R \bowtie S) \equiv (\pi_{A1, \dots, An}(R)) \bowtie_c (\pi_{B1, \dots, Bm}(S))$$

Si la condición de reunión c contiene atributos adicionales que no están en L , éstos deben añadirse a la lista de proyección, para lo que se requiere una operación π final. Por ejemplo, si los atributos $An+1, \dots, An+k$ de R y $Bm+1, \dots, Bm+p$ de S intervienen en la condición de reunión c , pero no están en la lista de proyección L , las operaciones se conmutan como sigue:

$$\pi_L(R \bowtie_c S) \equiv \pi_L((\pi_{A1, \dots, An, An+1, \dots, An+k}(R)) \bowtie_c (\pi_{B1, \dots, Bm, Bm+1, \dots, Bm+p}(S)))$$

En el caso de X no existe la condición c , así que la primera regla de transformación siempre se aplica, sustituyendo $\bowtie_c$ por X .

8. Conmutatividad de operaciones de conjuntos: Las operaciones de conjuntos $\cup$ y $\cap$ son conmutativas, pero $-$ no lo es.
9. Asociatividad de $\bowtie$, X , $\cup$ e $\cap$: Estas cuatro operaciones son asociativas individualmente; es decir, si θ representa *cualquiera* de estas cuatro operaciones (pero la misma en toda una expresión), tenemos

$$(R \theta S) \theta T \equiv R \theta (S \theta T)$$

10. Conmutación de σ con operaciones de conjuntos: La operación σ se conmuta con $\cup$, $\cap$ y $-$. Si θ representa una *cualquiera* de estas tres operaciones, tenemos

$$\sigma_c(R \theta S) \equiv (\sigma_c(R)) \theta (\sigma_c(S))$$

11. La operación π se conmuta con $\cup$. Si θ representa $\cup$, tenemos

$$\pi_L(R \theta S) \equiv (\pi_L(R)) \theta (\pi_L(S))$$

12. Otras transformaciones: Es posible efectuar otras transformaciones. Por ejemplo, un condición de selección $\circ$ de reunión c se puede convertir en una condición equivalente mediante las siguientes reglas (conocidas como leyes de DeMorgan):

$$c \equiv \text{NOT}(c1 \text{ AND } c2) \equiv (\text{NOT } c1) \text{ OR } (\text{NOT } c2)$$

$$c \equiv \text{NOT}(c1 \text{ OR } c2) \equiv (\text{NOT } c1) \text{ AND } (\text{NOT } c2)$$

Las transformaciones adicionales que vimos en el capítulo 6 no se repetirán aquí.

Estas reglas se resumen en el apéndice E. A continuación veremos cómo se utilizan estas reglas en la optimización heurística.

Bosquejo de un algoritmo de optimización algebraica heurística. Ahora podemos bosquejar los pasos de un algoritmo que utiliza algunas de las reglas anteriores para transformar un árbol de consulta inicial en un árbol optimizado cuya ejecución sea más eficiente (en la mayoría de los casos). Los pasos del algoritmo producirán transformaciones similares a las que vimos en nuestro ejemplo de la figura 16.4. Los pasos del algoritmo son los siguientes:

1. Empleando la regla 1, descomponer cualesquier operaciones SELECCIONAR que tengan condiciones conjuntivas en una cascada de operaciones SELECCIONAR. Esto permite desplazar hacia abajo más libremente las operaciones de selección por las diferentes ramas del árbol.
2. Empleando las reglas 2, 4, 6 y 10 (de conmutatividad de SELECCIONAR con otras operaciones), desplazar cada operación de selección tan abajo en el árbol de consulta como lo permitan los atributos que intervengan en la condición de selección.

3. Empleando la regla 9 (de asociatividad de las operaciones binarias), reacomodar los nodos hoja del árbol de modo que las relaciones de nodo hoja con las operaciones SELECCIONAR más restrictivas se ejecuten primero en la representación del árbol de consulta. Con "operaciones SELECCIONAR más restrictivas" queremos decir las que producen una relación con el menor número de tuplas o el tamaño absoluto más pequeño. Puede usarse cualquiera de las dos definiciones, ya que estas reglas son heurísticas. Otra posibilidad es definir el SELECCIONAR más restrictivo como el que tiene la selectividad más pequeña; esto es más práctico porque las selectividades estimadas a menudo están disponibles en el catálogo.
4. Combinar una operación de PRODUCTO CARTESIANO con una operación SELECCIONAR subsecuente cuya condición represente una condición de reunión, convirtiéndolas en una operación de REUNIÓN.
5. Empleando las reglas 3, 4, 7 y 11 (de cascada de PROYECTAR y de conmutación de PROYECTAR con otras operaciones), descomponer las listas de atributos de proyección y desplazarlas lo más abajo posible en el árbol creando nuevas operaciones PROYECTAR según sea necesario.
6. Identificar subárboles que representen grupos de operaciones que se puedan ejecutar con un solo algoritmo.

En nuestro ejemplo, la figura 16.4(b) muestra el árbol de la figura 16.4(a) después de aplicar los pasos 1 y 2 del algoritmo; la figura 16.4(c) muestra el árbol después de la aplicación del paso 3; la figura 16.4(d) después del paso 4 y la figura 16.4(e) después del paso 5. En el paso 6 podríamos agrupar en un solo algoritmo las operaciones del subárbol cuya raíz es la operación $\bowtie_{\text{NÚMEROP} = \text{NÚMP}}$. También podemos agrupar las operaciones restantes en otro subárbol, donde las tuplas que resultan del primer algoritmo sustituyen al subárbol cuya raíz es la operación $\bowtie_{\text{NÚMEROP} = \text{NÚMP}}$, porque la primera agrupación indica que este subárbol se ejecutará primero.

Resumen de la heurística para la optimización algebraica. Ahora resumiremos la heurística básica para la optimización algebraica. La principal heurística consiste en aplicar primero las operaciones que reducen el tamaño de los resultados intermedios. En esto entra la ejecución de las operaciones SELECCIONAR tan pronto como sea posible para reducir el número de tuplas, y la ejecución de las operaciones PROYECTAR tan pronto como sea posible para reducir el número de atributos. Esto se logra desplazando las operaciones SELECCIONAR y PROYECTAR lo más abajo que se pueda en el árbol. Además, se deben ejecutar las operaciones SELECCIONAR y REUNIÓN más restrictivas —esto es, las que producen relaciones con el menor número de tuplas o el tamaño absoluto más pequeño— antes que otras operaciones similares. Para ello se reacomodan los nodos hoja del árbol y se ajusta el resto del árbol según sea apropiado.

16.2.2 Optimización heurística de grafos de consulta (cálculo relacional) *

El enfoque que describimos en seguida se conoce como **descomposición de consultas** y se propuso inicialmente como técnica de optimización heurística para implementar el lenguaje QUEL (Cap. 8) en el SGSD INGRES. Las consultas en QUEL son muy similares al cálculo

relacional de tuplas (Cap. 8) y se representan como grafos de consulta. Aquí sólo examinaremos consultas de seleccionar-proyectar-reunir, aunque la técnica puede extenderse a consultas más generales. Recuerde que una consulta en QUEL puede incluir **variables de tupla**: una variable por cada relación que desempeñe un cierto papel en la consulta.

Un **grafo de consulta** es una representación de una expresión seleccionar-proyectar-reunir del cálculo relacional de tuplas o consulta QUEL. Cada variable de tupla se representa con un **nodo** en la gráfica, y las **aristas** entre estos nodos representan **condiciones de reunión** en las que intervienen estas variables. Los valores constantes que aparecen en la consulta se representan mediante nodos especiales llamados **nodos constantes**, los cuales se conectan mediante aristas, que representan condiciones de selección, a los nodos que representan las variables de tupla implicadas en dichas condiciones.

La figura 16.5 muestra un grafo de consulta para la siguiente consulta en QUEL, que se especifica sobre el esquema relacional de la figura 6.5 y que corresponde a la consulta C2 de los capítulos 6 a 8: Para cada proyecto ubicado en 'Santiago', obtener su número, el número del departamento que lo controla, y el apellido, la dirección y la fecha de nacimiento del gerente de ese departamento:

**C2: RANGE OF P IS PROYECTO, D IS DEPARTAMENTO, E IS EMPLEADO
 RETRIEVE (P.NÚMEROP, D.NÚMEROD, E.APELLIDO, E.FECHAN,
 E.DIRECCIÓN)
 WHERE P.NÚMD=D.NÚMEROD AND D.NSSGTE=E.NSS AND
 P.LUGARP='Santiago'**

En la figura 16.5 las variables de tupla P, D y E y el valor constante 'Santiago' se representan con nodos en el grafo. Los nodos constantes se indican con óvalos dobles. Las aristas se rotulan con las condiciones de selección o reunión correspondientes. Observe que el grafo de consulta *no* especifica ningún *orden de ejecución* sobre las operaciones, a diferencia del árbol de consulta, que tiene un orden implícito. Esto hace al grafo de consulta una representación totalmente neutral, que especifica *qué* va a obtener la consulta pero *no cómo* ejecutarla. El grafo de consulta es una **representación canónica** de una consulta de seleccionar-proyectar-reunir del cálculo relacional; cada consulta corresponde a uno y sólo un grafo. En esto difieren de los árboles de consulta, ya que muchos árboles distintos pueden representar la misma consulta.

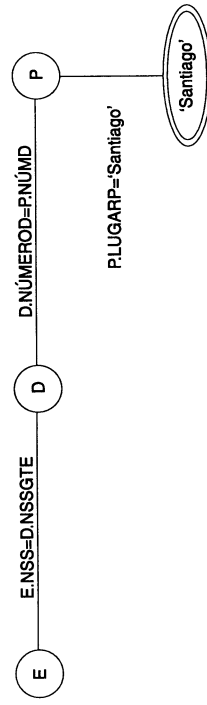


Figura 16.5 Grafo de consulta para C2.

Optimización y ejecución de un grafo de consulta por descomposición de la consulta. Ahora describiremos el método de descomposición de consultas que optimiza heurísticamente (deriva una "buena" estrategia de ejecución) y ejecuta un grafo de consulta. Ilustraremos nuestro análisis con la consulta C': Buscar los apellidos de los empleados nacidos después de 1957 que trabajan en un proyecto ubicado en 'Santiago', controlado por el departamento 4. Esta consulta se refiere a la base de datos de la figura 6.5 y se especifica en QUEL como sigue:

```
C':  RANGE OF  P IS PROYECTO, T IS TRABAJA_EN, E IS EMPLEADO
      RETRIEVE (E.APELLIDO)
      WHERE    P.PLUGARP='Santiago' AND P.NÚMD=4 AND
              P.NÚMEROP=T.NÚMP AND
              T.NSSE=E.NSS AND E.FECHAN>'31-DIC-57'
```

La figura 16.6(a) muestra el grafo de consulta de C'. El enfoque de descomposición de consultas utiliza la heurística de ejecutar las operaciones SELECCIONAR antes que las de REUNIÓN o de PRODUCTO CARTESIANO identificando **subconsultas de una variable**, en las que participa sólo una variable de tupla y una condición de selección. Estas subconsultas se ejecutan primero, y la lista de proyección de cada subconsulta incluye cualesquier atributos que se necesiten en el proceso subsecuente. Una consulta de múltiples variables se descompone en una secuencia de subconsultas de una sola variable por separación y sustitución de tuplas. La **separación** es el proceso de identificar y separar una subconsulta que tenga una sola variable en común con el resto de la consulta. La **sustitución de tuplas** es el proceso de sustituir una tupla a la vez por una de las variables de la consulta; esto es, se obtiene un ejemplar de la variable con *tuplas correspondientes reales* de la base de datos. Esto genera, para una consulta de n variables, varias consultas más simples de $(n - 1)$ variables, una por cada tupla sustituida.

La separación se aplica antes que la sustitución de tuplas, porque genera dos consultas a partir de una. La sustitución de tuplas se aplica sólo cuando la separación no es posible. Si usamos estas dos operaciones de manera repetitiva, podemos llegar a reducir la consulta a varios componentes irreducibles (constantes) cuya combinación produzca el resultado de la consulta. En el caso de C', primero se separan y ejecutan las siguientes subconsultas de una sola variable:

```
Ca:  RETRIEVE INTO P' (P.NÚMEROP)
      WHERE    PLUGARP='Santiago' AND P.NÚMD=4
Cb:  RETRIEVE INTO E' (E.APELLIDO, E.NSS)
      WHERE    E.FECHAN>'31-DIC-57'
```

El grafo de la figura 16.6(a) se ha reducido ahora al grafo de la figura 16.6(b) al haberse eliminado los nodos constantes e incluido nodos para representar las relaciones intermedias P' y E' correspondientes a los resultados de las subconsultas Ca y Cb. Las relaciones intermedias se representan mediante **nodos pequeños**, que aparecen como círculos dobles en la figura 16.6(b), para indicar que el número de tuplas de estas relaciones se ha reducido al aplicar una consulta de seleccionar-proyectar a una relación. Cuando ya *no es posible separar y ejecutar más subconsultas de una sola variable*, y si queda más de un nodo, el grafo representa una o más consultas de múltiples variables. Cada subgrafo conectado representa una subconsulta, y cada arista, una condición de reunión. La operación de pro-

cartesiano se debe aplicar a las subconsultas correspondientes a particiones del grafo. En nuestro ejemplo, el grafo de la figura 16.6(b) tiene sólo una partición y representa una consulta original modificada C'', la cual es una consulta de tres variables que se refiere a la relación TRABAJA_EN (T) y a las relaciones intermedias P' y E':

```
C'':  RETRIEVE (E'.APELLIDO)
      WHERE  P'.NÚMEROP=T.NÚMP AND T.NSSE=E'.NSS
```

El siguiente paso es aplicar la sustitución de tuplas, que especifica un orden en las operaciones de reunión y producto cartesiano con base en el método de ciclo anidado para ejecutar reuniones (método RI de la sección 16.1.2, pero extendido para reuniones multidireccionales). Sin embargo, en la implementación real podría usarse el método R2 si existen estructuras de acceso apropiadas sobre los atributos de reunión. La principal decisión de optimización implica elegir cuál de las relaciones se debe colocar en el ciclo exterior y cuál en el interior. Durante la sustitución de tuplas de esta consulta de n variables, la variable de tupla de la relación exterior se sustituye por una de sus tuplas a la vez durante el proceso de evaluación de la consulta. Para *cada tupla* de la relación externa, se genera una consulta de $(n - 1)$ variables, con la variable de la relación exterior sustituida por esa tupla constante, y podemos aplicar separación seguida de sustitución de tuplas recursivamente a cada una de estas consultas.

La heurística principal para escoger una relación para la sustitución de tuplas es el número de tuplas en la relación. Primero se debe escoger una relación pequeña. Si el grafo reducido tiene más de una relación pequeña, se debe escoger la que se estime que tenga el menor número de tuplas, si es posible determinar esto.[†]

Ilustraremos la sustitución de tuplas para nuestro ejemplo de la figura 16.6(b). A partir de la extensión de la base de datos de la figura 6.6, la relación P' tendrá dos tuplas PROYECTO con valores de NÚMEROP de 10 y 30, y E' tendrá tres tuplas EMPLEADO con valores de NSS de 999887777, 453453453 y 987987987. Si escogemos la relación P' para la sustitución de tuplas, podemos reemplazar el grafo de la figura 16.6(b) por los dos grafos de la figura 16.6(c): un grafo por cada tupla t de P'. Después de obtener el resultado de cada uno de estos grafos, tomaremos su UNIÓN para formar el resultado del grafo original de la figura 16.6(b). Para cada uno de los grafos de la figura 16.6(c), el nodo al que se aplicó la sustitución de tuplas se puede descomponer en uno (o más) nodos constantes, y volvemos a aplicar recursivamente el algoritmo de descomposición a cada grafo. Al final del algoritmo se producirá el resultado de la consulta original. Las figuras 16.6(d) a (f) ilustran el resto del proceso de nuestra consulta. Se ejecuta la UNIÓN de los resultados de cada grafo producido por la sustitución de tuplas.

16.3 Empleo de estimaciones de costo en la optimización de consultas^{*}

Un optimizador de consultas no debe depender exclusivamente de reglas heurísticas; también debe estimar y comparar los costos de ejecutar una consulta con diferentes estrategias

[†]En la práctica, el proceso de escoger la variable para la sustitución de tuplas, llamado *selección de variable*, es más complicado. Se puede utilizar otros factores como la disponibilidad de caminos de acceso a relaciones para estimar los costos de elegir diferentes variables.

de ejecución, y elegir la estrategia con el *más bajo costo estimado*. Para que este enfoque funcione se necesitan estimaciones de costo precisas para cada estrategia de ejecución, a fin de que las comparaciones sean justas y realistas. Además, debemos limitar el número de estrategias de ejecución que se considerarán; de lo contrario, se invertirá demasiado tiempo en elaborar estimaciones de costo para las muchas estrategias de ejecución posibles. Así pues, este enfoque es más apropiado para **consultas compiladas**, en las que la optimización se efectúa en el momento de la compilación y el código de la estrategia resultante se almacena y ejecuta directamente en el momento de la ejecución. En las **consultas interpretadas**, donde todo el proceso se muestra en la figura 16.1 se realiza en el momento de la ejecución, una optimización con todas las de la ley puede hacer muy lenta la respuesta. Una optimización más elaborada es lo indicado para las consultas compiladas, y una optimización parcial, menos larga, es lo mejor para las consultas interpretadas.

Esta estrategia se llama **optimización sistemática de consultas**,[†] y es similar a las técnicas de optimización tradicionales según las cuales se busca la solución a un problema dentro del espacio de soluciones al tiempo que se mantiene minimizada una función objetiva (el costo). Las funciones de costo empleadas en la optimización de consultas son estimaciones y no funciones exactas, por lo que es posible elegir una estrategia de ejecución que no sea la óptima. En la sección 16.3.1 examinaremos los componentes del costo de ejecución de las consultas. En la sección 16.3.2 veremos el tipo de información que se requiere para las funciones de costo. Esta información se mantiene en el catálogo del SCBD. En la sección 16.3.3 proporcionaremos ejemplos de funciones de costo para una operación SELECCIONAR, y en la sección 16.3.4 analizaremos las funciones de costo para operaciones de REUNIÓN bidireccional.

16.3.1 Componentes del costo de ejecución de una consulta

El costo de ejecutar una consulta incluye los siguientes componentes:

1. Costo de acceso a almacenamiento secundario: Éste es el costo de buscar, leer y escribir bloques de datos que residen en almacenamiento secundario, principalmente en disco. El costo de buscar registros de un archivo depende del tipo de estructuras de acceso que tenga ese archivo, como ordenamiento, dispersión e índices primarios y secundarios. Además, el costo de acceso se ve afectado por factores como la colocación de los bloques del archivo (contiguos en el mismo cilindro o desperdigados por todo el disco).
2. Costo de almacenamiento: Éste es el costo de almacenar cualesquier archivos intermedios que genere una estrategia de ejecución de la consulta.
3. Costo de cómputo: Éste es el costo de realizar operaciones en memoria con el almacenamiento intermedio de datos durante la ejecución de la consulta. Estas operaciones incluyen la búsqueda de registros, la ordenación de registros, la combinación de registros para una reunión y la realización de cálculos con los valores de los campos.
4. Costo de comunicación: Éste es el costo de enviar la consulta y sus resultados desde el sitio de la base de datos al sitio de la terminal donde se originó la consulta.

[†]Este enfoque se usó por vez primera en el optimizador del SCBD experimental System R creado en IBM.

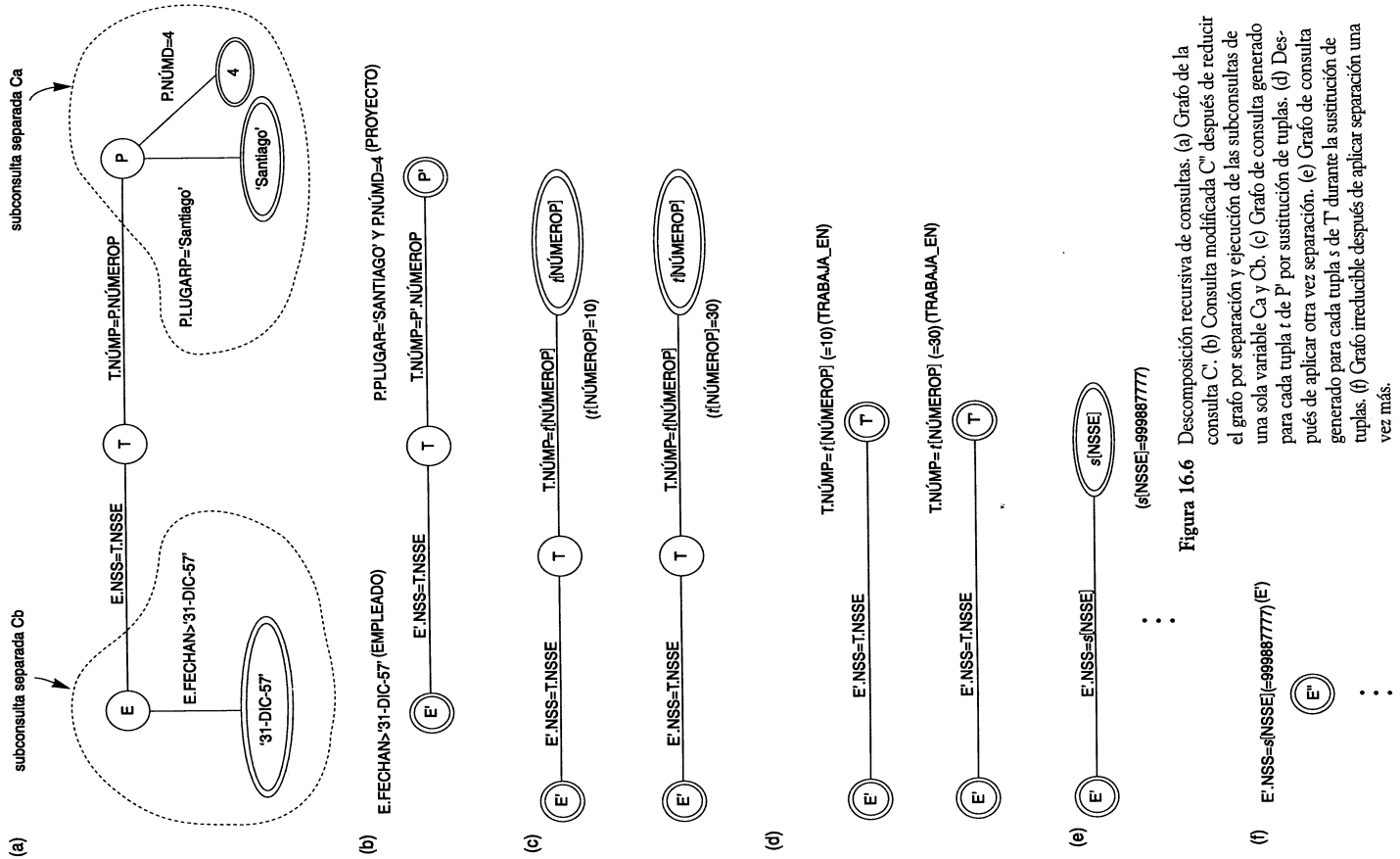


Figura 16.6 Descomposición recursiva de consultas. (a) Grafo de la consulta C. (b) Consulta modificada C' después de reducir el grafo por separación y ejecución de las subconsultas de una sola variable Ca y Cb. (c) Grafo de consulta generado para cada tupla t de P por sustitución de tuplas. (d) Después de aplicar otra vez separación. (e) Grafo de consulta generado para cada tupla s de T durante la sustitución de tuplas. (f) Grafo irreducible después de aplicar separación una vez más.

En las bases de datos grandes, se hace hincapié sobre todo en minimizar el costo de acceso a almacenamiento secundario. De hecho, muchas funciones de costo ignoran los demás factores y comparan las diferentes estrategias de ejecución en términos del número de transferencias de bloques entre el disco y la memoria principal. En las bases de datos más pequeñas, como casi todos los datos de los archivos implicados en la consulta se pueden almacenar totalmente en la memoria, lo más importante debe ser minimizar el costo de cómputo. En las bases de datos distribuidas, donde intervienen muchos sitios (véase el Cap. 23), también debe minimizarse el costo de comunicación. Es difícil incluir todos estos componentes de costo en una función (ponderada) de costo debido a la dificultad de asignar pesos adecuados a dichos componentes. Por esta razón, muchas funciones de costo consideran sólo un factor: el acceso a disco. En la siguiente sección examinaremos parte de la información que se necesita para formular funciones de costo.

16.3.2 Información del catálogo para las funciones de costo

Para estimar los costos de diversas estrategias de ejecución, debemos mantener la información que necesitan las funciones de costo. Esta información puede almacenarse en el catálogo del SGBD, de donde la obtendrá el optimizador de consultas. En primer lugar, debemos conocer el tamaño de cada archivo. En archivos cuyos registros sean todos del mismo tipo, se requerirán el **número de registros (tuplas)**, r , y el **número de bloques**, b , (o estimaciones aproximadas de ellos). El **factor de bloques** del archivo, fb , también puede ser útil. Además, debemos conocer el método de acceso primario y los atributos de acceso primario. Puede ser que los registros del archivo no estén ordenados, o que estén ordenados según un atributo con o sin índice primario o de agrupamiento, o que estén dispersos según un atributo clave. Se mantiene información sobre todos los índices secundarios y atributos de indización. Se requiere el **número de niveles**, x , de cada índice de múltiples niveles (primario, secundario o de agrupamiento) para las funciones de costo que estiman el número de los accesos a bloques que ocurren durante la ejecución de una consulta. En algunas funciones de costo se requiere el **número de bloques del primer nivel del índice**, b_1 .

Otro parámetro importante es el número de valores distintos, d , de un atributo de indización. Esto permite estimar la **cardinalidad de selección**, s , de un atributo, que es el número medio de registros que satisfarán una condición de selección sobre ese atributo. En el caso de un atributo clave, $s = 1$. En el caso de un atributo no clave, si suponemos que los d valores distintos están distribuidos de manera uniforme entre los registros, tenemos que

$$s = (r/d)$$

Es fácil mantener información tal como el número de niveles de un índice porque no cambia con mucha frecuencia; sin embargo, otros tipos de información pueden cambiar muy a menudo. Por ejemplo, el número de registros de un archivo, r , cambia cada vez que se inserta o elimina un registro. El optimizador de consultas necesita valores razonablemente ciertos, pero no por fuerza de último minuto, de estos parámetros para estimar el costo de las diversas estrategias de ejecución. En las siguientes dos secciones examinaremos cómo se usan algunos de estos parámetros en las funciones de costo para un optimizador sistemático de consultas.

16.3.3 Ejemplos de funciones de costo para SELECCIONAR

A continuación presentaremos funciones de costo para los algoritmos de selección S1 a S8 que analizamos en la sección 16.1.1 en términos del *número de transferencias de bloques* entre la memoria y el disco. Estas funciones de costo son estimaciones que no toman en cuenta el tiempo de cálculo, el costo de almacenamiento ni otros factores mencionados en la sección 16.3.1. Nos referiremos al costo del método Si como C_{Si} accesos a bloques.

S1. Enfoque de búsqueda lineal (por fuerza bruta): Examinar todos los bloques del archivo para obtener todos los registros que satisfacen la condición de selección; por tanto, $C_{S1a} = b$. En el caso de una condición de igualdad sobre una clave, sólo se examina en promedio la mitad de los bloques antes de encontrar el registro, así que $C_{S1b} = (b/2)$.

S2. Búsqueda binaria: Esta búsqueda tiene acceso a aproximadamente $C_{S2} = \log_2 b + \lceil (s/fb) \rceil - 1$ bloques de archivo. Esto se reduce a $\log_2 b$ si la condición de igualdad es sobre un atributo único (clave), porque $s = 1$ en este caso.

S3. Empleo de un índice primario (S3a) o una clave de dispersión (S3b) para obtener un solo registro: En el caso de un índice primario, se lee un bloque más que el número de niveles del índice; por tanto, $C_{S3a} = x + 1$. En el caso de la dispersión, la función de costo es aproximadamente $C_{S3b} = 1$.

S4. Empleo de un índice de ordenamiento para leer múltiples registros: Si la condición de comparación es $>$, $\geq$, $<$ o $\leq$ sobre un campo clave con un índice de ordenamiento, aproximadamente la mitad de los registros del archivo satisfarán la condición. Esto implica una función de costo de $C_{S4} = x + (b/2)$. Se trata de una estimación muy aproximada y, aunque en promedio puede ser correcta, podría ser muy inexacta en casos individuales.

S5. Empleo de un índice de agrupamiento para leer múltiples registros: Dada una condición de igualdad, s registros satisfarán la condición, donde s es la cardinalidad de selección del atributo de indización. Esto significa que se leerán $\lceil (s/fb) \rceil$ bloques del archivo, dando $C_{S5} = x + \lceil (s/fb) \rceil$.

S6. Empleo de un índice secundario (árbol B⁺): Si la comparación es de *igualdad*, s registros satisfarán la condición, donde s es la cardinalidad de selección del atributo de indización. Sin embargo, dado que el índice no es de agrupamiento, cada uno de los registros podría residir en un bloque distinto, por lo que la estimación de costo es $C_{S6a} = x + s$. Esto se reduce a $x + 1$ en el caso de un atributo de indización clave. Si la condición de comparación es $>$, $\geq$, $<$ o $\leq$, y se supone que la mitad de los registros del archivo satisfacen la condición, entonces (de manera muy aproximada) se leerá la mitad de los bloques del primer nivel del índice, más la mitad de los registros del archivo a través del índice. La estimación de costo en este caso, muy aproximadamente, es $C_{S6b} = x + (b_1/2) + (r/2)$.

S7. Selección conjuntiva: Se usa S1 o bien uno de los métodos S2 a S6 que acabamos de ver. En el segundo caso, se usa una condición para obtener los registros y luego se verifica en almacenamiento intermedio si cada registro obtenido satisface las demás condiciones de la conjunción.

S8. Selección conjuntiva empleando un índice compuesto: Igual que S3a, S5 o S6a, dependiendo del tipo de índice.

Ejemplo de empleo de las funciones de costo. En un optimizador de consultas es común enumerar las diversas estrategias posibles para ejecutar una consulta y estimar los costos para cada una. Se puede usar una técnica de optimización —la programación dinámica, por ejemplo— para hallar de manera eficiente el costo estimado óptimo (menor), sin tener que considerar todas las posibles estrategias de ejecución. No analizaremos aquí los algoritmos de optimización; más bien, ilustraremos con un ejemplo simple cómo pueden usarse las estimaciones de costos. Supongamos que el archivo EMPLEADO de la figura 6.5 tiene $r_E = 10\ 000$ registros almacenados en $b_E = 2000$ bloques de disco con $fb_E = 5$ registros por bloque y los siguientes caminos de acceso.

1. Un índice de agrupamiento sobre SALARIO, con $x_{\text{SALARIO}} = 3$ y cardinalidad de selección $s_{\text{SALARIO}} = 20$.
2. Un índice secundario sobre el atributo clave NSS, con $x_{\text{NSS}} = 4$ niveles ($s_{\text{NSS}} = 1$).
3. Un índice secundario sobre el atributo no clave ND, con $x_{\text{ND}} = 2$ niveles y $b_{\text{IND}} = 4$ bloques en el primer nivel del índice. Hay $d_{\text{ND}} = 125$ valores distintos de ND, así que la cardinalidad de selección de ND es $s_{\text{ND}} = (r_E/d_{\text{ND}}) = 80$.
4. Un índice secundario sobre SEXO, con $x_{\text{SEXO}} = 1$. Hay $d_{\text{SEXO}} = 2$ valores distintos para el atributo SEXO, así que la cardinalidad de selección es $s_{\text{SEXO}} = (r_E/d_{\text{SEXO}}) = 5000$.

Ilustraremos el empleo de las funciones de costo con los siguientes ejemplos:

- (OP1): $\sigma_{\text{NSS}=129456789}$ (EMPLEADO)
 (OP2): $\sigma_{\text{ND}=5}$ (EMPLEADO)
 (OP3): $\sigma_{\text{ND}=5}$ (EMPLEADO)
 (OP4): $\sigma_{\text{ND}=5 \text{ AND SALARIO} > 30000 \text{ AND SEXO} = F}$ (EMPLEADO)

El costo de la opción por fuerza bruta (búsqueda lineal) S1 se estimará como $C_{\text{S1a}} = b_E = 2000$ (para una selección sobre un atributo no clave) o $C_{\text{S1b}} = (b_E/2) = 1000$ (costo medio para una selección sobre un atributo clave). En el caso de OP1 podemos usar el método S1 o el S6a; la estimación de costo para S6a es $C_{\text{S6a}} = x_{\text{NSS}} + 1 = 4 + 1 = 5$, y se prefiere al método S1, cuyo costo medio es $C_{\text{S1b}} = 1000$. En el caso de OP2 podemos usar el método S1 (con costo estimado $C_{\text{S1a}} = 2000$) o el método S6b (con costo estimado $C_{\text{S6b}} = x_{\text{ND}} + (b_{\text{IND}}/2) + (r_E/2) = 2 + (4/2) + (10\ 000/2) = 5004$), así que escogemos el enfoque por fuerza bruta para OP2. En el caso de OP3 podemos usar el método S1 (con costo estimado $C_{\text{S1a}} = 2000$) o el método S6a (con costo estimado $C_{\text{S6a}} = x_{\text{ND}} + s_{\text{ND}} = 2 + 80 = 82$), así que elegimos el método S6a.

Por último, consideremos OP4, que tiene una condición de selección conjuntiva. Necesitamos estimar el costo de usar cada uno de los tres componentes de la selección para obtener los registros, más el enfoque por fuerza bruta. Este último tiene una estimación de costo de $C_{\text{S1a}} = 2000$. Si usamos la condición (ND = 5) primero, tendremos la estimación de costo $C_{\text{S6a}} = 82$. Si usamos la condición (SALARIO > 30 000) primero, tendremos la estimación de costo $C_{\text{S4}} = x_{\text{SALARIO}} + (b_E/2) = 3 + (2000/2) = 1003$. Si usamos la condición (SEXO = F) primero, tendremos la estimación de costo $C_{\text{S6a}} = x_{\text{SEXO}} + s_{\text{SEXO}} = 1 + 5000 = 5001$. El optimizador escogerá entonces el método S6a con el índice secundario sobre ND porque tiene la estimación de costo más baja. Se usa la condición (ND = 5) para obtener los registros, y se comprueba el resto de la condición conjuntiva (SALARIO > 30 000 AND SEXO = F) para cada uno de los registros seleccionados que están en la memoria.

16.3.4 Ejemplos de funciones de costos para REUNIÓN

A fin de obtener funciones de costo razonablemente exactas para las operaciones de reunión, debemos tener una idea del tamaño (número de tuplas) del archivo que resultará después de la operación de reunión. Esto suele guardarse como la razón entre el tamaño (número de tuplas) del archivo de reunión y el tamaño del archivo de producto cartesiano, si ambas operaciones se aplican a los mismos archivos, y se denomina **selectividad de reunión**, sr . Si denotamos el número de tuplas de una relación R con $|R|$, tenemos

$$sr = |R \bowtie S| / (|R \times S|) = (|R \bowtie S|) / (|R| * |S|)$$

Si no hay condición de reunión c , entonces $sr = 1$ y la reunión equivale al producto cartesiano. Si ningún par de tuplas de las relaciones satisface la condición de reunión, entonces $sr = 0$. En general, $0 \leq sr \leq 1$. Para una reunión en la que la condición c es una comparación de igualdad $R.A = S.B$, tenemos estos dos casos especiales:

1. Si A es una clave de R , entonces $|(R \bowtie_c S)| \leq |S|$, así que $sr \leq (1/|R|)$.
2. Si B es una clave de S , entonces $|(R \bowtie_c S)| \leq |R|$, así que $sr \leq (1/|S|)$.

Tener una estimación de la selectividad de reunión para condiciones de reunión que se presentan con frecuencia permite al optimizador de consultas estimar el tamaño del archivo resultante de la operación de reunión, dados los tamaños de los dos archivos de entrada, mediante la fórmula $|(R \bowtie_c S)| = sr * |R| * |S|$. Ahora podemos presentar algunas muestras de funciones de costo *aproximadas* para estimar el costo de algunos de los algoritmos de reunión que vimos en la sección 16.1.2. Las operaciones de reunión tienen la forma

$$R \bowtie_{A=B} S$$

donde A y B son atributos de dominio compatible de R y S , respectivamente. Supongamos que R tiene b_R bloques y S tiene b_S bloques:

R1. Enfoque de ciclo anidado: Suponga que usamos R para el ciclo exterior; en tal caso tendremos la siguiente función de costo para estimar el número de accesos a bloques con este método, suponiendo *dos buffers* en memoria. Suponemos también que el factor de bloques del archivo resultante es fb_{RG} y que se conoce la selectividad de reunión:

$$C_{\text{R1}} = b_R + (b_R * b_S) + ((sr * |R| * |S|) / fb_{\text{RG}})$$

La parte final de la fórmula es el costo de escribir el archivo resultante en disco. Podemos modificar esta fórmula para tener en cuenta diferentes números de *buffers* en memoria, como se explicó en la sección 16.1.2.

R2. Empleo de una estructura de acceso para obtener los registros coincidentes: Si hay un índice para el atributo de reunión B de S con x_B niveles, podemos obtener todos los registros de R y luego usar el índice para obtener todos los registros coincidentes t de S que satisfagan $t[B] = s[A]$. El costo depende del tipo de índice. En el caso de un índice secundario donde s_B es la cardinalidad de selección[†] del atributo de reunión B de S , tenemos

[†]La cardinalidad de selección se definió como el número medio de registros que satisfacen una condición de igualdad sobre un atributo, o sea que es el número medio de registros que tienen el mismo valor del atributo y que por tanto se reunirán con un solo registro del otro archivo.

$$C_{R2a} = b_R + (|R| * (x_B + s_B)) + ((sr * |R| * |S|) / fbl_{RS})$$

En el caso de un índice de agrupamiento donde s_B es la cardinalidad de selección de B , tenemos

$$C_{R2a} = b_R + (|R| * (x_B + (s_B / fbl_B))) + ((sr * |R| * |S|) / fbl_{RS})$$

En el caso de un índice primario, tenemos

$$C_{R2c} = b_R + (|R| * (x_B + 1)) + ((sr * |R| * |S|) / fbl_{RS})$$

Si hay una clave de dispersión para uno de los dos atributos de reunión —digamos, B de S — tenemos

$$C_{R2d} = b_R + (|R| * h) + ((sr * |R| * |S|) / fbl_{RS})$$

donde $h \geq 1$ es el número medio de accesos a bloque necesarios para obtener un registro, dado su valor de clave de dispersión.

R3. Reunión por ordenamiento-combinación: Si los archivos ya están ordenados sobre el atributo de reunión, la función de costo para este método es

$$C_{R3a} = b_R + b_S + ((sr * |R| * |S|) / fbl_{RS})$$

Si es preciso ordenar los archivos, hay que agregar el costo de la ordenación. Podemos aproximar este costo con $k * b * \log_2 b$ para un archivo de b bloques, donde k es un factor constante. Así pues, tenemos la siguiente función de costo:

$$C_{R3b} = k * ((b_R * \log_2 b_R) + (b_S * \log_2 b_S)) + b_R + b_S + ((sr * |R| * |S|) / fbl_{RS})$$

Ejemplo de uso de las funciones de costo. Suponga que tenemos el archivo EMPLEADO que describimos en el ejemplo de la sección anterior, y que el archivo DEPARTAMENTO de la figura 6.5 consta de $r_D = 125$ registros almacenados en $b_D = 13$ bloques de disco. Consideremos las operaciones de reunión:

(OP6): EMPLEADO^{NÚMERO} DEPARTAMENTO
(OP7): DEPARTAMENTO^{NÚMERO} EMPLEADO

Suponga que tenemos un índice primario sobre NÚMERO de DEPARTAMENTO con $x_{\text{NÚMERO}} = 1$ nivel y un índice secundario según NSSGTE de DEPARTAMENTO con cardinalidad de selección $s_{\text{NSSGTE}} = 1$ y $x_{\text{NSSGTE}} = 2$ niveles. Suponga que la selectividad de reunión para OP6 es $sr_{OP6} = (1/|DEPARTAMENTO|) = 1/125$ porque NÚMERO es una clave de DEPARTAMENTO. Suponga además que el factor de bloques del archivo de reunión resultantes es $fbl_{\text{RD}} = 4$ registros por bloque. Podemos estimar los costos de la operación de REUNIÓN OP6 con los métodos aplicables R1 y R2, como sigue:

1. Usando el método R1 con EMPLEADO como ciclo exterior:

$$C_{R1} = b_e + (b_e * b_D) + ((sr_{OP6} * r_e * r_D) / fbl_{\text{RD}}) = 2000 + (2000 * 13) + (((1/125) * 10\,000 * 125) / 4) = 30\,500$$

2. Usando el método R2 con DEPARTAMENTO como ciclo exterior:

$$C_{R1} = b_D + (b_e * b_D) + ((sr_{OP6} * r_e * r_D) / fbl_{\text{RD}}) = 13 + (2000 * 13) + (((1/125) * 10\,000 * 125) / 4) = 28\,513$$

3. Usando el método R2 con EMPLEADO como ciclo exterior:

$$C_{R2c} = b_e + (r_e * (x_{\text{NÚMERO}} + 1)) + ((sr_{OP6} * r_e * r_D) / fbl_{\text{RD}}) = 2000 + (10\,000 * 2) + (((1/125) * 10\,000 * 125) / 4) = 24\,500$$

4. Usando el método R2 con DEPARTAMENTO como ciclo exterior:

$$C_{R2a} = b_D + (r_D * (x_{ND} + s_{ND})) + ((sr_{OP6} * r_e * r_D) / fbl_{\text{RD}}) = 13 + (125 * (2 + 80)) + (((1/125) * 10\,000 * 125) / 4) = 12\,703$$

El caso 4 tiene la más baja estimación de costo y es el que se escogerá. Observe que si se dispusiera de 14 buffers en memoria (o más) para ejecutar la reunión, en vez de sólo 2, 13 de ellos podrían usarse para contener toda la relación DEPARTAMENTO en la memoria, y el costo del caso 2 se reduciría drásticamente a sólo $b_e + b_D + ((sr_{OP6} * r_e * r_D) / fbl_{\text{RD}}) = 4513$, como se explicó en la sección 16.1.2. Como ejercicio, el lector deberá realizar un análisis similar para OP7.

16.4 Optimización semántica de consultas*

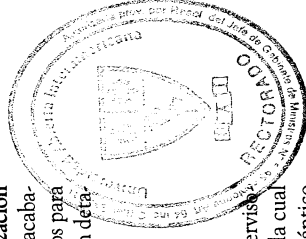
Se ha sugerido un enfoque diferente para la optimización de consultas, llamado **optimización semántica de consultas**. Esta técnica, que puede usarse en combinación con las que acabamos de ver, se vale de restricciones especificadas sobre el esquema de la base de datos para convertir una consulta en otra cuya ejecución sea más eficiente. No analizaremos con detalle este enfoque; sólo lo ilustraremos con un ejemplo. Consideremos la consulta SQL:

```
SELECT  E.APELLIDO, G.APELLIDO
FROM    EMPLEADO E
WHERE   ENSSUPER=G.NSS AND E.SALARIO>G.SALARIO
```

Esta consulta obtiene los apellidos de los empleados que ganan más que sus supervisores. Suponga que tuviéramos una restricción en el esquema de base de datos según la cual ningún empleado puede ganar más que su supervisor directo. Si el optimizador semántico de consultas verifica si existe esta restricción, no necesitará ejecutar la consulta porque sabrá que su resultado estará vacío. Se puede usar las técnicas llamadas de **demonstración de teoremas** para este fin. Esto puede ahorrar bastante tiempo si es posible efectuar la comprobación de restricciones con eficiencia. Sin embargo, una búsqueda entre muchas restricciones para encontrar las que son aplicables a una consulta dada y que la pueden optimizar semánticamente también puede consumir bastante tiempo. Con la aparición de las bases de conocimientos y los sistemas expertos, es posible que las técnicas de optimización semántica se incorporen en los SGBD del futuro.

16.5 Resumen

En este capítulo presentamos un panorama de las técnicas con las que los SGBD procesan y optimizan consultas de alto nivel. En la sección 16.1 vimos cómo un SGBD puede ejecutar diversas operaciones del álgebra relacional. Vimos que algunas operaciones, en particular SELECCIONAR y REUNIÓN, pueden tener muchas opciones de ejecución. También explicamos cómo las rutinas de acceso a la base de datos pueden implementar combinaciones de operaciones.



En la sección 16.2 examinamos enfoques heurísticos para la optimización de consultas, los cuales se valen de reglas heurísticas para mejorar la eficiencia de la ejecución de las consultas. En la sección 16.2.1 mostramos cómo se puede optimizar heurísticamente un árbol de consulta que represente una expresión del álgebra relacional mediante la reorganización de los nodos del árbol. También explicamos las reglas de transformación que conservan la equivalencia y que se pueden aplicar a un árbol de consulta. En la sección 16.2.2 analizamos la notación de un grafo de consulta, el cual representa una expresión del cálculo relacional, y mostramos cómo se puede optimizar y ejecutar con la técnica llamada *descomposición de consultas*. Esta técnica descompone una consulta en otras más simples e incluye heurísticas para ordenar la ejecución de las operaciones de la consulta.

En la sección 16.3 vimos el enfoque sistemático o de estimación de costos para optimizar las consultas. Mostramos la forma de elaborar funciones de costo para algunas rutinas de acceso, y de usar estas funciones para estimar el costo de diferentes estrategias de ejecución. Por último, en la sección 16.4, explicamos la técnica de optimización semántica de consultas.

Preguntas de repaso

- 16.1. ¿Qué significa el término *rutina de acceso*? ¿Por qué es importante implementar varias operaciones relacionales en una sola rutina de acceso?
- 16.2. Analice los diferentes algoritmos para implementar cada uno de los siguientes operadores relacionales y las circunstancias en las que puede usarse cada algoritmo: SELECCIONAR, REUNIÓN, PROYECTAR, UNIÓN, INTERSECCIÓN, DIFERENCIA, PRODUCTO CARTESIANO.
- 16.3. ¿Qué significa el término *optimización heurística*? Analice las principales heurísticas que se aplican durante la optimización de consultas.
- 16.4. ¿Cómo representa un árbol de consulta una operación del álgebra relacional? ¿Qué es una *ejecución* de un árbol de consulta? Explique las reglas para transformar árboles de consulta e indique cuándo debe aplicarse cada regla durante la optimización.
- 16.5. ¿Cómo representan los grafos de consulta operaciones del cálculo relacional? Explique el enfoque de descomposición de consultas para optimizar y ejecutar un grafo de consulta.
- 16.6. ¿Qué significa *optimización sistemática de consultas*?
- 16.7. Analice los componentes de costo de una función que sirve para estimar el costo de ejecución de una consulta. ¿Cuáles componentes de costo se usan con mayor frecuencia como base para las funciones de costo?
- 16.8. Explique los diferentes tipos de parámetros que se usan en las funciones de costo. ¿Dónde se guarda esta información?
- 16.9. Mencione las funciones de costo para los métodos de SELECCIONAR y REUNIÓN que se vieron en la sección 16.1.
- 16.10. ¿Qué significa *optimización semántica de consultas*? ¿En qué difiere de otras técnicas de optimización de consultas?

Ejercicios

- 16.11. Considere las consultas SQL C1, C8, C1B, C4 y C27 del capítulo 7.
 - a. Dibuje por lo menos dos árboles de consulta que puedan representar *cada una* de estas consultas. ¿En qué circunstancias usaría cada uno de sus árboles de consulta?
 - b. Dibuje el árbol de consulta inicial para cada una de estas consultas; en seguida, muestre cómo el algoritmo bosquejado en la sección 16.2.1 optimizaría el árbol de consulta.
 - c. Para cada consulta, compare sus propios árboles de consulta de la parte a y los árboles iniciales y finales de la parte b.
- 16.12. Dibuje el grafo de consulta para las consultas QUEL C0, C1, C3', C8, C1D y C27 del capítulo 8; en seguida, indique cómo el algoritmo de descomposición de consultas de la sección 16.2.2 optimizaría y ejecutaría cada grafo.
- 16.13. Elabore funciones de costo para los algoritmos de PROYECTAR, UNIÓN, INTERSECCIÓN, DIFERENCIA DE CONJUNTOS y PRODUCTO CARTESIANO que vimos en las secciones 16.1.3 y 16.1.4.
- 16.14. Elabore funciones de costo para un algoritmo que conste de dos operaciones SELECCIONAR, una REUNIÓN y una operación PROYECTAR final, en términos de las funciones de costo para las operaciones individuales.
- 16.15. Calcule las funciones de costo para las diferentes opciones de ejecución de la operación de REUNIÓN OP7 que analizamos en la sección 16.3.4.
- 16.16. Elabore fórmulas para el algoritmo de reunión por dispersión híbrida para calcular el tamaño del almacenamiento intermedio para la primera cubeta. Elabore fórmulas de estimación de costo más exactas para el algoritmo.
- 16.17. Estime el costo de las operaciones OP6 y OP7 mediante las fórmulas desarrolladas en el ejercicio 16.16.

Bibliografía selecta

Un estudio reciente (Graefe 1993) analiza el procesamiento de consultas en sistemas de base de datos e incluye una amplia bibliografía. Un artículo de Jarke y Koch (1984) ofrece una taxonomía de la optimización de consultas y contiene una bibliografía de los trabajos en esta área. Una de las primeras referencias sobre optimización de consultas es Rothnie (1975), que describe la optimización de consultas de dos variables en el SCBD experimental DAMAS. Palermo (1974) muestra cómo las expresiones del cálculo relacional pueden convertirse al álgebra relacional y luego optimizarse. Smith y Chang (1975) dan un algoritmo detallado para la optimización en el álgebra relacional. Hali (1976) examina las transformaciones del álgebra relacional en el SCBD experimental PRTV.

Los algoritmos de descomposición para QUEL, empleados en el SCBD experimental INGRES, se presentan en Wong y Youssefi (1976); y en Youssefi y Wong (1979) se estudia su rendimiento. Whang (1985) analiza la optimización de consultas en OBE (Office-By-Example), que es un sistema basado en QBE.

La optimización sistemática con funciones de costo se usó en el SCBD experimental System R y se examina en Astrahan *et al.* (1976). Selinger *et al.* (1979) trata la optimización de reuniones multidireccionales en System R. Los algoritmos de reunión se analizan en Gottlieb (1975), Blasgen y Eswaran (1976) y Whang *et al.* (1982). Los algoritmos de dispersión para implementar las reuniones

se describen y analizan en DeWitt *et al.* (1984), Bratbergsengen (1984), Shapiro (1986), Katsuregawa *et al.* (1989) y Blakeley y Martin (1990), entre otros. Kim (1982) estudia las transformaciones de consultas SQL anidadas en representaciones canónicas. La optimización de funciones agregadas se analiza en Klug (1982) y en Muralikrishna (1992).

Salzberg *et al.* (1990) describe un algoritmo de ordenación externa rápido, y Lipton *et al.* (1990) analiza la estimación de la selectividad. Yao (1979) presenta un análisis comparativo de muchos algoritmos conocidos de procesamiento de consultas. Kim *et al.* (1985) trata temas avanzados de optimización de consultas. La optimización semántica se analiza en King (1981) y en Malley y Zdonick (1986). Trabajos más recientes sobre optimización semántica de consultas se publican en Chakravarthi *et al.* (1990), Shenoy y Ozsoyoglu (1989) y Siegel *et al.* (1992). Aho *et al.* (1979a) analiza una técnica de optimización para consultas del cálculo relacional que sólo incluyen comparaciones de igualdad, operadores AND y cuantificadores existenciales, basada en retablos, que son tablas donde las columnas representan atributos y las filas representan condiciones. Ullman (1982) analiza los retablos. Sagiv y Yannakakis (1980) extienden los retablos para manejar UNIÓN y DIFERENCIA DE CONJUNTOS.

Beck *et al.* (1989) explica la técnica llamada clasificación, que puede servir para optimizar consultas sobre un modelo semántico de los datos. En fechas recientes ha aumentado el interés por los algoritmos paralelos para la ejecución de consultas, como se ve por los trabajos de Mikilineni y Su (1988).

CAPÍTULO 17

Conceptos de procesamiento de transacciones

En este capítulo presentamos el concepto de transacción atómica, con el cual se representa una unidad lógica de procesamiento de base de datos. Analizamos el problema de control de la concurrencia, que ocurre cuando múltiples transacciones introducidas por varios usuarios se interfieren de modo que los resultados obtenidos son incorrectos. También estudiamos la recuperación después de fallar una transacción. La sección 17.1 explica de manera informal por qué son necesarios el control de concurrencia y la recuperación en un sistema de bases de datos. La sección 17.2 presenta el concepto de **transacción atómica** y examina conceptos adicionales relacionados con el procesamiento de transacciones en los sistemas de bases de datos. La sección 17.3 presenta los conceptos de atomicidad, consistencia, aislamiento y durabilidad: las llamadas **propiedades ACID** (por sus siglas en inglés) que se consideran deseables en las transacciones. La sección 17.4 presenta el concepto de **planes** (o **historias**) de transacciones en ejecución y caracteriza la recuperabilidad de los planes. En la sección 17.5 se trata el concepto de la seriabilidad de las ejecuciones de transacciones concurrentes, con el que se pueden definir las secuencias de ejecución correctas para las transacciones concurrentes. En el capítulo 18 veremos las técnicas de control de concurrencia. El capítulo 19 presenta un panorama sobre las técnicas de recuperación.

17.1 Introducción al procesamiento de transacciones

En esta sección presentaremos de manera informal los conceptos de ejecución concurrente de transacciones y de recuperación después de fallar una transacción. En la sección 17.1.1 compararemos los sistemas de bases de datos de uno y de varios usuarios y mostraremos cómo puede darse la ejecución concurrente de transacciones en sistemas multiusuario. La sección

17.1.2 presenta un modelo simple de ejecución de transacciones, basado en operaciones de lectura y escritura de la base de datos, con que se formalizan los conceptos de concurrencia y recuperación. En la sección 17.1.3 mostramos con la ayuda de ejemplos informáticos por qué en los sistemas multiusuario son necesarias las técnicas de control de concurrencia. Por último, en la sección 17.1.4, para analizar las razones por las que se requieren técnicas que hagan posible recuperarse de fallos, presentaremos las diferentes formas en que pueden fallar las transacciones durante su ejecución.

17.1.1 Sistemas de uno y de varios usuarios

Un criterio para clasificar los sistemas de bases de datos es por el número de usuarios que pueden utilizarlos de manera *concurrente*, es decir, al mismo tiempo. Un SGBD es **monousuario** si un solo usuario a la vez puede utilizarlo, y es **multiusuario** si muchos usuarios pueden utilizarlo al mismo tiempo. Los SGBD monousuario suelen estar restringidos a algunos sistemas de microcomputador, y casi todos los demás SGBD son multiusuario. Por ejemplo, el sistema para reservar vuelos en una línea aérea da servicio a cientos de agentes de viajes y empleados de reservación de manera concurrente. Los sistemas de los bancos, de las agencias aseguradoras, de los mercados de valores, etc., también los operan muchos usuarios que introducen al sistema transacciones de manera concurrente.

El que muchos usuarios puedan utilizar los sistemas de computador al mismo tiempo se debe al concepto de **multiprogramación**, que permite al computador procesar simultáneamente varios programas (o transacciones). Si sólo hay una unidad central de procesamiento (UCP), en realidad sólo se puede procesar un programa a la vez. Sin embargo, los **sistemas operativos de multiprogramación** ejecutan algunas órdenes de un programa, luego suspenden ese programa y ejecutan algunas órdenes del siguiente programa, y así sucesivamente. Cuando a un programa le llega el turno de usar la UCP otra vez, se reanuda su ejecución en el punto en el que se suspendió. Así pues, la ejecución concurrente de los programas es en realidad **intercalada**, como se ilustra en la figura 17.1 (idéntica a la figura 4.3), donde se muestra la ejecución concurrente de dos programas A y B de manera intercalada. La intercalación mantiene ocupada a la UCP cuando un programa en ejecución requiere una operación de entrada o de salida (E/S), como leer un bloque de un disco. La UCP se conmuta para ejecutar otro programa en vez de permanecer ociosa durante el tiempo de E/S.

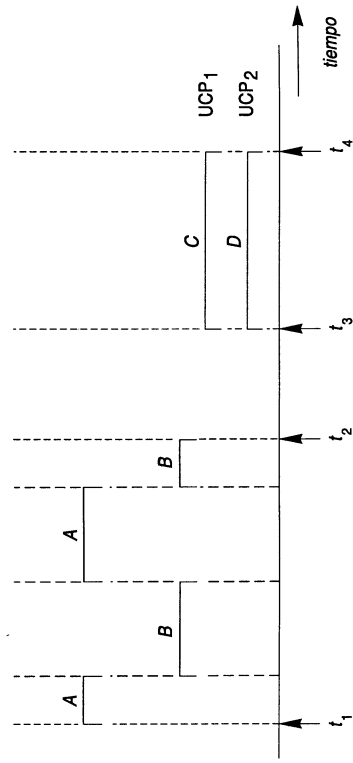


Figura 17.1 Concurrencia intercalada vs. simultánea.

Si el computador cuenta con múltiples procesadores (UCP) en hardware, es posible el procesamiento **simultáneo** de varios programas, dando lugar a una concurrencia simultánea, no intercalada, como lo ilustran los programas C y D de la figura 17.1. Casi toda la teoría sobre el control de concurrencia en las bases de datos se desarrolló en términos de la concurrencia intercalada, aunque se puede adaptar a la concurrencia simultánea. Para el resto de nuestra exposición en este capítulo supondremos el *modelo intercalado de la ejecución concurrente*.

En un SGBD multiusuario, los elementos de información almacenados son los recursos primarios a los que pueden tener acceso concurrente los programas de usuarios, que consistentemente obtienen información de la base de datos y la modifican. A la *ejecución de un programa* que lee o modifica el contenido de la base de datos es a lo que llamamos una **transacción**.

17.1.2 Operaciones de lectura y escritura en una transacción

Al hablar de las técnicas de control de concurrencia y de recuperación nos ocuparemos de las transacciones en el nivel de elementos de información y bloques de discos. En este nivel, las operaciones de acceso a la base de datos que una transacción puede incluir son:

- **leer elemento (X)**: Lee un elemento de la base de datos llamado X y lo coloca en una variable de programa. Para simplificar nuestra notación, supondremos que la *variable de programa también se llama X*.
- **escribir elemento (X)**: Escribe el valor de la variable de programa X en el elemento de la base de datos llamado X.

Como vimos en el capítulo 4, un bloque es la unidad básica de transferencia de datos del disco a la memoria principal del computador. En general, un elemento de información será un campo de algún registro de la base de datos, aunque puede ser una unidad mayor, como un registro o incluso un bloque completo. La ejecución de una orden leer elemento (X) incluye los siguientes pasos:

1. Encontrar la dirección del bloque de disco que contiene el elemento X.
2. Copiar ese bloque de disco en un almacenamiento intermedio (*buffer*) dentro de la memoria principal (si ese bloque no está ya en almacenamiento intermedio).
3. Copiar el elemento X del almacenamiento intermedio a la variable de programa llamada X.

La ejecución de una orden escribir elemento (X) incluye los siguientes pasos:

1. Encontrar la dirección del bloque de disco que contiene el elemento X.
2. Copiar ese bloque de disco en un almacenamiento intermedio dentro de la memoria principal (si ese bloque no está ya en almacenamiento intermedio).
3. Copiar el elemento X desde la variable de programa llamada X al lugar correcto dentro del almacenamiento intermedio.
4. Transferir el bloque actualizado desde el almacenamiento intermedio al disco (ya sea de inmediato o en algún momento posterior).

- (a) leer_elemento(X);
 $X = X + N$;
 escribir_elemento(X);
 leer_elemento(Y);
 $Y = Y + N$;
 escribir_elemento(Y);
- (b) leer_elemento(X);
 $X = X + M$;
 escribir_elemento(X);

Figura 17.2 Dos transacciones simples. (a) La transacción T_1 . (b) La transacción T_2 .

El paso 4 es el que de hecho actualiza la base de datos en disco. En algunos casos el almacenamiento intermedio no se transfiere de inmediato al disco, por si fuera necesario hacer cambios adicionales en el *buffer*. Por lo regular, la decisión sobre cuándo almacenar en disco un bloque modificado que esté en un almacenamiento intermedio corresponde al gestor de recuperación o al sistema operativo.

Para tener acceso a la base de datos, una transacción deberá incluir operaciones leer_elemento y escribir_elemento. La figura 17.2 muestra ejemplos de dos transacciones muy simples. Los mecanismos de control de concurrencia y de recuperación se ocupan principalmente de las órdenes de acceso a la base de datos incluidas en una transacción.

Las transacciones introducidas por los diversos usuarios se podrían ejecutar de manera concurrente y podrían leer y actualizar los mismos elementos de la base de datos. Si esta ejecución concurrente no se controla, puede provocar que la *base de datos* no sea consistente. En la siguiente sección presentaremos de *manera informal* tres de los problemas que pueden presentarse cuando se ejecutan sin control las transacciones concurrentes.

17.1.3 Por qué es necesario el control de concurrencia

Pueden surgir varios problemas cuando las transacciones concurrentes se ejecutan de manera no controlada. Ilustraremos algunos de ellos con referencia a una base de datos simple para hacer reservas en una línea aérea, en la que se almacene un registro por cada vuelo. Cada registro incluye el número de asientos reservados en ese vuelo como *elemento de información con nombre*, entre otra información. La figura 17.2(a) muestra una transacción T_1 que cancela N reservas de un vuelo, cuyo número de asientos reservados está almacenado en el elemento de la base de datos llamado X , y reserva el mismo número de asientos en otro vuelo, cuyo número de asientos reservados está almacenado en el elemento de la base de datos llamado Y . La figura 17.2(b) muestra una transacción más simple, T_2 , que se limita a reservar M asientos en el primer vuelo al que hace referencia la transacción T_1 . Para simplificar nuestro ejemplo, no mostraremos otras partes de las transacciones, como sería la comprobación de si un vuelo tiene suficientes asientos disponibles antes de reservar más asientos.

Cuando se escribe un programa para esta base de datos, tiene como parámetros los números de vuelos, sus fechas y el número de asientos que se reservarán; por tanto, se puede usar el mismo programa para ejecutar muchas transacciones, cada uno con diferentes vuelos y números de asientos por reservar. En lo que al control de concurrencia concierne, una transacción es una *ejecución específica* de un programa sobre una fecha, un vuelo y un número de asientos específicos. En la figura 17.2(a) y (b) las transacciones T_1 y T_2 son *ejecuciones específicas* de los programas que hacen referencia a los vuelos específicos cuyos números de asientos están almacenados en los elementos de información X y Y de la base de datos. Ahora veremos los tipos de problemas a los que podemos enfrentarnos con estas dos transacciones.

El problema de la actualización perdida. Esto ocurre cuando dos transacciones que tienen acceso a los mismos elementos de la base de datos tienen sus operaciones intercaladas de modo que hacen incorrecto el valor de algún elemento. Supongamos que las transacciones T_1 y T_2 se introducen aproximadamente al mismo tiempo, y que el sistema operativo intercala sus operaciones como se muestra en la figura 17.3(a); en tal caso, el valor final del elemento X es incorrecto, porque T_2 lee el valor de X antes de que T_1 lo modifique en la base de datos, con lo que se pierde el *valor actualizado* que resulta de T_1 . Por ejemplo, si $X = 80$ al principio (originalmente había 80 reservas en el vuelo), $N = 5$ (T_1 cancela cinco asientos en el vuelo que corresponde a X y los reserva en el vuelo que corresponde a Y) y $M = 4$ (T_2 reserva cuatro asientos en X), el resultado final deberá ser $X = 79$; sin embargo, en el plan de la figura 17.3(a) es $X = 84$, porque la actualización que canceló cinco asientos se ha perdido.

El problema de la actualización temporal (o lectura sucia). Esto ocurre cuando una transacción actualiza un elemento de la base de datos y luego la transacción falla por alguna razón (véase la Sec. 17.1.4). Otra transacción tiene acceso al elemento actualizado antes de

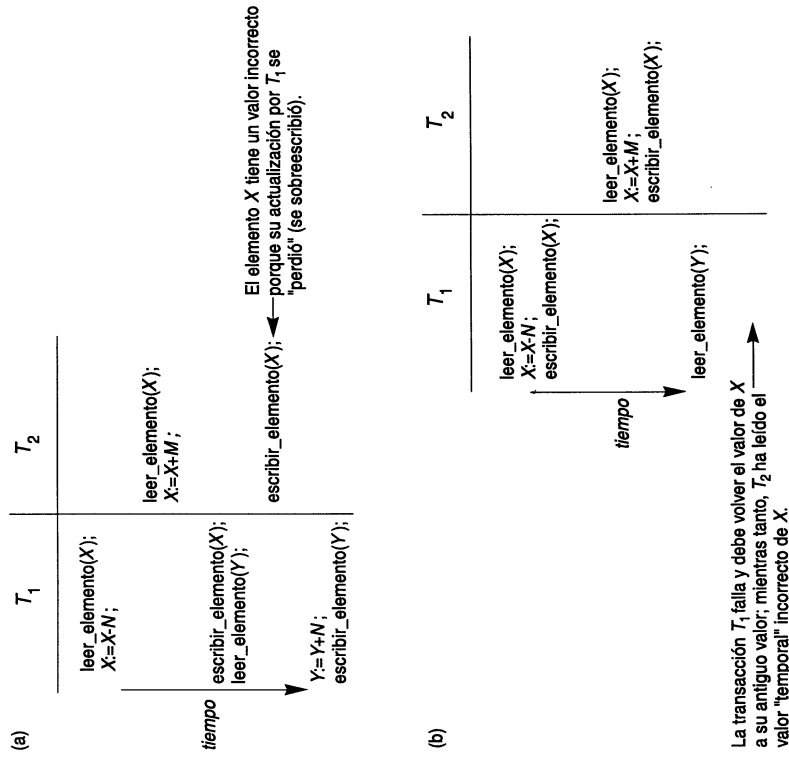


Figura 17.3 Algunos problemas que se presentan cuando la ejecución concurrente no está controlada. (a) El problema de la actualización perdida. (b) El problema de la actualización temporal. (continúa en la siguiente página)

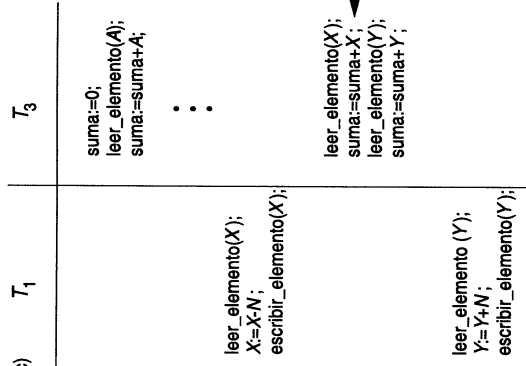


Figura 17.3 (continuación) (c) El problema del resumen incorrecto.

que se restaure a su valor original. La figura 17.3(b) muestra un ejemplo en el que T_1 actualiza el elemento X y después falla antes de completarse, así que el sistema debe cambiar X otra vez a su valor original. Antes de que pueda hacerlo, empero, la transacción T_2 lee el valor "temporal" de X , que no se grabará permanentemente en la base de datos debido al fallo de T_1 . El valor del elemento X que T_2 lee se denomina *dato sucio*, porque fue creado por una transacción que no se completó ni se ha confirmado; por ello, este problema se conoce también como problema de *lectura sucia*.

Problema del resumen incorrecto. Si una transacción está calculando una función agregada de resumen sobre varios registros mientras otras transacciones están actualizando algunos de ellos, puede ser que la función agregada calcule algunos valores antes de que se actualicen y otros después de actualizarse. Por ejemplo, supongamos que una transacción T_1 está calculando el número total de reservas de todos los vuelos; mientras tanto, se está ejecutando la transacción T_1 . Si ocurre la intercalación de operaciones que se muestra en la figura 17.3(c), el resultado de T_1 será erróneo por una cantidad N porque T_1 lee el valor de X después de que se le han restado N asientos, pero lee el valor de Y antes de que se le sumen esos N asientos.

Otro problema que puede presentarse es el de la **lectura no repetible**, en el que una transacción T lee un elemento dos veces y otra transacción T' modifica el elemento entre las dos lecturas. Así, T recibe *diferentes valores* en sus dos lecturas del mismo elemento.

17.1.4 Por qué es necesaria la recuperación

Siempre que se introduce una transacción a un SGBD para ejecutarla, el sistema tiene que asegurarse de que (a) todas las operaciones de la transacción se completen con éxito y su efecto quede asentado permanentemente en la base de datos, o que (b) la transacción no

tenga efecto alguno sobre la base de datos ni sobre cualquier otra transacción. El SGBD no debe permitir que se apliquen a la base de datos algunas operaciones de una transacción T pero no otras operaciones de T . Esto puede suceder si una transacción **falla** después de ejecutar algunas de sus operaciones, pero antes de ejecutarlas todas.

Tipos de fallos. Hay varias razones por las que una transacción puede fallar mientras se está ejecutando:

1. Un fallo del computador (caída): Un error de hardware o software ocurre en el sistema de cómputo durante la ejecución de una transacción. Si el equipo falla, es posible que se pierda el contenido de la memoria interna del computador.
2. Un error de la transacción o del sistema: Alguna operación de una transacción puede hacer que ésta falle, por ejemplo, un desbordamiento de enteros o una división entre cero. También puede haber un fallo de transacción debido a valores erróneos de los parámetros o a un error lógico de programación. Por añadidura, puede suceder que el usuario interrumpa a propósito la transacción durante su ejecución; por ejemplo, al emitir control-C en un entorno VAX/VMS o UNIX.
3. Errores locales o condiciones de excepción detectadas por la transacción: Durante la ejecución de transacciones pueden presentarse ciertas condiciones que requieren la cancelación de la transacción. Por ejemplo, es posible que no se encuentren los datos para la transacción. Una condición, como un saldo insuficiente en una cuenta de una base de datos bancaria, puede hacer que se cancele una transacción, como un retiro de fondos de esa cuenta. Esto puede hacerse por una instrucción ABORTAR programada en la transacción misma.
4. Imposición del control de concurrencia: El método de control de concurrencia puede decidir que se aborte la transacción, para reiniciarla después, porque viola la seriabilidad o porque varias transacciones se encuentran en un estado de bloqueo mortal (véase el Cap. 18).
5. Fallo del disco: Algunos bloques de disco pueden perder sus datos por un mal funcionamiento de lectura o de escritura o por un aterrizaje de una cabeza de lectura/escritura. Esto puede suceder durante una operación de lectura o de escritura de la transacción.
6. Problemas y catástrofes físicos: Esto se refiere a una interminable lista de problemas que incluyen interrupción del suministro de energía, fallo del acondicionamiento de aire, incendio, robo, sabotaje, sobreescritura en discos o cintas por error, y que el operador haya montado la cinta equivocada.

Los fallos 1, 2, 3 y 4 son más comunes que los fallos 5 y 6. Siempre que ocurre un fallo del tipo 1 al 4, el sistema debe mantener suficiente información para recuperarse del fallo. Los fallos de disco o los catastróficos (tipos 5 o 6) no ocurren con tanta frecuencia; si se dan, la recuperación es una tarea bastante complicada. En el capítulo 19 analizaremos conceptos y técnicas de recuperación de fallos.

El concepto de transacción atómica es fundamental para muchas técnicas de control de concurrencia y de recuperación de fallos. En la siguiente sección presentaremos el concepto de transacción y veremos por qué es importante.

17.2 Conceptos de transacciones y sistemas

Como se dijo antes, la ejecución de un programa que incluye operaciones de acceso a la base de datos se denomina **transacción de base de datos** o simplemente **transacción**. Si las operaciones de base de datos en una transacción no actualizan datos, sino que sólo los leen, se habla de una **transacción sólo de lectura**. En este capítulo nuestro interés se centra en las transacciones que sí actualizan la base de datos, por lo que la palabra *transacción* se referirá a una ejecución de programa que *actualiza la base de datos*, a menos que explícitamente se diga lo contrario.

17.2.1 Estados de las transacciones y operaciones adicionales

Una transacción es una unidad atómica de trabajo que se realiza por completo o bien no se efectúa en absoluto. Para fines de recuperación, el sistema necesita mantenerse al tanto de cuándo la transacción se inicia, termina y se confirma o aborta (véase más adelante). Así pues, el gestor de recuperación se mantiene al tanto de las siguientes operaciones:

- **INICIO_DE_TRANSACCION:** Ésta marca el principio de la ejecución de la transacción.
- **LEER o ESCRIBIR:** Éstas especifican operaciones de lectura o escritura de elementos de la base de datos que se efectúan como parte de una transacción.
- **FIN_DE_TRANSACCION:** Ésta especifica que las operaciones de LEER y ESCRIBIR de la transacción han terminado y marca el límite de la ejecución de la transacción. Sin embargo, en este punto puede ser necesario verificar si los cambios introducidos por la transacción se pueden aplicar permanentemente a la base de datos (confirmar) o si la transacción debe abortarse porque viola el control de concurrencia o por alguna otra razón.
- **CONFIRMAR_TRANSACCION:** Ésta señala que la transacción terminó con éxito y que cualesquier cambios (actualizaciones) ejecutados por ella se pueden **confirmar** sin peligro en la base de datos y que no se cancelarán.
- **REVERTIR (o ABORTAR):** Ésta indica que la transacción terminó sin éxito y que cualesquier cambios o efectos que pueda haber aplicado a la base de datos se deben *cancelar*.

Además de las anteriores, algunas técnicas de recuperación requieren operaciones adicionales como las siguientes:

- **DESACER:** Similar a revertir, pero se aplica a una sola operación y no a una transacción completa.
- **REHACER:** Ésta especifica que ciertas operaciones de transacción se deben *repetir (rehacer)* para asegurar que todas las operaciones de una transacción confirmada se hayan aplicado con éxito a la base de datos.

La figura 17.4 muestra un diagrama de transición de estados que describe el paso de una transacción por sus estados de ejecución. Una transacción entra en un estado **activo** inmediatamente después de que inicia su ejecución, y ahí puede emitir operaciones LEER y ESCRIBIR. Cuando la transacción termina, pasa al estado **parcialmente confirmado**. En este punto,

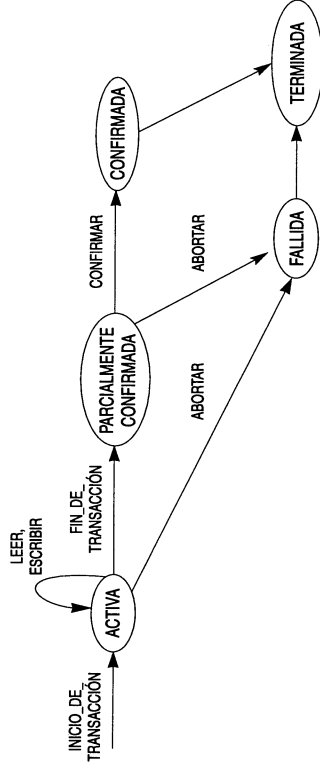


Figura 17.4 Diagrama de transición de estados para la ejecución de transacciones.

algunas técnicas de control de concurrencia requieren que se efectúen ciertas verificaciones para garantizar que la transacción no interfiera otras transacciones en ejecución. Por añadidura, algunos protocolos de recuperación deben asegurar que un fallo del sistema no provocará una incapacidad para registrar permanentemente los cambios hechos por la transacción (usualmente asentando los cambios en una bitácora del sistema, de la cual hablaremos en la siguiente subsección). Una vez realizadas con éxito ambas verificaciones, se dice que la transacción ha llegado a su punto de confirmación y pasa al **estado confirmado**. Los puntos de confirmación se tratarán con mayor detalle en la sección 17.2.3. Una vez que una transacción está en el estado confirmado, ha concluido con éxito su ejecución.

Sin embargo, una transacción puede pasar al estado **fallo** si una de las verificaciones falla o si la transacción se aborta mientras está en el estado activo. En tal caso, es posible que la transacción deba revertirse para anular los efectos de sus operaciones ESCRIBIR sobre la base de datos. El estado **terminado** corresponde al abandono del sistema por parte de la transacción. Las transacciones fallidas o abortadas pueden reiniciarse posteriormente, ya sea en forma automática o después de reintroducirse como transacciones nuevas.

17.2.2 La bitácora del sistema

Para poderse recuperar de los fallos de transacciones, el sistema mantiene una **bitácora** (a veces llamada **diario**) que sigue la pista a todas las operaciones de transacciones que afectan los valores de elementos de la base de datos. La bitácora se mantiene en disco, de modo que no la afecta ningún tipo de fallo, más que los de disco o los catastróficos. Además, la bitácora se respalda periódicamente en cinta para protegerse contra fallos catastróficos. A continuación mencionaremos los tipos de entradas que se escriben en la bitácora y la acción que realiza cada una de ellas. En estas entradas, *T* se refiere a un **identificador de transacción** único que el sistema genera automáticamente y que sirve para identificar cada transacción:

1. **[inicio_de_transacción, T]:** Asienta que se ha iniciado la ejecución de la transacción *T*.
2. **[escribir_elemento, T, X, valor_anterior, nuevo_valor]:** Asienta que la transacción *T* cambió el valor del elemento de base de datos *X* de valor *valor_anterior* a *nuevo_valor*.
3. **[leer_elemento, T, X]:** Asienta que la transacción *T* leyó el valor del elemento de base de datos *X*.

4. [confirmar,T]: Asienta que la transacción *T* terminó con éxito y establece que su efecto se puede confirmar (asentar permanentemente) en la base de datos.
5. [abortar,T]: Asienta que se abortó la transacción *T*.

Como veremos en el capítulo 19, los protocolos de recuperación que evitan las reversiones en cascada no requieren que las operaciones *LEER* se asienten en la bitácora del sistema, pero otros protocolos sí necesitan estas entradas para la recuperación. En el primer caso, el costo extra de asentar las operaciones en la bitácora se reduce, porque en ella se registran menos operaciones —sólo las de *ESCRIBIR*—. Además, los protocolos estrictos requieren entradas *ESCRIBIR* más simples que no incluyen nuevo_valor (véase la Sec. 17.4).

Cabe señalar que aquí estamos suponiendo que las transacciones no se pueden anidar. También suponemos que todos los cambios permanentes de la base de datos ocurren dentro de las transacciones, así que la noción de recuperarse de una transacción equivale a deshacer o rehacer las operaciones recuperables individualmente a partir de la bitácora. Si el sistema se cae, podemos recuperar la base de datos a un estado consistente examinando la bitácora y usando una de las técnicas descritas en el capítulo 19. Dado que la bitácora contiene un registro de cada operación *ESCRIBIR* que altera el valor de algún elemento de la base de datos, es posible *deshacer* (cancelar) el efecto de estas operaciones *ESCRIBIR* de una transacción *T* rastreando hacia atrás en la bitácora y restableciendo todos los elementos alterados con una operación *ESCRIBIR* de *T* a su valor anterior. También podemos *rehacer* el efecto de las operaciones *ESCRIBIR* de una transacción *T* rastreando hacia adelante en la bitácora y cambiando todos los elementos modificados por una operación *ESCRIBIR* de *T* a su nuevo valor. Puede ser necesario rehacer las operaciones de una transacción si todas sus actualizaciones están asentadas en la bitácora pero ocurre un fallo antes de que estemos seguros de que todos los nuevos valores se han escrito permanentemente en la base de datos real.

17.2.3 Punto de confirmación de una transacción

Una transacción *T* llega a su **punto de confirmación** cuando todas sus operaciones que tienen acceso a la base de datos se han ejecutado con éxito y el efecto de todas estas operaciones se ha asentado en la bitácora. Más allá del punto de confirmación, se dice que la transacción está **confirmada**, y se supone que su efecto se *asentó permanentemente* en la base de datos. En ese momento la transacción escribe una entrada [confirmar,T] en la bitácora. Si ocurre un fallo del sistema, buscamos en la bitácora todas las transacciones *T* que han escrito una entrada [inicio_de_transacción,T] en la bitácora pero no han escrito todavía su entrada [confirmar,T]; es posible que durante el proceso de recuperación estas transacciones tengan que *revertirse* para cancelar su efecto sobre la base de datos. Las transacciones que ya escribieron su entrada de confirmación en la bitácora también deberán haber asentado en ella todas sus operaciones *ESCRIBIR*, pues de lo contrario no estarían confirmadas; su efecto sobre la base de datos puede *rehacerse* a partir de las entradas de la bitácora.

Observe que el archivo de la bitácora debe mantenerse en disco. Como se explicó en el capítulo 4, la actualización de un archivo en disco implica copiar el bloque apropiado del archivo del disco a un almacenamiento intermedio en la memoria principal, actualizar este almacenamiento intermedio y copiarlo de la memoria principal al disco. Con frecuencia se mantiene un bloque del archivo de bitácora en la memoria principal hasta que se llena de entradas y luego se escribe una sola vez en el disco, en vez de escribirlo cada vez que se añade una entrada. Esto ahorra el gasto extra de escribir varias veces en disco el mismo bloque de

archivo. Cuando se cae el sistema, sólo las entradas de bitácora que se han *escrito en disco* se considera que están en el proceso de recuperación, porque puede haberse perdido el contenido de la memoria principal. Por ello, *antes* de que una transacción llegue a su punto de confirmación, cualquier porción de la bitácora que no se haya escrito en el disco todavía se deberá grabar. Este proceso se denomina **forzar la escritura** del archivo de bitácora antes de confirmar una transacción.

17.2.4 Puntos de control en la bitácora del sistema

Otro tipo de entradas en la bitácora es el llamado **punto de control**.[†] Un registro [punto_de_control] se escribe en la bitácora periódicamente cada vez que el sistema escribe en la base de datos en disco el efecto de todas las operaciones *ESCRIBIR* de las transacciones confirmadas. Así pues, todas las transacciones que tengan sus entradas [confirmar,T] en la bitácora antes de una entrada [punto_de_control] no necesitarán la *repetición* de sus operaciones *ESCRIBIR* en caso de una caída del sistema.

El gestor de recuperación de un SGBD debe decidir con qué intervalos asentar un punto de control. El intervalo puede medirse en el tiempo —digamos, cada *m* minutos— o por el número *t* de transacciones confirmadas desde el último punto de control, donde los valores de *m* o de *t* son parámetros del sistema. El asentamiento de un punto de control consiste en las siguientes acciones:

1. Suspender temporalmente la ejecución de las transacciones.
2. Forzar la escritura de todas las operaciones de actualización pertenecientes a transacciones confirmadas del almacenamiento intermedio al disco.
3. Escribir un registro [punto_de_control] en la bitácora y forzar la escritura de la bitácora en el disco.
4. Reanudar la ejecución de transacciones.

Un registro de punto de control en la bitácora también puede incluir información adicional, como una lista de identificadores de las transacciones activas o las ubicaciones (direcciones) del primer registro y del más reciente (último) en la bitácora para cada transacción activa. Esto puede facilitar la anulación de operaciones en caso de ser necesaria la reversión de una transacción.

17.3 Propiedades deseables en las transacciones

Las transacciones atómicas deben poseer varias propiedades. Éstas se conocen como **propiedades ACID** (por sus iniciales en inglés) y deben ser impuestas por los métodos de control de concurrencia y de recuperación del SGBD. Las propiedades ACID son:

1. **Atomicidad:** Una transacción es una unidad atómica de procesamiento; o bien se realiza por completo o no se realiza en absoluto.
2. **Conservación de la consistencia:** Una ejecución correcta de la transacción debe llevar a la base de datos de un estado consistente a otro.

[†]El término *punto de control* se ha usado para describir situaciones más restrictivas en algunos sistemas, como DB2. En la literatura también se ha empleado para describir conceptos totalmente distintos.

3. **Aislamiento:** Una transacción no debe dejar que otras transacciones puedan ver sus actualizaciones antes de que sea confirmada; esta propiedad, cuando se hace cumplir estrictamente, resuelve el problema de la actualización temporal y hace innecesarias las reversiones en cascada de las transacciones (véase el Cap. 19).

4. **Durabilidad o permanencia:** Una vez que una transacción ha modificado la base de datos y las modificaciones se han confirmado, éstas nunca deben perderse por un fallo subsecuente.

La propiedad de atomicidad requiere que ejecutemos una transacción hasta completarla. Es obligación del método de recuperación garantizar la atomicidad. Si por alguna razón una transacción no puede completarse, como por una caída del sistema a la mitad de su ejecución, el método de recuperación debe cancelar todos los efectos de la transacción sobre la base de datos.

En general, se considera que la propiedad de conservación de la consistencia es responsabilidad de los programadores que escriben los programas de base de datos o del módulo del SGBD que impone las restricciones de integridad. Recuerde que un **estado de la base de datos** es una colección de todos los elementos de información (valores) almacenados en la base de datos en un momento dado. Un **estado consistente** de la base de datos satisface las restricciones especificadas en el esquema, además de cualesquier otras restricciones que deban cumplirse en la base de datos. Los programas de base de datos deben elaborarse a modo de garantizar que, si la base de datos está en un estado consistente antes de ejecutar la transacción, estará en un estado consistente después de la ejecución *completa* de la transacción, suponiendo que *no hay interferencias con otras transacciones*.

El aislamiento lo impone el método de control de concurrencia. Se ha intentado definir el *nivel o grado de aislamiento* de una transacción. Se dice que una transacción tiene aislamiento de grado 0 (cero) si no sobrescribe las lecturas sucias de transacciones de nivel más alto; una transacción con nivel de aislamiento de grado 1 (uno) no tiene actualizaciones perdidas, y el aislamiento grado 2 no tiene actualizaciones perdidas ni lecturas sucias. Por último, el aislamiento grado 3 (también llamado *aislamiento verdadero*) tiene, además de las propiedades del grado 2, lecturas repetibles. La propiedad de durabilidad es responsabilidad del método de recuperación.

17.4 Planes y recuperabilidad

Cuando varias transacciones se ejecutan de manera concurrente e intercalada, el orden de ejecución de sus operaciones constituye lo que se conoce como **plan** (o **historia**) de las transacciones. En esta sección definiremos primero el concepto de plan, y después caracterizaremos los tipos de planes que facilitan la recuperación cuando ocurren fallos. En la sección 17.5 caracterizaremos los planes en términos de la interferencia provocada por las transacciones participantes, lo que conduce a los conceptos de seriabilidad y de planes serializables.

17.4.1 Planes (historias) de transacciones

Un **plan** (o **historia**) P de n transacciones $T_1, T_2, \dots, T_n$ es un ordenamiento para las operaciones de las transacciones sujeto a la restricción de que, para cada transacción T_i que participe en P , las operaciones de T_i en P deben aparecer en el mismo orden en que ocurren

en T_i . Cabe señalar, empero, que las operaciones de otras transacciones T_j se pueden intercalar con las operaciones de T_i en P . Por ahora, consideraremos que el orden de las operaciones en P es un *ordenamiento total*, aunque es posible manejar planes cuyas operaciones formen *órdenes parciales* (como veremos más adelante).

La figura 17.3(a) y (b) muestra dos posibles planes para las transacciones T_1 y T_2 . Para fines de recuperación y control de concurrencia, nos interesan principalmente las operaciones *leer_elemento* y *escribir_elemento* de las transacciones, así como las operaciones de *confirmar* y *abortar*. Una notación abreviada para describir un plan usa los símbolos l_i, e_i, c_i y a_i para las operaciones *leer_elemento*, *escribir_elemento*, *confirmar* y *abortar*, respectivamente, añadiendo como subíndice el identificador de la transacción (número de la transacción) a cada operación del plan. En esta notación, el elemento de la base de datos que se lee o escribe, X , sigue a las operaciones l_i y e_i entre paréntesis. Por ejemplo, el plan de la figura 17.3(a), que llamaremos P_a , se puede escribir como sigue después de añadirle las operaciones de confirmación:

$$P_a: l_1(X); l_1(X); e_1(X); l_1(Y); e_2(X); c_2; e_1(Y); c_1;$$

De manera similar, el plan de la figura 17.3(b), al que llamaremos P_b , se puede escribir como sigue, suponiendo que la transacción T_1 abortó después de su operación *leer_elemento*(Y):

$$P_b: l_1(X); e_1(X); l_2(X); e_2(X); c_2; l_1(Y); a_1;$$

Se dice que dos operaciones de un plan **están en conflicto** si pertenecen a diferentes transacciones, si tienen acceso al mismo elemento X y si una de las dos operaciones es *escribir_elemento*(X). Por ejemplo, en el plan P_a , las operaciones $l_1(X)$ y $e_2(X)$ están en conflicto, lo mismo que $l_2(X)$ y $e_1(X)$, y que $e_1(X)$ y $e_2(X)$. Sin embargo, las operaciones $l_1(X)$ y $l_2(X)$ no están en conflicto porque ambas son operaciones de lectura; y las operaciones $e_2(X)$ y $e_1(Y)$ tampoco están en conflicto, porque operan sobre diferentes elementos de información, X y Y .

Se dice que un plan P de n transacciones $T_1, T_2, \dots, T_n$ es un **plan completo** si se cumplen las siguientes condiciones:

1. Las operaciones de P son exactamente las operaciones de $T_1, T_2, \dots, T_n$ incluidas una operación de confirmar o de abortar como última operación de cada transacción en el plan.
2. Para cualquier par de operaciones de la misma transacción T_i , su orden de aparición en P es el mismo que su orden de aparición en T_i .
3. Para cualesquier dos operaciones en conflicto, una de ellas debe ocurrir antes que la otra en el plan.

Las condiciones anteriores permiten que dos operaciones *que no estén en conflicto* ocurran en el plan sin definir cuál lo haga primero, dando lugar a la definición de un plan como un **orden parcial** de las operaciones en las n transacciones. Sin embargo, se debe especificar un orden total en el plan para cualquier par de operaciones en conflicto (condición 3) y para cualquier par de operaciones de la misma transacción (condición 2). La condición 1 simplemente dice que todas las operaciones en las transacciones deben aparecer en el plan completo. Puesto que toda transacción ha sido confirmada o abortada, un plan completo nunca contendrá transacciones activas.

En general, es difícil encontrar planes completos en un sistema de procesamiento de transacciones, porque continuamente se están introduciendo nuevas transacciones en el sistema. Es por ello que conviene definir el concepto de **proyección confirmada** $C(P)$ de un plan P , que incluye sólo las operaciones de P que pertenecen a transacciones confirmadas; esto es, transacciones T_i cuya operación de confirmación c_i está en P .

17.4.2 Caracterización de planes con base en su recuperabilidad

En algunos planes resulta fácil recuperarse de fallos de transacciones, pero en otros dicho proceso puede ser bastante complicado. Por ello, es importante caracterizar los tipos de planes en los cuales es posible la recuperación, así como aquellos en los que la recuperación es relativamente simple. En primer lugar, nos gustaría estar seguros de que, una vez que se ha confirmado una transacción T_i , *nunca será necesario revertirla*. Los planes que satisfacen este criterio se denominan **planes recuperables**.

Se dice que un plan P es **recuperable** si ninguna transacción T de P se confirma antes de haberse confirmado todas las transacciones T' que han escrito un elemento que T lee. Se dice que una transacción T lee de la transacción T' en un plan P si T' escribe primero algún elemento X y luego T lo lee. En el plan P , T' no deberá haber abortado antes de que T lea el elemento X , y no deberá haber transacciones que escriban X después de que T' lo haya escrito y antes de que T lo lea (a menos que esas transacciones, de existir, se hayan abortado antes de que T lea X).

El plan P_e dado en la sección anterior es recuperable. Sin embargo, consideremos los dos planes (parciales) P_c y P_d :

P_c : $l_1(X); e_1(X); l_2(X); l_1(Y); e_2(X); c_2; a_1$;

P_d : $l_1(X); e_1(X); l_2(X); l_1(Y); e_2(X); e_1(Y); c_1; c_2$;

P_c no es recuperable, porque T_2 lee el elemento X de T_1 , y luego se confirma antes de que se confirme T_1 . Si T_1 aborta antes de la operación c_2 en P_c , el valor de X que T_2 lee ya no es válido y T_2 deberá abortar *después de haberse confirmado*, dando lugar a un plan que no es recuperable. Para que el plan sea recuperable, la operación c_2 de P_c se debe posponer hasta después de que T_1 se confirme, como se muestra en P_d .

En un plan recuperable, ninguna transacción confirmada tiene que revertirse jamás. Sin embargo, sí es posible que ocurra un fenómeno conocido como *reversión en cascada* (o *aborto en cascada*), en el que una transacción no confirmada tiene que revertirse porque leyó un elemento de una transacción fallida. Esto se ilustra en el plan P_e donde la transacción T_2 tiene que revertirse porque leyó el elemento X de T_1 , y ésta abortó subsecuentemente:

P_e : $l_1(X); e_1(X); l_2(X); l_1(Y); e_2(X); e_1(Y); a_1$;

Como la reversión en cascada puede consumir mucho tiempo —pues podría ser preciso revertir un gran número de transacciones (véase el Cap. 19)— es importante caracterizar los planes en los que se garantiza que este fenómeno no ocurrirá. Se dice que un plan **evita la reversión en cascada** si toda transacción del plan sólo lee elementos escritos por transacciones *confirmadas*. En este caso, todos los elementos leídos se confirmarán, y no ocurrirá ninguna reversión en cascada. Para satisfacer este criterio, la orden $l_2(X)$ del plan P_e deberá posponerse hasta después de que T_1 se haya confirmado (o abortado), retrasándose así T_2 , pero asegurándose que no habrá reversión en cascada si T_1 aborta.

Por último, hay un tercer tipo de plan, más restrictivo, llamado **plan estricto**, en el que las transacciones no pueden leer *ni escribir* un elemento X en tanto no se haya confirmado (o abortado) la última transacción que escribió X . Los planes estrictos simplifican el proceso de recuperar las operaciones de escritura, reduciéndolo a una restauración de la **imagen "anterior"** de un elemento de información X , que es el valor que X tenía antes de la operación de escritura abortada. Puede ser que este sencillo procedimiento no siempre funcione correctamente si el plan no es estricto. Por ejemplo, consideremos el plan P_f :

P_f : $e_1(X, 5); e_2(X, 8); a_1$;

Supongamos que originalmente el valor de X era 9, lo cual es la imagen "anterior" almacenada en la bitácora del sistema junto con la operación $e_1(X, 5)$. Si T_1 aborta, como en P_f , el procedimiento de recuperación que restaura la imagen "anterior" de una operación de escritura abortada restaurará el valor de X a 9, aunque la transacción T_2 ya lo haya cambiado a 8, lo que da lugar a resultados potencialmente incorrectos. Aunque P_f evita los abortos en cascada, no es un plan estricto, pues permite a T_2 escribir el elemento X aunque la transacción T_1 que fue la última en escribir X todavía no se haya confirmado (o abortado). Un plan estricto no tiene este problema; también es recuperable y evita la reversión en cascada.

Ya hemos caracterizado los planes en términos de su recuperabilidad. En la siguiente sección caracterizaremos el tipo de planes que se consideran correctos cuando se ejecutan transacciones concurrentes; éstos se denominan **planes seriabiles**.

17.5 Seriabilidad de los planes

Supongamos que dos usuarios —encargados de las reservas de vuelos en una línea aérea— introducen al SCBD las transacciones T_1 y T_2 de la figura 17.2 aproximadamente al mismo tiempo. Si no se permite la intercalación, sólo hay dos formas posibles de ordenar las operaciones de las dos transacciones para su ejecución:

1. Ejecutar todas las operaciones de la transacción T_1 (en orden) seguidas de todas las operaciones de la transacción T_2 (en orden).
2. Ejecutar todas las operaciones de la transacción T_2 (en orden) seguidas de todas las operaciones de la transacción T_1 (en orden).

Estas alternativas se muestran en las figuras 17.5(a) y (b), respectivamente. Si se permite la intercalación de operaciones, habrá muchos órdenes posibles en que el sistema podrá ejecutar las operaciones individuales de las transacciones. Dos posibles planes se ilustran en la figura 17.5(c).

Un aspecto importante del control de concurrencia es la llamada **teoría de la seriabilidad**, con la que se intenta determinar cuáles planes son "correctos" y cuáles no, e inventar técnicas que sólo permitan planes correctos. En esta sección se define la seriabilidad de los planes, se presenta parte de esta teoría y se explica cómo puede usarse en la práctica.

17.5.1 Planes en serie, no en serie y seriabiles por conflictos

La figura 17.5 muestra varios planes posibles para las transacciones T_1 y T_2 de la figura 17.2. Los planes A y B de las figuras 17.5(a) y (b) se denominan **planes en serie** porque las operaciones de cada transacción se ejecutan de manera consecutiva, sin operaciones intercaladas

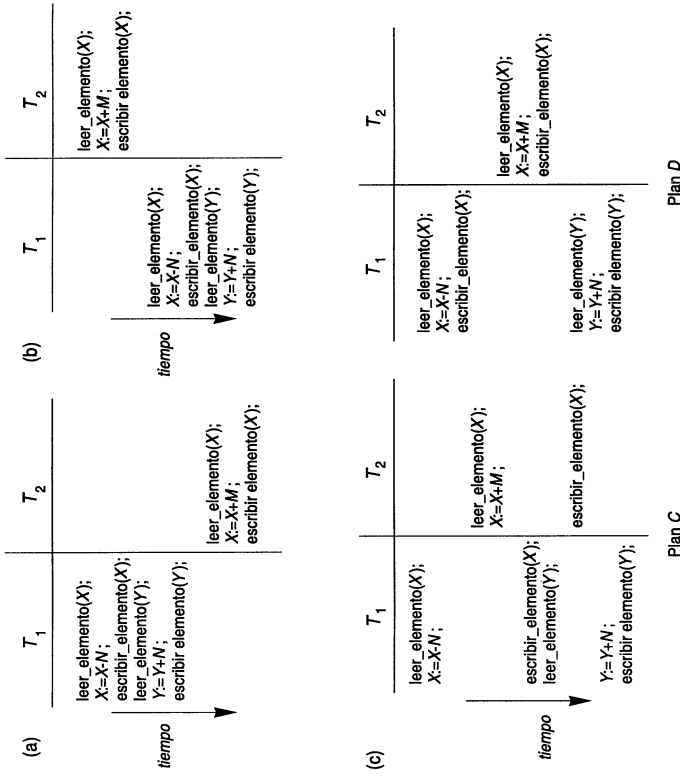


Figura 17.5 Algunos planes en los que intervienen las transacciones T_1 y T_2 . (a) Plan A: T_1 seguida de T_2 . (b) Plan B: T_1 seguida de T_2 . (c) Dos planes con intercalación de operaciones.

de la otra transacción. En un plan en serie, se llevan a cabo transacciones completas en serie: T_1 y luego T_2 en la figura 17.5(a), y T_2 y luego T_1 en la figura 17.5(b). Los planes C y D de la figura 17.5(c) son *no en serie* porque cada secuencia intercala operaciones de las dos transacciones.

En términos formales, un plan P es **en serie** si, por cada transacción T que participa en el plan, todas las operaciones de T se ejecutan consecutivamente en el plan; de lo contrario, el plan **no es en serie**. Una suposición razonable que podemos hacer, si consideramos que las transacciones son *independientes*, es que *todo plan en serie se considera correcto*. La razón es que suponemos que toda transacción es correcta si se ejecuta por sí sola (por la propiedad de conservación de la consistencia, Sec. 17.3), y que las transacciones son independientes entre sí. Por ello, no importa cuál transacción se ejecute primero. En tanto toda transacción se ejecute de principio a fin sin que la interfieran las operaciones de otras transacciones, obtendremos un resultado final correcto en la base de datos. El problema con los planes en serie es que limitan la concurrencia o la intercalación de las operaciones. En un plan en serie, si una transacción está esperando que se complete una operación de E/S, no podremos conmutar el procesador a otra transacción, desperdiándose así un tiempo valioso de procesamiento de la UCP y haciendo a los planes en serie inaceptables en general.

Para ilustrar lo anterior, consideremos los planes de la figura 17.5, y supongamos que los valores iniciales de los elementos de información son $X = 90$, $Y = 90$, y que $N = 3$ y $M = 2$.

Después de ejecutar las transacciones T_1 y T_2 , esperaríamos que los valores de la base de datos fueran $X = 89$ y $Y = 93$, de acuerdo con el significado de las transacciones. Y así es; la ejecución de cualquiera de los planes en serie A o B da los resultados correctos. Consideremos ahora los planes no en serie C y D. El plan C (igual al de la Fig. 17.3(a)) da los resultados $X = 92$ y $Y = 93$, donde el valor de X es erróneo, y el plan D da los resultados correctos.

El plan C produce un resultado erróneo debido al problema de la actualización perdida que vimos en la sección 17.1.3; la transacción T_2 lee el valor de X antes de que la transacción T_1 lo modifique, así que sólo el efecto de T_2 sobre X se refleja en la base de datos. El efecto de T_1 sobre X se *pierde*, pues T_2 lo sobrescribe, dando lugar al resultado incorrecto para el elemento X . No obstante, algunos planes no en serie producen el resultado correcto esperado, como el plan D. Nos gustaría determinar cuáles de los planes no en serie siempre dan un resultado correcto y cuáles pueden dar resultados erróneos. El concepto con el que se caracterizan los planes de esta manera es el de la seriabilidad de un plan.

Un plan P de n transacciones es **seriable** si es *equivalente a algún plan en serie* de las mismas n transacciones. Definiremos el concepto de equivalencia de los planes en un momento más. Observe que existen (n)! posibles planes en serie de n transacciones y muchos más planes no en serie posibles. Podemos formar dos grupos disjuntos de los planes que no son en serie: los que son equivalentes a uno (o más) de los planes en serie, y por tanto son seriables; y los que no son equivalentes a *ningún* plan en serie, y por tanto no son seriables. Decir que un plan no en serie P es seriable equivale a decir que es correcto, porque es equivalente a un plan en serie, que se considera correcto. La duda que queda es: ¿cuándo se considerarán "equivalentes" dos planes? Hay varias formas de definir la equivalencia entre planes. La definición más simple, pero menos satisfactoria, implica comparar los efectos de los planes sobre la base de datos. Se dice que dos planes son **equivalentes por resultados** si producen el mismo estado final en la base de datos. Sin embargo, dos planes distintos pueden producir el mismo estado final por accidente. Por ejemplo, en la figura 17.6, los planes P_1 y P_2 producen el mismo estado final si se ejecutan en una base de datos en la que el valor inicial de X es 100, pero para otros valores iniciales de X los dos planes *no* son equivalentes por resultados. Es más, estos dos planes ejecutan diferentes transacciones, por lo que en definitiva no deben considerarse equivalentes. Es por ello que la equivalencia por resultados *no* sirve para definir la equivalencia entre dos planes.

El enfoque más seguro y general para definir la equivalencia de dos planes consiste en no hacer suposición alguna sobre los tipos de operaciones que intervienen en las transacciones. Para que dos planes sean equivalentes, las operaciones que se aplican a cada uno de los elementos de información afectados por los planes se deben aplicar a ese elemento en el mismo orden en ambos planes. Por lo regular se aceptan dos definiciones de equivalencia de planes: *equivalencia por conflictos* y *equivalencia de vistas*. A continuación haremos un análisis de ellas.

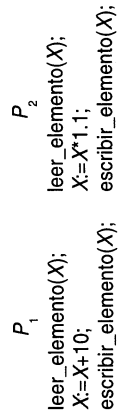


Figura 17.6 Dos planes que son equivalentes para el valor inicial de $X = 100$, pero que no son equivalentes en lo general.

Se dice que dos planes son **equivalentes por conflictos** si el orden de cualesquier dos operaciones en *conflicto* es el mismo en ambos planes. Recuerde lo dicho en la sección 17.4.1 respecto a que dos operaciones de un plan están en *conflicto* si pertenecen a diferentes transacciones, si tienen acceso al mismo elemento de la base de datos, y si una de las dos operaciones es escribir elemento. Si dos operaciones en conflicto se aplican en *diferente orden* en dos planes, el efecto de los planes puede ser diferente, ya sea sobre las transacciones o sobre la base de datos, de modo que los dos planes no son equivalentes por conflictos. Por ejemplo, si una operación de lectura y una de escritura ocurren en el orden $l_1(X)$, $e_2(X)$ en un plan P_a y en el orden inverso $e_2(X)$, $l_1(X)$ en un plan P_b , el valor que lee la operación $l_1(X)$ puede ser diferente en los dos planes. De manera similar, si dos operaciones de escritura se efectúan en el orden $e_1(X)$, $e_2(X)$ en un plan P_a y en el orden inverso $e_2(X)$, $e_1(X)$ en otro plan P_b , es posible que la siguiente operación $l(X)$ de los dos planes lea valores distintos; o bien, si las dos operaciones de escritura son las últimas que tienen acceso a X en los dos planes, el valor final del elemento X podrá ser distinto para los dos planes.

Con la noción de equivalencia por conflictos, definimos que un plan P es **seriable por conflictos** si es equivalente (por conflictos) a algún plan en serie P' . En tal caso, podremos reordenar las operaciones de P que *no están en conflicto* hasta formar el plan en serie equivalente P' . Según esta definición, el plan D de la figura 17.5(c) es equivalente al plan en serie A de la figura 17.5(a). En ambos planes, la operación leer elemento(X) de T_2 lee el valor de X que T_1 escribió, en tanto que las demás operaciones leer elemento leen los valores del estado inicial de la base de datos. Además, T_1 es la última transacción que escribe el elemento Y y T_2 es la última transacción que escribe X en ambos planes. Como A es un plan en serie y el plan D es equivalente a A , D es un *plan seriable*. Advuértase que las operaciones $l_1(Y)$ y $e_1(Y)$ del plan D no están en conflicto con las operaciones $l_1(X)$ y $e_2(X)$, ya que tienen acceso a diferentes elementos de información. Así pues, podemos ejecutar $l_1(Y)$, $e_1(Y)$ antes que $l_2(X)$, $e_2(X)$, dando lugar al plan en serie equivalente T_1, T_2 .

El plan C de la figura 17.5(c) no es equivalente a ninguno de los dos planes en serie posibles A y B . No es equivalente a A porque la operación leer elemento(X) de T_2 lee el valor escrito por T_1 en el plan A , pero lee el valor original de X en el plan C , violando así la regla 1 de equivalencia. Por lo mismo, el plan C no es equivalente a B . Por tanto, el plan C *no es seriable*. Si tratamos de reordenar las operaciones del plan C para encontrar un plan en serie equivalente, fracasaremos porque $l_2(X)$ y $e_1(X)$ están en conflicto y no podemos mover $l_1(X)$ hacia abajo para obtener el plan en serie equivalente T_1, T_2 . De manera similar, como $e_1(X)$ y $e_2(X)$ están en conflicto, no podemos desplazar $e_1(X)$ hacia abajo para obtener el plan en serie equivalente T_2, T_1 .

En la sección 17.5.4 se examinará otra definición de equivalencia, más compleja —llamada *equivalencia de vistas*, que conduce al concepto de *seriabilidad por vistas*—.

17.5.2 Prueba de la seriabilidad por conflictos de un plan

Existe un algoritmo sencillo para determinar si un plan es seriable por conflictos. La mayor parte de los métodos de control de concurrencia *no* comprueban realmente la seriabilidad; más bien, se elaboran protocolos, o reglas, que garantizan que un plan es seriable. Analizaremos aquí el algoritmo para comprobar la seriabilidad por conflictos de los planes a fin de entender mejor dichos protocolos, que veremos en el capítulo 18.

El algoritmo 17.1 puede servir para comprobar si un plan es seriable por conflictos. El algoritmo examina sólo las operaciones leer elemento y escribir elemento del plan para

construir un **grafo de precedencia** (o **grafo de seriación**). Un grafo de precedencia es un **grafo dirigido** $G = (N, A)$ que consiste en un conjunto de nodos $N = \{T_1, T_2, \dots, T_n\}$ y un conjunto de aristas dirigidas $A = \{a_1, a_2, \dots, a_m\}$. El grafo contiene un nodo por cada transacción T_i del plan. Cada arista a_i del grafo tiene la forma $(T_i \rightarrow T_j)$, con $1 \leq j \leq n$, $1 \leq k \leq n$, donde T_j es el **nodo inicial** de a_i y T_k es el **nodo final** de a_i de tal modo que una de las operaciones de T_j aparecerá en el plan antes que alguna operación en conflicto de T_k .

ALGORITMO 17.1 Prueba de la seriabilidad por conflictos de un plan P

- (1) para cada transacción T_i que participa en el plan P
crear un nodo rotulado T_i en el grafo de precedencia;
- (2) para cada caso en P donde T_j ejecuta una orden leer elemento(X) después de que T_i ejecuta una orden escribir elemento(X)
crear una arista $(T_i \rightarrow T_j)$ en el grafo de precedencia;
- (3) para cada caso en P donde T_j ejecuta escribir elemento(X) después de que T_i ejecuta leer elemento(X)
crear una arista $(T_i \rightarrow T_j)$ en el grafo de precedencia;
- (4) para cada caso en P donde T_j ejecuta una orden escribir elemento(X) después de que T_i ejecuta una orden escribir elemento(X)
crear una arista $(T_i \rightarrow T_j)$ en el grafo de precedencia;
- (5) el plan P es seriable si y sólo si el grafo de precedencia no tiene ciclos;

El grafo de precedencia se construye como en el algoritmo 17.1. Si hay un ciclo en el grafo, el plan P no es seriable (por conflictos); si no hay ciclos, P es seriable. Un ciclo en un grafo dirigido es una *secuencia* de aristas $C = ((T_i \rightarrow T_j), (T_j \rightarrow T_k), \dots, (T_n \rightarrow T_i))$ con la propiedad de que el nodo inicial de cada arista —con excepción de la primera— es el mismo que el nodo final de la arista anterior, y el nodo inicial de la primera arista es el mismo que el nodo final de la última arista. Así pues, la secuencia comienza y termina en el mismo nodo.

En el grafo de precedencia, una arista de T_i a T_j indica que la transacción T_i debe ir antes que la transacción T_j en cualquier plan en serie que sea equivalente a P , porque dos operaciones en conflicto aparecerán en el plan en ese orden. Si no hay ciclos en el grafo de precedencia, podemos crear un **plan en serie** P' que sea **equivalente** a P , ordenando las transacciones que participan en P como sigue: Siempre que en el grafo de precedencia haya una arista de T_i a T_j , T_i deberá aparecer antes que T_j en el plan en serie equivalente P' . Este proceso se denomina *ordenación topológica*. Observe que las aristas $(T_i \rightarrow T_j)$ en un grafo de precedencia se pueden rotular también con el nombre o nombres de los elementos de información que originaron la creación de esa arista.

En general, varios planes en serie pueden ser equivalentes a P si el grafo de precedencia no tiene ciclos. Por otro lado, si el grafo tiene un ciclo, es fácil demostrar que no podemos crear ningún plan en serie equivalente, por lo que el plan P no será seriable. Los grafos de precedencia creados para los planes A a D , respectivamente, de la figura 17.5 aparecen en las figuras 17.7(a) a (d). El grafo de precedencia del plan C tiene un ciclo, así que no será seriable. El grafo del plan D no tiene ciclos, de modo que es seriable, y el plan en serie equivalente es T_1 seguida de T_2 . Los grafos de los planes A y B no tienen ciclos, como era de esperar, porque los planes son en serie y por tanto seriables.

Otro ejemplo, en el que participan tres transacciones, se muestra en la figura 17.8. La figura 17.8(a) ilustra las operaciones leer elemento y escribir elemento de cada transacción. En la figura 17.8(b) y (c) se muestran dos planes E y F , respectivamente, para esas

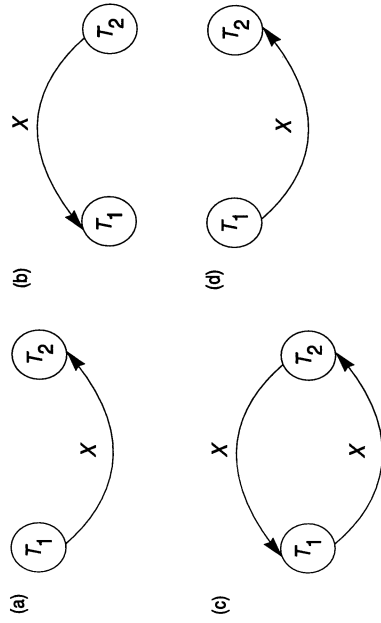


Figura 17.7 Construcción de grafos de precedencia para los planes A a D, a fin de comprobar la serializabilidad por conflictos. (a) Grafo de precedencia del plan A de la figura 17.5. (b) Grafo de precedencia del plan B de la figura 17.5. (c) Grafo de precedencia del plan C de la figura 17.5 (no serializable). (d) Grafo de precedencia del plan D de la figura 17.5 (serializable, equivalente al plan A).

transacciones. Los grafos de precedencia de estos dos planes se muestran en la figura 17.8(d) y (e). El plan E no es serializable, porque el grafo de precedencia correspondiente tiene ciclos. El plan F es serializable, y el plan en serie equivalente a F se muestra en la figura 17.8(e). Aunque sólo existe un plan en serie equivalente para el plan F, en general puede haber más de un plan en serie equivalente a un plan serializable dado. La figura 17.8 (f) muestra un grafo de precedencia que representa un plan que tiene dos planes en serie equivalentes.

17.5.3 Aplicaciones de la serializabilidad

Como vimos antes, decir que un plan P es serializable (por conflictos) —esto es, que P es equivalente (por conflictos) a un plan en serie— es tanto como decir que P es correcto. Sin embargo, ser *serializable* no es lo mismo que ser *en serie*. Un plan en serie representa un proceso ineficiente porque no se permite la intercalación de operaciones de diferentes transacciones. Esto puede dar pie a un bajo aprovechamiento de la UCP mientras una transacción espera una E/S de disco, haciendo al procesamiento bastante más lento. Un plan serializable ofrece los beneficios de la ejecución concurrente sin dejar de ser correcto.

En la práctica, es muy difícil comprobar la serializabilidad de un plan. Casi siempre, el planificador del sistema operativo determina la intercalación de operaciones de transacciones concurrentes. Factores como la carga del sistema, el momento de introducción de las transacciones y las prioridades de las transacciones contribuyen al orden en que el sistema operativo coloca las operaciones en un plan. Por esto, es prácticamente imposible determinar por anticipado cómo se intercalarán las operaciones de un plan cuando se intenta asegurar la serializabilidad.

Si las transacciones se ejecutan a voluntad y luego se comprueba si el plan es serializable, deberemos cancelar el efecto del plan si resulta que no es serializable. Éste es un problema grave que hace poco práctico este enfoque. Por ello, la estrategia que se sigue en casi todos los sistemas prácticos es encontrar métodos que garanticen la serializabilidad sin tener que

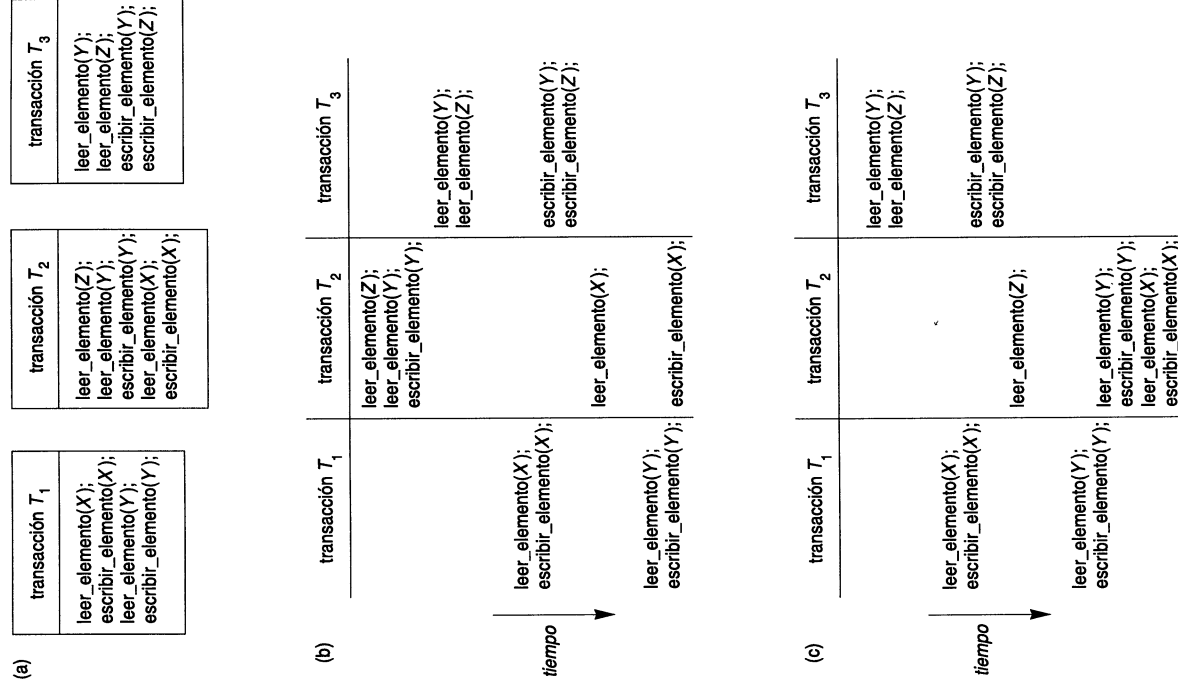


Figura 17.8 Otro ejemplo de prueba de serializabilidad. (a) Las operaciones LEER y ESCRIBIR de tres transacciones. (b) Plan E de las transacciones T_1 , T_2 y T_3 . (c) Plan F de las transacciones T_1 , T_2 y T_3 . (continúa en la siguiente página)

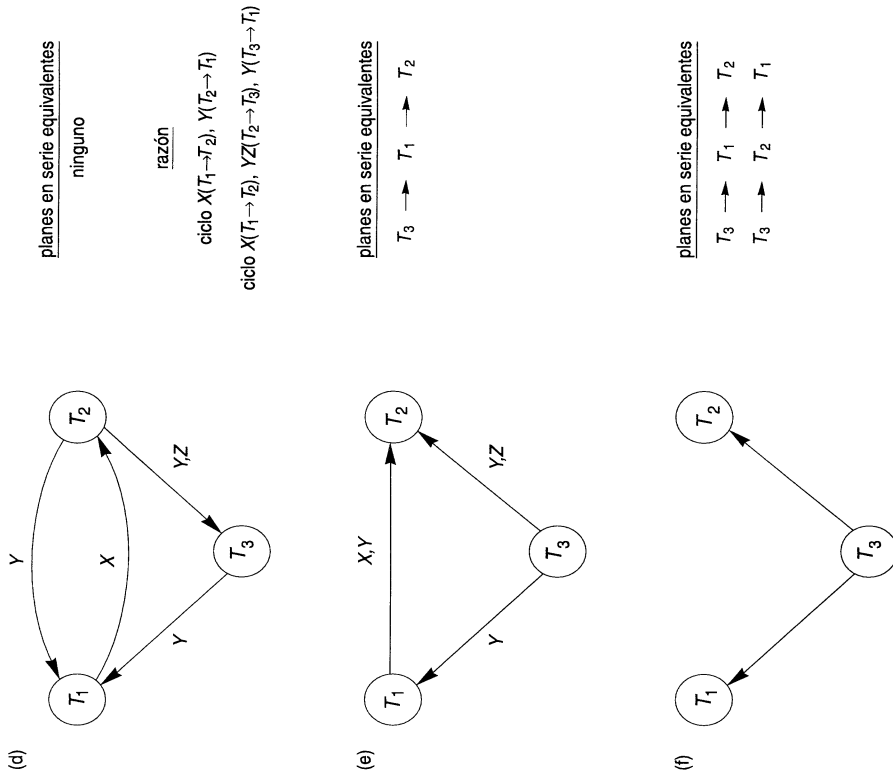


Figura 17.8 (continuación) (d) Grafo de precedencia del plan E. (e) Grafo de precedencia del plan F. (f) Grafo de precedencia con dos planes en serie equivalentes.

verificar la seriabilidad de los planes mismos después de haberlos ejecutado. Uno de estos métodos se basa en la teoría de la seriabilidad para determinar **protocolos** o conjuntos de reglas que, si todas las transacciones individuales los siguen o si un subsistema de control de concurrencia del SGBD los impone, garanticen la seriabilidad de todos los planes en los que participen las transacciones. Así pues, con este enfoque nunca tendremos que ocuparnos de los planes mismos.

Otro problema es que, cuando las transacciones se introducen continuamente en el sistema, es difícil determinar cuándo comienza y cuándo termina un plan. La teoría de la seriabilidad puede adaptarse para resolver este problema considerando exclusivamente la proyección confirmada de un plan P . Recuerde que en la sección 17.4.1 se dijo que la proyección confirmada $C(P)$ de un plan P incluye sólo las operaciones de P que pertenecen a

transacciones confirmadas. Podemos definir que un plan P es serializable si su proyección confirmada $C(P)$ es equivalente a algún plan en serie, ya que el SGBD sólo garantiza las transacciones confirmadas.

En el capítulo 18 estudiaremos varios protocolos de control de concurrencia que garantizan la seriabilidad. La técnica más común, el *bloqueo de dos fases*, se basa en bloquear elementos de información para evitar la interferencia entre transacciones concurrentes. Otra técnica se basa en el *ordenamiento por marca de tiempo*, donde a cada transacción se le asigna una marca de tiempo única y el protocolo garantiza que cualesquier operaciones en conflicto se ejecutarán en el orden de las marcas de tiempo de las transacciones. Otras técnicas se basan en mantener *múltiples versiones* de los elementos de información y en la *certificación* o *validación*.

Otro factor que afecta el control de concurrencia es la **granularidad** de los elementos; esto es, qué porción de la base de datos representa un elemento. Un elemento puede ser tan pequeño como un valor individual o tan grande como la base de datos misma. Desde luego, en este último caso es muy poca la concurrencia permitida. Hablaremos de la granularidad de los elementos en la sección 18.5.

17.5.4 Equivalencia de vistas y seriabilidad por vistas*

En la sección 17.5.1 definimos los conceptos de equivalencia por conflictos para los planes y de seriabilidad por conflictos. Otra definición menos restrictiva de la equivalencia entre planes se denomina *equivalencia de vistas*. Esto conduce a otra definición de la seriabilidad llamada *seriabilidad por vistas*. Se dice que dos planes son **equivalentes por vistas** si se cumplen estas tres condiciones:

1. El mismo conjunto de transacciones participa en P y en P' , y P y P' incluyen las mismas operaciones de esas transacciones.
2. Para cualquier operación $l_i(X)$ de T_i en P , si el valor de X que lee la operación fue escrito por una operación $e_j(X)$ de T_j (o si es el valor original de X antes de comenzar el plan), debe cumplirse la misma condición para el valor de X que lee la operación $l_i(X)$ de T_i en P' .
3. Si la operación $e_k(Y)$ de T_k es la última operación que escribe el elemento Y en P , entonces $e_k(Y)$ de T_k debe ser también la última operación que escribe Y en P' .

La equivalencia de vistas se basa en la idea de que, en tanto cada operación de lectura de una transacción lea el resultado de la *misma operación de escritura* en ambos planes, las operaciones de escritura de ambas transacciones producirán los mismos resultados. Así pues, se dice que las operaciones de lectura perciben *la misma vista* en ambos planes. La condición 3 garantiza que la operación de escritura final de cada elemento de información sea la misma en ambos planes, con lo que el estado de la base de datos deberá ser el mismo al final de los dos. Se dice que un plan P es **serializable por vistas** si es equivalente por vistas a un plan en serie.

Las definiciones de seriabilidad por conflictos y por vistas resultan similares si se cumple una condición conocida como suposición de **escritura restringida** en todas las transacciones del plan. Esta condición dice que cualquier operación de escritura $e_i(X)$ en T_i va precedida por una operación $l_i(X)$ en T_i , y que el valor escrito por $e_i(X)$ en T_i depende sólo

del valor de X que $l_1(X)$ lee. Esto supone que el cálculo del nuevo valor de X es una función $f(X)$ basada en el antiguo valor de X leído de la base de datos. Sin embargo, la definición de seriabilidad por vistas es menos restringida que la de seriabilidad por conflictos si se supone la **escritura no restringida**, donde el valor escrito por una operación $e_i(X)$ en T_i puede ser independiente de cualquier valor anterior de la base de datos. Esto se conoce como **escritura ciega**, y se ilustra con el siguiente plan de tres transacciones $T_1: l_1(X), e_1(X); T_2: e_2(X); T_3: e_3(X)$:

$P_s: l_1(X); e_2(X); e_1(X); e_3(X); c_1; c_2; c_3;$

En P_s , las operaciones $e_2(X)$ y $e_3(X)$ son escrituras ciegas, porque ni T_2 ni T_3 leen el valor de X . El plan P_s es serializable por vistas, ya que es equivalente por vistas al plan en serie T_1, T_2, T_3 . Sin embargo, P_s no es serializable por conflictos, pues no es equivalente por conflictos a ningún plan en serie.

Se ha demostrado que cualquier plan serializable por conflictos también es serializable por vistas, pero no viceversa, como lo ilustra el ejemplo anterior. Hay un algoritmo para comprobar si un plan P es serializable por vistas o no, pero se ha demostrado que el problema de comprobar la seriabilidad por vistas es NP-completo, lo que significa que la probabilidad de encontrar un algoritmo eficiente para este problema es muy baja.

17.5.5 Otros tipos de equivalencia de planes*

Hay quienes consideran la seriabilidad de los planes demasiado restrictiva como condición para asegurar la corrección de las ejecuciones concurrentes. Algunas aplicaciones pueden producir planes correctos satisfaciendo condiciones menos exigentes que la seriabilidad por conflictos o la seriabilidad por vistas. Un ejemplo de ello son las transacciones de cargo y abono, como las que aplican depósitos y retiros a un elemento de información cuyo valor es el saldo actual de una cuenta bancaria. La semántica de las operaciones de cargo y abono es que actualizan el valor de un elemento de información X reduciendo o aumentando el valor de dicho elemento. Dado que las operaciones de resta y de suma son conmutativas —es decir, se pueden aplicar en cualquier orden— es posible producir planes correctos que no sean serializables. Por ejemplo, consideremos las dos transacciones que siguen, cada una de las cuales puede servir para transferir una cantidad de dinero entre dos cuentas bancarias:

$T_1: l_1(X); X := X - 10; e_1(X); l_1(Y); Y := Y + 10; e_1(Y);$
 $T_2: l_1(Y); Y := Y - 20; e_2(Y); l_1(Z); Z := Z + 20; e_2(Z);$

Consideremos el siguiente plan P para las dos transacciones:

$P: l_1(X); e_1(X); l_2(Y); e_2(Y); l_1(Y); e_1(Y); l_2(Z); e_2(Z);$

Con el conocimiento adicional, o *semántica*, de que las operaciones entre cada $l_i(X)$ y $e_i(X)$ son conmutativas, sabemos que el orden de ejecutar las secuencias que consisten en (leer, actualizar, escribir) no es importante en tanto no se interrumpa ninguna de estas secuencias. Por esto, se considera que el plan P es correcto aunque no es serializable. Recientemente, algunos investigadores han estado tratando de extender la teoría del control de la concurrencia para manejar casos en los que la seriabilidad se considera una condición demasiado restrictiva para la corrección de los planes.

17.6 Resumen

En este capítulo analizamos conceptos de SGBD para el procesamiento de transacciones. En la sección 17.1 presentamos el concepto de transacción de base de datos y las operaciones que intervienen en dicho procesamiento. Comparamos los sistemas monousuario y multiusuario y luego examinamos algunos ejemplos de cómo la ejecución no controlada de transacciones concurrentes en un sistema multiusuario puede dar pie a resultados y valores incorrectos en la base de datos. También vimos los diversos tipos de fallos que pueden ocurrir durante la ejecución de transacciones.

En la sección 17.2 presentamos primero los estados por los que suele pasar una transacción durante su ejecución; luego analizamos varios conceptos que se usan en los métodos de recuperación y de control de concurrencia. La bitácora del sistema sigue la pista a los accesos a la base de datos y utiliza esta información para recuperarse de los fallos. Una transacción o bien tiene éxito y llega a su punto de confirmación o falla y debe revertirse. Las modificaciones de una transacción confirmada se asientan permanentemente en la base de datos. Los puntos de control sirven para indicar que todas las transacciones confirmadas hasta el punto de control han asentado sus actualizaciones permanentemente en la base de datos. En la sección 17.3 presentamos un panorama sobre las propiedades deseables de las transacciones, a saber, atomicidad, conservación de la consistencia, aislamiento y durabilidad, a las que se suele llamar propiedades ACID.

En la sección 17.4 definimos primero un plan (o historia) como una secuencia de ejecución de las operaciones de varias transacciones con una posible intercalación. En seguida caracterizamos los planes en términos de su recuperabilidad. Los planes recuperables garantizan que, una vez confirmada una transacción, nunca tendrá que anularse. Los planes sin cascada aseguran que ninguna transacción abortada requerirá que otras transacciones se aborten en cascada. Con los planes estrictos tenemos un esquema de recuperación simple que consiste en reescribir los antiguos valores de los elementos que hayan sido modificados por una transacción abortada.

En la sección 17.5 definimos la equivalencia entre planes y vimos que un plan serializable es equivalente a algún plan en serie. Definimos los conceptos de equivalencia por conflictos y equivalencia por vistas, que conducen a los conceptos de seriabilidad por conflictos y seriabilidad por vistas. Se considera que un plan serializable es correcto. Después presentamos algoritmos para comprobar la seriabilidad (por conflictos) de un plan. Por último, analizamos por qué la comprobación de la seriabilidad no es práctica en un sistema real, aunque puede servir para definir y verificar los protocolos de control de concurrencia, y mencionamos brevemente otras definiciones menos restrictivas de la equivalencia entre planes.

En el capítulo 18 trataremos varias técnicas de control de concurrencia, en el capítulo 19 estudiaremos las técnicas para la recuperación de fallos de transacciones.

Preguntas de repaso

17.1. ¿Qué significa *ejecución concurrente* cuando hablamos de *transacciones de bases de datos* en un sistema multiusuario? Comente por qué es necesario el control de la concurrencia, y dé ejemplos informales.

17.2. Explique los diferentes tipos de fallos de transacciones. ¿Qué significa *fallo catastrófico*?

- 17.3. Analice las acciones que realizan las operaciones leer_elemento y escribir_elemento en una base de datos.
- 17.4. Dibuje un diagrama de estados, y analice los estados por los que suele pasar una transacción durante su ejecución.
- 17.5. ¿Para qué sirven las bitácoras del sistema, y qué tipos de entradas suelen contener? ¿Qué son los puntos de control, y por qué son importantes? ¿Qué son los puntos de confirmación de las transacciones, y por qué son importantes?
- 17.6. Explique las propiedades de atomicidad, conservación de la consistencia, aislamiento y durabilidad de una transacción de base de datos.
- 17.7. ¿Qué es un plan (historia)? Defina los conceptos de plan recuperable, sin cascada y estricto, y compárelos en términos de su recuperabilidad.
- 17.8. Analice las diferentes medidas de equivalencia entre transacciones. ¿Qué diferencia hay entre la equivalencia por conflictos y la equivalencia por vistas?
- 17.9. ¿Qué es un plan en serie? ¿Qué es un plan serializable? ¿Por qué se considera correcto un plan en serie? ¿Por qué se considera correcto un plan serializable?
- 17.10. ¿Qué diferencia hay entre las suposiciones de escritura restringida y no restringida? ¿Cuál es más realista?
- 17.11. Explique cómo se aplica la serializabilidad para imponer el control de concurrencia en un sistema de base de datos. ¿Por qué a veces se considera la serializabilidad demasiado restrictiva como medida de la corrección de los planes?

Ejercicios

- 17.12. Modifique la transacción T_2 de la figura 17.2(b) de modo que sea:
 leer_elemento(X);
 $X := X + M$;
 si $X > 90$ entonces salir
 de otro modo escribir_elemento(X);
 Analice el resultado final de los diferentes planes de la figura 17.3, donde $M = 2$ y $N = 2$, respecto a las siguientes preguntas. La adición de la condición anterior, altera el resultado final? ¿Obedece el resultado la regla de consistencia implícita (que la capacidad de X es 90)?
- 17.13. Repita el ejercicio 17.12 añadiendo un control en T_1 de modo que Y no exceda 90.
- 17.14. Añada la operación confirmar al final de las dos transacciones T_1 y T_2 de la figura 17.2; luego haga una lista de todos los planes posibles para las transacciones modificadas. Determine cuáles de los planes son recuperables, cuáles son sin cascada y cuáles son estrictos.
- 17.15. Haga una lista con todos los planes posibles para las transacciones T_1 y T_2 de la figura 17.2, y determine cuáles son serializables por conflictos (correctos) y cuáles no.
- 17.16. ¿Cuántos planes en serie existen para las tres transacciones de la figura 17.8(a)?

- 17.17. Escriba un programa para crear todos los planes posibles para las tres transacciones de la figura 17.8(a), y para determinar cuáles de esos planes son serializables por conflictos y cuáles no. Para cada plan serializable por conflictos, el programa deberá imprimir el plan y una lista de todos los planes en serie equivalentes.

Bibliografía selecta

El concepto de transacción atómica se analiza en Gray (1981). El libro de Bernstein, Hadzilacos y Goodman (1987) está dedicado a las técnicas de control de concurrencia y de recuperación en sistemas de bases de datos, tanto centralizados como distribuidos; es una referencia excelente. El libro de Papadimitriou (1986) ofrece una perspectiva más teórica. Un libro de referencia de más de 1000 páginas, Gray y Reuter (1993), es una perspectiva más práctica sobre los conceptos y técnicas del procesamiento de transacciones. Dos volúmenes editados —Elmagarmid (1992) y Bhargava (1989)— ofrecen colecciones de artículos de investigación sobre el procesamiento de transacciones.

Los conceptos de serializabilidad se presentaron por primera vez en Gray *et al.* (1975). La serializabilidad por vistas se define en Yannakakis (1984). La recuperabilidad de los planes se analiza en Hadzilacos (1983, 1986).

18.1 Técnicas de bloqueo para el control de concurrencia

Una de las principales técnicas para controlar la ejecución concurrente de las transacciones se basa en el concepto de bloquear elementos de información. Un **candado** es una variable asociada a un elemento de información de la base de datos y describe el estado de ese elemento respecto a las posibles operaciones que se pueden aplicar a él. En general, hay un candado por cada elemento de información en la base de datos. Usamos los candados como una forma de sincronizar el acceso a los elementos de la base de datos por parte de transacciones concurrentes. En la sección 18.1.1 estudiaremos la naturaleza y los tipos de candados. Después, en la sección 18.1.2 presentaremos protocolos que garantizan, mediante bloqueo, que los planes de transacciones sean serializables. Por último, en la sección 18.1.3, estudiaremos dos problemas asociados al empleo de candados —a saber, bloqueo mortal y espera indefinida— y explicaremos cómo se resuelven estos problemas.

18.1.1 Tipos de candados

Podemos usar varios tipos de candados en el control de concurrencia. Primero presentaremos los candados binarios, que son simples aunque un tanto restrictivos en su aplicación. En seguida trataremos los candados compartidos y exclusivos, que tienen capacidades de bloqueo más generales.

Candados binarios. Un **candado binario** puede tener dos **estados** o **valores**: **bloqueado** y **desbloqueado** (o 1 y 0, por sencillez). A *cada* elemento X de la base de datos se asocia un candado distinto. Si el valor del candado sobre X es 1, *ninguna* operación de base de datos que solicite el elemento *podrá tener acceso* a él. Si el valor del candado sobre X es 0, se podrá tener acceso al elemento cuando se solicite. Nos referiremos al *valor* del candado asociado al elemento X como $CANDADO(X)$.

Cuando se usa bloqueo binario se debe incluir en las transacciones dos operaciones, **bloquear_elemento** y **desbloquear_elemento**. Una transacción solicita acceso a un elemento X emitiendo una operación **bloquear_elemento**(X). Si $CANDADO(X) = 1$, la transacción tiene que esperar; si no, la transacción asigna 1 a $CANDADO(X)$ (bloquea el elemento) y obtiene el acceso. Cuando la transacción termina de usar el elemento, emite una operación **desbloquear_elemento**(X), que asigna 0 a $CANDADO(X)$ (desbloquea el elemento) para que otras transacciones puedan tener acceso a X . Así pues, un candado binario impone una *exclusión mutua* sobre el elemento de información. En la figura 18.1 se describen las operaciones **bloquear_elemento** y **desbloquear_elemento**.

Observe que estas dos operaciones se deben implementar como unidades indivisibles (conocidas como *secciones críticas* en los sistemas operativos); esto es, no debe permitirse intercalación alguna una vez iniciada una operación de bloqueo o desbloqueo hasta que la operación termine o la transacción espere. En la figura 18.1, la orden de esperar dentro de la operación **bloquear_elemento**(X) suele implementarse colocando la transacción en una cola de espera para el elemento X hasta que X quede desbloqueado y se conceda a la transacción acceso a él. Otras transacciones que también requieran acceso a X se colocarán en la misma cola. Por esto, se considera que la orden de esperar queda fuera de la operación **bloquear_elemento**. El SGBD cuenta con un subsistema **gestor de bloqueo** para mantenerse al tanto del acceso a los candados y controlarlo.

Cuando se utiliza el esquema de bloqueo binario, toda transacción debe obedecer las siguientes reglas:

CAPÍTULO 18

Técnicas de control de concurrencia

En este capítulo analizaremos varias técnicas de control de concurrencia que sirven para garantizar la no interferencia o el aislamiento de transacciones que se ejecutan de manera concurrente. Casi todas ellas aseguran que los planes sean serializables (véase la Sec. 17.5) empleando **protocolos** o conjuntos de reglas que garantizan la serializabilidad. Un importante conjunto de protocolos emplea la técnica de **bloquear** (mediante un **candado**) elementos de información para evitar que más de una transacción tenga acceso a los elementos al mismo tiempo; en la sección 18.1 se describen varios protocolos de bloqueo. Otro conjunto de protocolos para el control de concurrencia utiliza **marcas de tiempo** de transacciones. Una marca de tiempo es un identificador único generado por el sistema para cada transacción. En la sección 18.2 se estudian los protocolos de control de concurrencia que usan ordenamiento de marca de tiempo para asegurar la serializabilidad. En la sección 18.3 veremos los protocolos de control de concurrencia de **multiversión** que emplean múltiples versiones de un elemento de información. En la sección 18.4 presentamos un protocolo basado en el concepto de *validación* o *certificación* de una transacción después de que ejecuta sus operaciones; éstos suelen recibir el nombre de protocolos **optimistas**.

Otro factor que afecta el control de concurrencia es la **granularidad** de los elementos; esto es, qué porción de la base de datos representa un elemento. Un elemento puede ser tan pequeño como un solo valor o tan grande como un bloque de disco, un archivo o incluso la base de datos entera. Estudiaremos la granularidad de los elementos en la sección 18.5. Por último, en la sección 18.6, trataremos algunos aspectos adicionales de control de concurrencia.

1. Una transacción T debe emitir la operación bloquear elemento(X) antes de que se realice cualquier operación leer elemento(X) o escribir elemento(X) en T .
2. Una transacción T debe emitir la operación desbloquear elemento(X) después de haber completado todas las operaciones leer elemento(X) y escribir elemento(X) en T .
3. Una transacción T no emitirá una operación bloquear elemento(X) si ya posee el bloqueo del elemento X .
4. Una transacción T no emitirá una operación desbloquear elemento(X) a menos que ya posea el bloqueo del elemento X .

Un módulo del SCBD puede encargarse de hacer que se cumplan estas reglas. Entre las operaciones bloquear elemento(X) y desbloquear elemento(X) dentro de la transacción T , se dice que T **posee el candado** del elemento X . Como máximo, una transacción puede poseer el candado de un elemento en particular; dos transacciones no pueden tener acceso concurrente al mismo elemento. Adviértase que es muy fácil implementar un candado binario; todo lo que se necesita es una variable de dos valores, CANDADO, asociada a cada elemento X de la base de datos. En su forma más simple, cada candado puede ser un registro con dos campos: <nombre del elemento de información, CANDADO> más una cola para las transacciones que esperan. El sistema sólo tiene que mantener estos registros para los elementos bloqueados en una **tabla de bloqueo**.

Candados compartidos y exclusivos. El esquema de bloqueo binario es demasiado restrictivo en lo general, porque como máximo una transacción puede poseer un candado sobre un elemento dado. Debemos permitir que varias transacciones tengan acceso al mismo elemento X si lo hacen *exclusivamente para leerlo*. Sin embargo, si una transacción va a escribir un elemento X deberá poseer acceso exclusivo a él. Para este fin, podemos usar un tipo de candado diferente llamado **candado de modo múltiple**. En este esquema hay tres operaciones de bloqueo: bloquear lectura(X), bloquear escritura(X) y desbloquear(X). Ahora, un candado asociado a un elemento X , CANDADO(X), tiene tres posibles estados: "bloqueado para leer", "bloqueado para escribir" o "desbloqueado". Un elemento bloqueado para leer posee un **candado compartido**, porque otras transacciones pueden leer el elemento; en cambio, un elemento bloqueado para escribir posee un **candado exclusivo**, porque una sola transacción posee de manera exclusiva el candado del elemento.

bloquear_elemento(X):

B: si CANDADO(X)=0 (* el elemento está desbloqueado *)
entonces CANDADO(X) $\leftarrow$ 1 (* bloquear el elemento *)
de otro modo comenzar

esperar (hasta que CANDADO(X)=0 y
el gestor de bloqueo despierte la transacción);
ir a **B**
fin;

desbloquear_elemento(X):

CANDADO(X) $\leftarrow$ 0 (* desbloquear el elemento *);
si alguna transacción está esperando
entonces despertar una de las transacciones que esperan;

Figura 18.1 Operaciones de bloquear y desbloquear para candados binarios.

Un método simple, aunque no del todo general, para implementar las tres operaciones anteriores con un candado de modo múltiple es mantenerse al tanto del número de transacciones que poseen un candado compartido sobre un elemento. Cada candado puede ser un registro con tres campos: <nombre del elemento de información, CANDADO, núm. de lecturas>. El valor de CANDADO es un código apropiado para representar uno de los tres estados: bloqueado para leer, bloqueado para escribir o desbloqueado. Una vez más, para ahorrar espacio, el sistema sólo necesita mantener en la tabla de bloqueo registros de candado para los elementos bloqueados. Las tres operaciones bloquear lectura(X), bloquear escritura(X) y desbloquear(X) se describen en la figura 18.2. Como antes, todas estas operaciones deben considerarse indivisibles; no debe permitirse la intercalación una vez que una de las operaciones se haya iniciado hasta que la operación termine o la transacción se coloque en una cola de espera del elemento.

bloquear_lectura(X):

B: si CANDADO(X)="desbloqueado"

entonces comenzar CANDADO(X) $\leftarrow$ "bloqueado para leer";
núm. de lecturas(X) $\leftarrow$ 1

fin

de otro modo si CANDADO(X)="bloqueado para leer"

entonces núm. de lecturas(X) $\leftarrow$ núm. de lecturas(X) + 1
de otro modo comenzar esperar (hasta que CANDADO(X)="desbloqueado"
y el gestor de bloqueo despierte la transacción);
ir a **B**
fin;

bloquear_escritura(X):

B: si CANDADO(X)="desbloqueado"

entonces CANDADO(X) $\leftarrow$ "bloqueado para escribir"
de otro modo comenzar

esperar (hasta que CANDADO(X)="desbloqueado" y
el gestor de bloqueo despierte la transacción);
ir a **B**
fin;

desbloquear(X):

si CANDADO(X)="bloqueado para escribir"

entonces comenzar CANDADO(X) $\leftarrow$ "desbloqueado";
despertar una de las transacciones que esperan, si las hay
fin

de otro modo si CANDADO(X)="bloqueado para leer"

entonces comenzar
núm. de lecturas(X) $\leftarrow$ núm. de lecturas(X) - 1;
si núm. de lecturas(X)=0
entonces comenzar CANDADO(X) $\leftarrow$ "desbloqueado";
despertar una de las transacciones que esperan, si las hay
fin;
fin;

Figura 18.2 Operaciones de bloqueo y desbloqueo para candados de dos modos (leer-escribir o compartido-exclusivo).

Cuando usamos el esquema de bloqueo de modo múltiple, el sistema debe hacer cumplir las siguientes reglas:

1. Una transacción T debe emitir la operación `bloquear_lectura(X)` o `bloquear_escritura(X)` antes de que se realice cualquier operación `leer_elemento(X)` de T .
2. Una transacción T debe emitir la operación `bloquear_escritura(X)` antes de que se realice cualquier operación `escribir_elemento(X)` de T .
3. Una transacción T debe emitir la operación `desbloquear(X)` una vez que se hayan completado todas las operaciones `leer_elemento(X)` y `escribir_elemento(X)` de T .
4. Una transacción T no emitirá una operación `bloquear_lectura(X)` si ya posee un candado de lectura (compartido) o de escritura (exclusivo) para el elemento X . Esta regla puede permitir excepciones, como veremos más adelante.
5. Una transacción T no emitirá una operación `bloquear_escritura(X)` si ya posee un candado de lectura (compartido) o de escritura (exclusivo) para el elemento X . Esta regla puede permitir excepciones, como veremos en breve.
6. Una transacción T no emitirá una operación `desbloquear(X)` a menos que ya posea un candado de lectura (compartido) o de escritura (exclusivo) para el elemento X .

En ocasiones es deseable permitir excepciones a las condiciones 4 y 5 de la lista anterior. Por ejemplo, es posible que una transacción T emita un `bloquear_lectura(X)` y más adelante **promueva** el bloqueo emitiendo una operación `bloquear_escritura(X)`. Si T es la única transacción con bloqueo de lectura para X en el momento en que emite la operación `bloquear_escritura(X)`, es posible promover el bloqueo. Una transacción T también puede emitir un `bloquear_escritura(X)` y después **degradar** el candado emitiendo una operación `bloquear_lectura(X)`. Si permitimos la promoción y la degradación de candados, deberemos incluir identificadores de transacciones en la estructura de registro de cada candado para almacenar la información de cuál transacción posee candados para el elemento, y deberemos hacer modificaciones apropiadas a las descripciones de las operaciones `bloquear_lectura(X)` y `bloquear_escritura(X)` en la figura 18.2. Dejamos esto como ejercicio para el lector.

El empleo de candados binarios o de modo múltiple en las transacciones, según la descripción anterior, *no garantiza la seriabilidad* de los planes en los que participan las transacciones. La figura 18.3 muestra un ejemplo en el que se siguen las reglas de bloqueo anteriores pero que puede resultar un plan no serializable. Esto se debe a que en la figura 18.3(a) los elementos Y en T_1 y X en T_2 se *desbloquearon antes de tiempo*. Esto permite que ocurra un plan como el que se ilustra en la figura 18.3(c); éste no es un plan serializable y por ello da resultados incorrectos. Para garantizar la seriabilidad debemos seguir un *protocolo adicional* sobre la ubicación de las operaciones de bloqueo y desbloqueo dentro de las transacciones. El protocolo mejor conocido, el bloqueo de dos fases, se describe en la siguiente sección.

18.1.2 Cómo garantizar la seriabilidad con el bloqueo de dos fases

Se dice que una transacción sigue el **protocolo de bloqueo de dos fases**¹ si todas las operaciones de bloqueo (`bloquear_lectura`, `bloquear_escritura`) preceden a la *primera* operación

¹Este nada tiene que ver con el protocolo de confirmación de dos fases para la recuperación en bases de datos distribuidas (véase el Cap. 19).

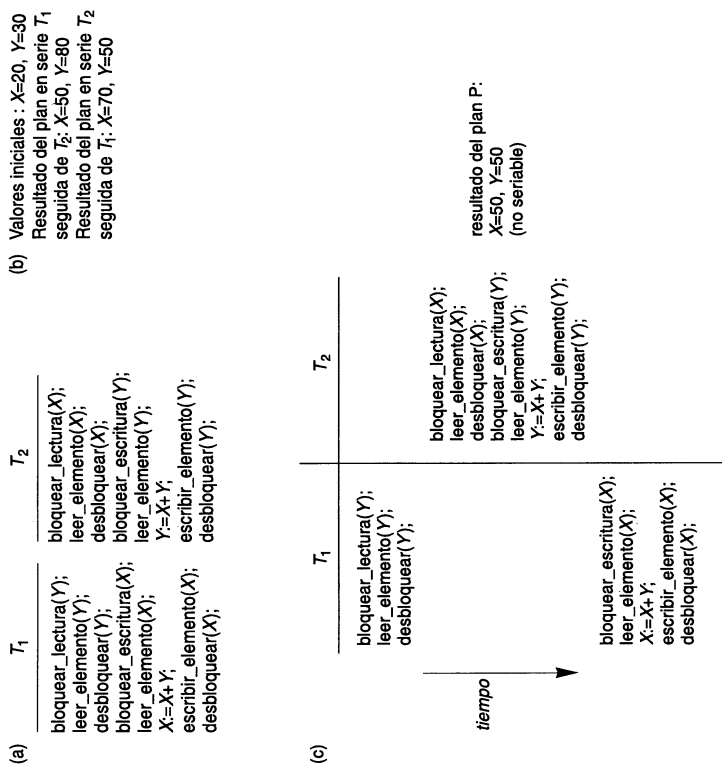


Figura 18.3 Transacciones que no obedecen el bloqueo de dos fases. (a) Dos transacciones T_1 y T_2 . (b) Resultados de posibles planes en serie de T_1 y T_2 . (c) Un plan P no serializable que usa candados.

de desbloqueo en la transacción. Una transacción así puede dividirse en dos fases: una **fase de expansión** (o de **crecimiento**), durante la cual se pueden adquirir nuevos candados sobre elementos pero no se puede liberar ninguno; y una **fase de contracción**, durante la cual se pueden liberar los candados existentes, pero no se pueden adquirir nuevos candados. Si se permite promover los candados, esta definición no cambia. Pero, si también se permite la degradación de candados, la definición debe alterarse ligeramente, porque *todas las degradaciones deben efectuarse en la fase de contracción*. Así, una operación `bloquear_lectura(X)` que degrada un candado de escritura que ya se tenía sobre X sólo puede aparecer en la fase de contracción de la transacción.

Las transacciones T_1 y T_2 de la figura 18.3(a) no se ajustan al protocolo de bloqueo de dos fases porque la operación `bloquear_escritura(X)` sigue a la operación `desbloquear(Y)` en T_1 , e igualmente la operación `bloquear_escritura(Y)` sigue a la operación `desbloquear(X)` en T_2 . Si imponemos el bloqueo de dos fases, las transacciones se pueden reescribir como T_1' y T_2' (Fig. 18.4). Ahora bien, el plan que se muestra en la figura 18.3(c) no está permitido para T_1' y T_2' según las reglas de bloqueo descritas

T_1	T_2
bloquear_lectura(Y);	bloquear_lectura(X);
leer_elemento(Y);	leer_elemento(X);
bloquear_escritura(X);	bloquear_escritura(Y);
desbloquear(Y);	desbloquear(X);
leer_elemento(X);	leer_elemento(Y);
$X = X + Y;$	$Y = X + Y;$
escribir_elemento(X);	escribir_elemento(Y);
desbloquear(X);	desbloquear(Y);

Figura 18.4 Las transacciones T_1 y T_2 , que son las mismas que T_1 y T_2 de la figura 18.3, pero siguiendo el protocolo de bloqueo de dos fases.

en la sección 18.1.1. Esto se debe a que T_1 emitirá su bloquear_escritura(X) antes de desbloquear el elemento Y; en consecuencia, cuando T_2 emita su bloquear_lectura(X), se verá obligada a esperar hasta que T_1 emita su desbloquear(X) en el plan.

Puede demostrarse que, si todas las transacciones de un plan siguen el protocolo de bloqueo de dos fases, el plan es serializable, así que ya no es necesario comprobar la serializabilidad de los planes. El mecanismo de bloqueo, al imponer las reglas de bloqueo de dos fases, también impone la serializabilidad.

El bloqueo de dos fases puede limitar el grado de concurrencia que pueda haber en un plan. La razón es que una transacción T tal vez no pueda liberar un elemento X cuando termine de usarlo si T debe bloquear un elemento adicional Y más adelante; o bien, T deberá bloquear el elemento adicional Y antes de necesitarlo para poder liberar X . Así pues, X debe permanecer bloqueado por T hasta que ésta haya bloqueado todos los elementos que va a necesitar; sólo entonces podrá T liberar X . Mientras tanto, otra transacción que desee tener acceso a X podría verse obligada a esperar, aunque T ya haya terminado de usarlo; o bien, si Y queda bloqueado antes de que se le necesite realmente, otra transacción que desee tener acceso a él se verá obligada a esperar aunque T todavía no esté usando Y . Éste es el precio de garantizar la serializabilidad de todos los planes sin tener que examinar los planes mismos.

Bloqueo de dos fases básico, conservador y estricto. El bloqueo de dos fases (B2F) tiene algunas variaciones. La técnica que acabamos de describir se denomina B2F básico. Una variación conocida como B2F conservador (o B2F estático) requiere que una transacción bloquee todos los elementos a los que tendrá acceso antes de comenzar a ejecutarse, **predeclarando** su conjunto de lectura y su conjunto de escritura. Recuerde que el conjunto de lectura de una transacción es el conjunto de todos los elementos que lee, y que su conjunto de escritura es el conjunto de todos los elementos que escribe. Si no es posible bloquear cualquiera de los elementos predeclarados necesarios, la transacción no bloqueará ningún elemento; en vez de ello, esperará hasta que todos los elementos estén disponibles para bloquearlos. El B2F conservador es un protocolo libre de bloqueo mortal, como veremos en la sección 18.1.3 cuando hablemos del problema del bloqueo mortal.

En la práctica, la variación más utilizada de B2F es el B2F estricto, que garantiza planes estrictos (véase la Sec. 17.4.2). En él, una transacción T no libera ninguno de sus candados antes de confirmarse o de abortar; por tanto, ninguna otra transacción puede leer

o escribir un elemento escrito por T a menos que T ya se haya confirmado, dando lugar a un plan estricto en cuanto a recuperabilidad. Observe la diferencia entre el B2F conservador y el estricto; el primero debe bloquear todos sus elementos antes de iniciarse, en tanto que el segundo no desbloquea ninguno de sus elementos sino hasta después de terminar (sea confirmando o abortando). El B2F estricto no está libre de bloqueo mortal a menos que se combine con el B2F conservador.

Aunque el bloqueo de dos fases garantiza la serializabilidad, el empleo de candados puede provocar dos problemas adicionales: bloqueo mortal y espera indefinida. Hablaremos de esos problemas y de su resolución en la siguiente sección.

18.1.3 Resolución del bloqueo mortal y de la espera indefinida

Hay **bloqueo mortal** cuando dos transacciones están esperando que la otra libere su candado sobre un elemento. En la figura 18.5(a) se muestra un ejemplo sencillo, donde las dos transacciones T_1 y T_2 se están bloqueando mutuamente en un plan parcial; T_1 está esperando que T_2 libere el elemento X , y T_2 está esperando que T_1 libere el elemento Y . Mientras esto sucede, ninguna de las dos puede proceder a desbloquear el elemento que la otra está esperando, y las demás transacciones no pueden tener acceso ni al elemento X ni a Y . También puede haber un bloqueo mortal en el que intervienen más de dos transacciones, como habremos de ver.

Una forma de evitar el bloqueo mortal es con un **protocolo de prevención del bloqueo mortal**. Un protocolo de este tipo, que se usa en el bloqueo de dos fases conservador, requiere que toda transacción bloquee por adelantado todos los elementos que vaya a necesitar; si no es posible obtener alguno de los elementos, no se bloqueará ninguno de ellos, y la transacción tendrá que esperar y luego intentar otra vez bloquear todos los elementos que necesite. Es obvio que esta solución limita todavía más la concurrencia. Un segundo protocolo, que también limita la concurrencia, consiste en ordenar todos los elementos de la base de datos y asegurarse de que una transacción que necesite varios elementos los bloqueará según ese orden. Esto obliga al programador a conocer el orden elegido, lo cual no es muy práctico en el contexto de las bases de datos.

Se han propuesto otros varios esquemas para evitar el bloqueo mortal, los cuales dependen si una transacción implicada en una situación de posible bloqueo mortal debe verse obligada a esperar o se debe abortar o debe hacer que otra transacción aborte. Estas técnicas aplican el concepto de **marca de tiempo de transacción**, $MT(T)$, que es un identificador único asignado a cada transacción. Las marcas de tiempo se ordenan con base en el orden en que se inician las transacciones; así pues, si la transacción T_1 se inicia antes que la transacción T_2 , entonces $MT(T_1) < MT(T_2)$. Advuértase que la transacción más antigua tiene el valor de marca de tiempo más pequeño. Dos esquemas que evitan el bloqueo mortal se denominan esperar-morir y herir-esperar. Supongamos que la transacción T_1 intenta bloquear un elemento X , pero no puede hacerlo porque X está bloqueado por alguna otra transacción T_2 con un candado en conflicto. Las reglas que siguen estos esquemas son:

- **esperar-morir:** si $MT(T_1) < MT(T_2)$ (T_1 es más antigua que T_2), entonces T_1 puede esperar

en caso contrario, se aborta T_1 (T_1 muere) y se le reinicia posteriormente con la misma marca de tiempo

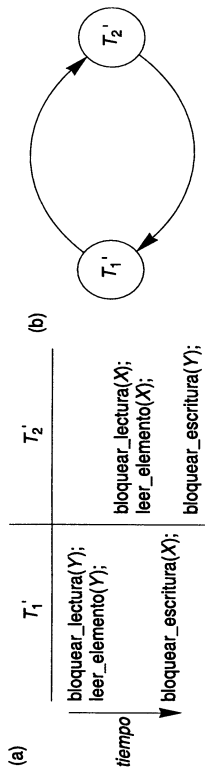


Figura 18.5 El problema del bloqueo mortal. (a) Plan parcial de T_1 y T_2 que está en estado de bloqueo mortal. (b) Grafo de espera del plan parcial de la figura 18.5(a).

- **herir-esperar:** si $MT(T_i) < MT(T_j)$ (T_i es más antigua que T_j) entonces se aborta T_j (T_j hiera a T_i) y se le reinicia posteriormente con la misma marca de tiempo
en caso contrario, T_i puede esperar

En esperar-morir, una transacción más antigua puede esperar a que termine una más reciente, pero una transacción más reciente que solicite un elemento bloqueado por una más antigua se abortará y reiniciará. El enfoque herir-esperar hace lo contrario: una transacción más reciente puede esperar a que termine una más antigua, pero una transacción más antigua que solicite un elemento bloqueado por una más reciente *desalojará* a la más reciente abortándola. En última instancia, ambos esquemas hacen que aborte la *más reciente* de las dos transacciones que *podrían intervenir* en un bloqueo mortal, y puede demostrarse que estas dos técnicas están libres de bloqueo mortal. Sin embargo, ambas hacen que algunas transacciones se aborten y reinicien, a pesar de que tal vez *nunca provoquen realmente un bloqueo mortal*. Podría presentarse otro problema con el enfoque de esperar-morir, donde la transacción T_1 puede abortar y reiniciarse varias veces seguidas porque una transacción T_j más antigua sigue bloqueando el elemento que T_1 necesita.

Hay otro grupo de protocolos para evitar el bloqueo mortal que no requieren marcas de tiempo. Entre ellos están los algoritmos de no esperar (NW: *no waiting*) y de espera cautelosa (CW: *cautious waiting*). En el algoritmo de **no esperar**, si una transacción no puede obtener un candado, se aborta de inmediato y se reinicia después de cierto lapso sin comprobar si realmente ocurrirá un bloqueo mortal o no. En vista de que este esquema puede hacer que algunas transacciones se aborten y reinicien innecesariamente, se propuso el enfoque de **espera cautelosa** para intentar reducir el número de abortos/reinicios innecesarios. Supongamos que la transacción T_1 intenta bloquear un elemento X pero no puede hacerlo porque X está bloqueado por alguna otra transacción T_j con un candado en conflicto. Las reglas de la espera cautelosa son las siguientes:

- **espera cautelosa:** si T_j no está detenida (no está esperando algún otro elemento bloqueado) entonces se detiene T_j , la cual puede esperar de lo contrario, se aborta T_i

Puede demostrarse que la espera cautelosa está libre de bloqueos mortales, considerando los momentos en que una transacción T queda detenida, $d(T)$. Si las dos transacciones

T_i y T_j antes mencionadas quedan detenidas, y T_i está esperando a que T_j termine, entonces $d(T_i) < d(T_j)$, ya que una transacción sólo puede esperar a que termine otra cuando no está detenida. Por tanto, los momentos de detención constituyen un ordenamiento total sobre todas las transacciones detenidas, y no puede ocurrir ningún ciclo que provoque un bloqueo mortal.

Otro esquema para evitar el bloqueo mortal implica usar **tiempos predefinidos**. Si una transacción espera durante un lapso mayor que un tiempo predefinido por el sistema, éste supondrá que la transacción está en un bloqueo mortal y la abortará, sin importar si existe o no realmente una situación de bloqueo mortal.

Un segundo enfoque para resolver el problema es la **detección de bloqueo mortal**, en la que se verifica periódicamente si el sistema está en un estado de bloqueo mortal. Esta solución es atractiva si sabemos que va a haber poca interferencia entre las transacciones; es decir, si las diferentes transacciones casi nunca tendrán acceso a los mismos elementos al mismo tiempo. Esto puede suceder si las transacciones son cortas y si cada una bloquea apenas unos cuantos elementos, o si la carga de transacciones es ligera. Por otro lado, si las transacciones son largas y cada una utiliza muchos elementos, o si la carga de transacciones es bastante pesada, es más conveniente usar un esquema de prevención del bloqueo mortal.

Una forma sencilla de detectar un estado de bloqueo mortal es construir un **grafo de espera**. Se crea un nodo en el grafo de espera por cada transacción que se está ejecutando actualmente en el plan. Siempre que una transacción T_i está esperando para bloquear un elemento X que está bloqueado momentáneamente por una transacción T_j , se crea una arista dirigida ($T_i \rightarrow T_j$). Cuando T_j libera el o los candados sobre los elementos que T_i estaba esperando, la arista dirigida se borra del grafo de espera. Tenemos un estado de bloqueo mortal si y sólo si el grafo de espera tiene un ciclo. Un problema de este enfoque es **cuándo** determinar que el sistema deba verificar si hay bloqueo mortal. Se puede usar criterios como el número de transacciones que se están ejecutando o el tiempo que varias transacciones han estado en espera de bloquear elementos, para determinar si ya es oportuno que el sistema compruebe si hay o no bloqueo mortal. La figura 18.5(b) muestra el grafo de espera del plan parcial de la figura 18.5(a). Si descubrimos que el sistema está en un estado de bloqueo mortal, será preciso abortar algunas de las transacciones que lo están provocando. La elección de las transacciones por abortar se conoce como **selección de víctimas**. En general, el algoritmo para seleccionar víctimas debe evitar elegir transacciones que se hayan estado ejecutando durante mucho tiempo y que hayan efectuado muchas actualizaciones, y debe procurar seleccionar más bien transacciones que no hayan realizado muchos cambios o que participen en más de un ciclo de bloqueo mortal en el grafo de espera. Un problema que puede ocurrir es el denominado **reinicio cíclico**, en el que una transacción se aborta y reinicia sólo para caer en otro bloqueo mortal. El algoritmo de selección de víctimas puede conferir prioridades más altas a las transacciones que se han abortado varias veces, a fin de que no se escojan como víctimas una y otra vez.

Aún hay otro problema que puede presentarse cuando usamos bloqueo: la **espera indefinida**. Una transacción se encuentra en un estado de espera indefinida si no puede continuar durante un periodo indeterminado mientras otras transacciones del sistema continúan con normalidad. Esto puede ocurrir si el esquema de espera de elementos bloqueados es injusto y confiere a algunas transacciones mayor prioridad que a otras. La solución estándar para la espera indefinida es contar con un esquema de espera justo. Un esquema de este tipo utiliza una cola de **primero que llega, primero que se atiende**; las transacciones pueden bloquear un elemento en el orden en que originalmente solicitaron el bloqueo de

dicho elemento. Otro esquema permite que algunas transacciones tengan mayor prioridad que otras pero aumenta la prioridad de una transacción a medida que ésta espera, hasta que llegue el momento en que tenga la prioridad más alta de todas y continúe. Un problema similar a la espera indefinida, llamado **inanición**, puede ocurrir debido a los algoritmos que resuelven el bloqueo mortal. Se presenta si los algoritmos seleccionan la misma transacción como víctima una y otra vez, haciendo que aborte y que nunca termine su ejecución. Los esquemas esperar-morir y herir-esperar que vimos antes evitan la inanición.

18.2 Control de concurrencia basado en ordenamiento por marca de tiempo^{*}

El empleo de candados, en combinación con el protocolo de bloqueo de dos fases, nos permite garantizar que los planes sean serializables. El orden de las transacciones en el plan en serie equivalente se basa en el orden en que las transacciones en ejecución bloquean los elementos que requieren. Si una transacción necesita un elemento que ya está bloqueado, puede verse obligada a esperar hasta que se libere dicho elemento. Un enfoque distinto que garantiza la serializabilidad implica el uso de marcas de tiempo de las transacciones para ordenar la ejecución de las transacciones según un plan en serie equivalente. En la sección 18.2.1 definiremos las marcas de tiempo; luego, en la sección 18.2.2, analizaremos cómo se impone la serializabilidad mediante el ordenamiento de las transacciones con base en sus marcas de tiempo.

18.2.1 Marcas de tiempo

Una **marca de tiempo** es un identificador único que el SGBD crea para identificar una transacción. Por lo regular, los valores de marca de tiempo se asignan en el orden en que las transacciones se introducen en el sistema, por lo que una marca de tiempo puede considerarse como el *tiempo de inicio de una transacción*. Nos referiremos a la marca de tiempo de una transacción T como $MT(T)$. Las técnicas para el control de concurrencia basadas en marcas de tiempo no usan bloqueos; por tanto, *no puede* haber bloqueos mortales.

Las marcas de tiempo se pueden generar de diversas maneras. Una posibilidad consiste en usar un contador que se incrementa cada vez que su valor se asigna a una transacción. Las marcas de tiempo de las transacciones están numeradas 1, 2, 3, ... en este esquema. Un contador de computador tiene un valor máximo finito, por lo que el sistema deberá restablecer periódicamente a cero el contador cuando haya un lapso corto en el que ninguna transacción se esté ejecutando. Otra forma de implementar las marcas de tiempo consiste en emplear el valor actual del reloj del sistema y asegurar que no se generen dos valores de marca de tiempo antes de que el reloj cambie.

18.2.2 El algoritmo de ordenamiento por marca de tiempo

Este esquema se basa en la idea de ordenar las transacciones con base en sus marcas de tiempo. Así, un plan en el que intervengan las transacciones será serializable, y el plan en serie equivalente tendrá las transacciones en orden según sus valores de marca de tiempo. Esto se denomina **ordenamiento por marca de tiempo (OMT)**. Observe la diferencia respecto al

bloqueo de dos fases, en el cual un plan es serializable si es equivalente a *algún* plan en serie permitido por los protocolos de bloqueo; en el ordenamiento por marca de tiempo, en cambio, el plan es equivalente al orden en serie *específico* que corresponde al orden de las marcas de tiempo de las transacciones. El algoritmo debe asegurar que, para cada elemento utilizado por más de una transacción en el plan, el orden en que se tiene acceso al elemento no viole la serializabilidad del plan. Para lograr esto, el algoritmo de OMT **básico** asocia a cada elemento X de la base de datos dos valores de marca de tiempo (**MT**):

1. **MT_lectura(X)**: La **marca de tiempo de lectura** del elemento X ; ésta es la más grande de todas las marcas de tiempo de las transacciones que han leído con éxito el elemento X .
2. **MT_escritura(X)**: La **marca de tiempo de escritura** del elemento X ; ésta es la más grande de todas las marcas de tiempo de las transacciones que han escrito con éxito el elemento X .

Siempre que una transacción T intenta emitir una operación leer_elemento(X) o escribir_elemento(X), el algoritmo de OMT básico compara la marca de tiempo de T con las marcas de tiempo de lectura y de escritura de X para asegurar que no se viole el orden de ejecución por marca de tiempo de las transacciones. Si la operación viola dicho orden, la transacción T violará el plan en serie equivalente, así que se aborta. Entonces se vuelve a introducir T al sistema como transacción nueva con una *nueva* marca de tiempo. Si T aborta y se revierte, cualquier transacción T_1 que pueda haber usado un valor escrito por T deberá revertirse también. De manera similar, cualquier transacción T_2 que pueda haber usado un valor escrito por T_1 deberá revertir también, y así sucesivamente. Este efecto se conoce como **reversión en cascada** y es uno de los problemas asociados al OMT básico, ya que los planes producidos no son recuperables. El algoritmo para el control de concurrencia debe verificar si se viola el ordenamiento por marca de tiempo de las transacciones en estos dos casos:

1. La transacción T emite una operación escribir_elemento(X):
 - a. Si $MT_lectura(X) > MT(T)$ o $MT_escritura(X) > MT(T)$, entonces abortar y revertir T y rechazar la operación. Esto debe hacerse porque alguna transacción con una marca de tiempo mayor que $MT(T)$ —y por tanto posterior a T en el ordenamiento por marca de tiempo— ya leyó o escribió el valor del elemento X antes de que T tuviera oportunidad de escribir X , violándose así el ordenamiento por marca de tiempo.
 - b. Si la situación de la parte a no se presenta, se ejecuta la operación escribir_elemento(X) de T y se asigna $MT(T)$ a $MT_escritura(X)$.
2. La transacción T emite una operación leer_elemento(X):
 - a. Si $MT_escritura(X) > MT(T)$, se debe abortar y revertir T y rechazar la operación. Esto debe hacerse porque alguna transacción con marca de tiempo mayor que $MT(T)$ —y por tanto *posterior* a T en el ordenamiento por marca de tiempo— ya escribió el valor de X antes de que T tuviera oportunidad de leer X .
 - b. Si $MT_escritura(X) \leq MT(T)$, se ejecuta la operación leer_elemento(X) de T y se asigna a $MT_lectura(X)$ el mayor de los dos valores: $MT(T)$ y $MT_lectura(X)$ actual.

Así pues, el algoritmo OMT básico comprueba si dos operaciones en conflicto ocurren en el orden incorrecto, y rechaza la más reciente de las dos abortando la transacción que la emitió. De este modo, se garantiza que los planes producidos por el OMT básico sean seriamente posibles por conflictos. Una modificación del algoritmo, conocida como **regla de escritura de Thomas**, no impone la seriabilidad por conflictos, pero rechaza menos operaciones de escritura, modificando las verificaciones de la operación escribir elemento(X) como sigue:

- Si $MT_lectura(X) > MT(T)$, abortar y revertir T y rechazar la operación.
- Si $MT_escritura(X) > MT(T)$, no ejecutar la operación de escritura pero continuar procesando. Esto se debe a que alguna transacción con marca de tiempo mayor que $MT(T)$ —y por tanto posterior a T en el ordenamiento por marca de tiempo— ya escribió el valor de X . Por ello, debemos ignorar la operación escribir elemento(X) de T porque ya es obsoleta. Advirtiéndose que cualquier conflicto originado por esta situación se detectaría por el caso a.
- Si no ocurre ninguna de las situaciones de las partes a y b, se ejecuta la operación escribir elemento(X) de T y se asigna $MT(T)$ a $MT_escritura(X)$.

El protocolo de ordenamiento por marca de tiempo, al igual que el de bloqueo de dos fases, garantiza la seriabilidad de los planes. Sin embargo, algunos planes que son posibles con uno de estos protocolos no lo son con el otro. Así pues, ninguno de dichos protocolos permite todos los planes seriamente posibles. Como ya se mencionó, no hay bloqueo mortal con el ordenamiento por marca de tiempo; sin embargo, el reinicio cíclico (y por tanto la inanición) puede ocurrir si una transacción se aborta y reinicia continuamente.

Como se dijo antes, el algoritmo de OMT básico impone la seriabilidad por conflictos, pero no garantiza que los planes sean recuperables; por tanto, tampoco garantiza planes sin cascada ni estrictos (véase la Sec. 17.4.2). Una variación del OMT básico llamado **OMT estricto** garantiza que los planes sean tanto estrictos como seriables (por conflictos). En esta variación, una transacción T que emite una operación leer elemento(X) o escribir elemento(X) tal que $MT(T) > MT_escritura(X)$ sufre un retraso de su operación de lectura o escritura hasta que la transacción T que escribió el valor de X (de modo que $MT(T') = MT_escritura(X)$) se haya confirmado o abortado. Para implementar este algoritmo es necesario simular el bloqueo de un elemento X , escrito por la transacción T' , hasta que T' se haya confirmado o abortado. Este algoritmo no provoca bloqueo mortal, porque T espera a que termine T' sólo si $MT(T) > MT(T')$.

18.3 Técnicas para el control de concurrencia de multiversión*

Otros protocolos para controlar la concurrencia conservan los valores antiguos de un elemento de información cuando éste se actualiza. Éstos se conocen como técnicas de **control de concurrencia de multiversión**, porque se mantienen varias versiones (valores) de un elemento. Cuando una transacción requiere acceso a un elemento, se elige una versión apropiada para mantener la seriabilidad del plan que se está ejecutando, si es posible. La idea consiste en que algunas operaciones de lectura que serían rechazadas si se usaran otras técnicas se pueden aceptar leyendo una versión anterior del elemento a fin de mantener la seriabilidad. Cuando una transacción escribe un elemento, escribe una nueva versión y se conserva la versión anterior de dicho elemento. En general, los algoritmos para el control

de concurrencia de multiversión utilizan el concepto de seriabilidad por vistas, no el de seriabilidad por conflictos.

Una desventaja obvia de las técnicas de multiversión es que se requiere más almacenamiento para mantener múltiples versiones de los elementos de la base de datos. Sin embargo, es posible que de todos modos sea necesario mantener versiones anteriores; por ejemplo, con fines de recuperación. Por añadidura, algunas aplicaciones de bases de datos requieren la conservación de versiones anteriores para mantener una historia de cómo evolucionan los valores de los elementos de información. El caso extremo es una *base de datos temporal*, que sigue la pista a todos los cambios y los momentos en que ocurrieron. En tales casos, no hay gasto adicional por las técnicas de multiversión, ya que de todos modos se mantienen versiones anteriores.

Se han propuesto varios esquemas para el control de concurrencia de multiversión. Aquí examinaremos dos de ellos, uno basado en ordenamiento por marca de tiempo y el otro basado en bloqueo de dos fases.

18.3.1 Técnica de multiversión basada en ordenamiento por marca de tiempo

En esta técnica de multiversión, el sistema conserva varias versiones, $X_1, X_2, \dots, X_i$ de cada elemento de información X . Para cada versión, se conservan el valor de la versión, X_i , y estas dos marcas de tiempo:

1. $MT_lectura(X_i)$: La **marca de tiempo de lectura** de X_i ; ésta es la más grande entre todas las marcas de tiempo de las transacciones que han leído con éxito la versión X_i .
2. $MT_escritura(X_i)$: La **marca de tiempo de escritura** de X_i ; ésta es la marca de tiempo de la transacción que escribió el valor de la versión X_i .

Siempre que se permite a una transacción T ejecutar una operación escribir elemento(X), se crea una nueva versión X_{i+1} del elemento X , asignando $MT(T)$ tanto a $MT_escritura(X_{i+1})$ como a $MT_lectura(X_{i+1})$. De igual manera, cada vez que se permite a una transacción T leer el valor de la versión X_i , se asigna a $MT_lectura(X_i)$ el mayor de estos dos valores: $MT_lectura(X_i)$ o $MT(T)$.

Para asegurar la seriabilidad, usamos dos reglas para controlar la lectura y escritura de elementos de información:

1. Si la transacción T emite una operación escribir elemento(X), y la versión i de X tiene la $MT_escritura(X_i)$ más alta de todas las versiones de X que también es menor que $MT(T)$ o igual a ella, y $MT(T) < MT_lectura(X_i)$, entonces se debe abortar y revertir la transacción T ; en caso contrario, se crea una nueva versión X_j de X con $MT_lectura(X_j) = MT_escritura(X_j) = MT(T)$.
2. Si la transacción T emite una operación leer elemento(X), se busca la versión i de X que tiene la $MT_escritura(X_i)$ más alta de todas las versiones de X que también es menor que $MT(T)$ o igual a ella; luego se devuelve el valor de X_i a la transacción T y se asigna a $MT_lectura(X_i)$ el más alto de estos dos valores: $MT(T)$ y el $MT_lectura(X_i)$ actual.

En el caso 1, la transacción T podría abortar y revertirse. Esto sucede si T está intentando escribir una versión de X que debería haber leído otra transacción T' cuya marca de

tiempo es MT_lectura(X); sin embargo, T_i ya leyó la versión X_i escrita por la transacción cuya marca de tiempo es MT_escritura(X_i). Si se presenta este conflicto, T_i se revertirá; si no, se creará una nueva versión de X_i escrita por la transacción T. Cabe señalar que, si T_i se revierte, puede haber reversión en cascada. Por ello, a fin de asegurar la recuperabilidad, no se permite que se confirme una transacción T antes de que se hayan confirmado todas las transacciones que hayan escrito versiones que T haya leído.

18.3.2 Bloqueo de dos fases de multiversión

En este esquema, un elemento tiene tres modos de bloqueo: lectura, escritura y certificación. Así, el estado de un elemento X puede ser “bloqueado para lectura”, “bloqueado para escritura”, “bloqueado para certificación” o “desbloqueado”. En el esquema de bloqueo estándar que sólo incluye candados de lectura y de escritura (véase la Sec. 18.1.1), un candado para escritura es un candado exclusivo. Podemos describir la relación entre los candados de lectura y de escritura en el esquema estándar mediante la **tabla de compatibilidad de candados** que aparece en la figura 18.6(a). Una entrada sí significa que, si una transacción T posee para el elemento X el tipo de candado especificado en la cabecera de la columna, y si la transacción T_i solicita para el mismo elemento el tipo de candado especificado en la cabecera de la fila, entonces T_i puede obtener el *candado* porque los modos de bloqueo son compatibles. Por otro lado, una entrada no en la tabla indica que los candados no son compatibles, y T_i debe esperar a que T libere el candado.

En el esquema de bloqueo estándar una vez que una transacción obtiene un candado de escritura para un elemento, ninguna otra transacción puede tener acceso a ese elemento. El bloqueo de dos fases de multiversión se basa en la idea de permitir que otras transacciones T_i lean un elemento X mientras una sola transacción T posea un candado de escritura para X. Esto se logra permitiendo **dos versiones** para cada elemento X, una de las cuales siempre debe haber sido escrita por una transacción confirmada. La segunda versión, X', se crea cuando una transacción T adquiere un candado de escritura para ese elemento. Otras transacciones pueden seguir leyendo la versión confirmada de X en tanto T posea el candado de escritura. Ahora la transacción T puede modificar el valor de X' según sus necesidades, sin afectar el valor de la versión confirmada de X. No obstante, una vez que T está lista para confirmarse, deberá obtener un **candado de certificación** para todos los elementos para los cuales posee actualmente candados de escritura, antes de poder confirmarse. El candado de certificación no es compatible con los candados de lectura, así que la transacción podría verse obligada a postergar su confirmación hasta que todos sus elementos bloqueados para escritura sean liberados por cualesquier transacciones lectoras. En este punto, se asigna a la versión confirmada X del elemento el valor de la versión X', la versión X' se desecha, y los candados de certificación se liberan. La tabla de compatibilidad de candados para este esquema se muestra en la figura 18.6(b).

En este esquema de bloqueo de dos fases de multiversión, las lecturas pueden realizarse al mismo tiempo que una operación de escritura, algo que no se permita en los esquemas de bloqueo de dos fases estándar. El costo radica en la posibilidad de que una transacción deba postergar su confirmación hasta obtener candados de certificación exclusivos para todos los elementos que actualizó. Puede demostrarse que este esquema evita los abortos en cascada, ya que sólo se permite que las demostraciones lean la versión de X escrita por una transacción confirmada. Sin embargo, sí puede haber bloqueo mortal si se permite la conversión de un candado de lectura en un candado de escritura, y esto debe resolverse con variaciones de las técnicas que vimos en la sección 18.1.3.

(a)

	Lectura	Escritura
Lectura	sí	no
Escritura	no	no

(b)

	Lectura	Escritura	Certificación
Lectura	sí	sí	no
Escritura	sí	no	no
Certificación	no	no	no

Figura 18.6 Compatibilidad de candados. (a) Tabla de compatibilidad para el esquema de bloqueo estándar. (b) Tabla de compatibilidad para el esquema de bloqueo de dos fases de multiversión.

18.4 Técnicas para el control de concurrencia de validación (optimistas)*

En todas las técnicas para controlar la concurrencia que hemos visto hasta ahora, se realiza una cierta verificación *antes* de que pueda ejecutarse una operación de base de datos. En el bloqueo, por ejemplo, se comprueba si el elemento al que se desea tener acceso está bloqueado. En el ordenamiento por marca de tiempo, se verifica la marca de tiempo de la transacción comparándola con las marcas de tiempo de lectura y de escritura del elemento. Esta verificación representa un gasto extra durante la ejecución de las transacciones, y su efecto es hacerlas más lentas.

En las técnicas de control de concurrencia **optimista**, también llamadas técnicas de validación o de certificación, *no se efectúa verificación alguna* durante la ejecución de las transacciones. Varios métodos de control de concurrencia propuestos utilizan la técnica de validación. Aquí describiremos sólo un esquema, en el cual las actualizaciones incluidas en una transacción no se aplican directamente a los elementos de la base de datos sino hasta que la transacción llega a su fin. Mientras se ejecuta la transacción, todas las actualizaciones se aplican a *copias locales* de los elementos, que se mantienen para la transacción. Al final de la ejecución, una **fase de validación** comprueba si cualquiera de las actualizaciones viola la seriabilidad. El sistema debe mantener cierta información que necesita para la fase de validación. Si no se viola la seriabilidad, la transacción se confirma y la base de datos se actualiza a partir de las copias locales; en caso contrario, la transacción aborta y se reinicia posteriormente.

Este protocolo de control de concurrencia comprende tres fases:

1. **Fase de lectura:** Una transacción puede leer valores de elementos de información a partir de la base de datos. Sin embargo, las actualizaciones se aplican sólo

a copias locales de los elementos, las que se mantienen en el espacio de trabajo de la transacción.

2. **Fase de validación:** Se efectúa una verificación para asegurarse de que no se violará la seriability si las actualizaciones de la transacción se aplican a la base de datos.
3. **Fase de escritura:** Si la fase de validación se realiza con éxito, las actualizaciones de la transacción se aplican a la base de datos; si no, las actualizaciones se desechan y la transacción se reinicia.

El control de concurrencia optimista se basa en la idea de efectuar todas las actualizaciones de una vez; así pues, la ejecución de las transacciones se efectúa con un mínimo de gasto extra hasta llegar a la fase de validación. Si hay poca interferencia entre las transacciones, casi todas se validarán sin dificultad. Sin embargo, si hay mucha interferencia, se desecharán los resultados de muchas transacciones que se ejecutaron hasta su término, y éstas tendrán que reiniciarse posteriormente. En estas circunstancias, las técnicas optimistas no funcionan bien. Estas técnicas se llaman "optimistas" porque suponen que habrá poca interferencia y que no será necesario realizar la verificación durante la ejecución de las transacciones.

El protocolo optimista describe emplea marcas de tiempo de transacciones y además requiere que el sistema mantenga los conjuntos de escritura y de lectura de las transacciones. Por añadidura, es preciso asentar horas de inicio y horas de terminación para las tres fases de cada transacción. El **conjunto de escritura** de una transacción es el conjunto de elementos que escribe, y el **conjunto de lectura** es el conjunto de elementos que lee. En su fase de validación para la transacción T_i , el protocolo comprueba que T_i no interfiera ninguna transacción confirmada ni cualesquier otras transacciones que estén actualmente en su fase de validación. La fase de validación de T_i verifica que, para cada una de esas transacciones T_j , que está confirmada o en su fase de validación, se cumpla una de las siguientes condiciones:

1. La transacción T_j completa su fase de escritura antes de que T_i inicie su fase de lectura.
2. T_i inicia su fase de escritura después de que T_j complete su fase de escritura, y el conjunto de lectura de T_i no tiene elementos en común con el conjunto de escritura de T_j .
3. Ni el conjunto de lectura ni el conjunto de escritura de T_i tienen elementos en común con el conjunto de escritura de T_j , y T_j completa su fase de lectura antes de que T_i complete su fase de lectura.

Si se cumple cualquiera de estas tres condiciones, no habrá interferencia y T_i se validará con éxito. Si no se cumple ninguna de ellas, fallará la validación de la transacción T_i y ésta se abortará y reiniciará posteriormente porque pudo haber interferencias.

18.5 Granularidad de los datos

Todas las técnicas para controlar las concurrencias supusieron que la base de datos consistía en varios elementos. Puede decidirse que un elemento de la base de datos sea cualquiera de los siguientes:

- Un registro de la base de datos.
- Un valor de campo de un registro de la base de datos.
- Un bloque de disco.
- Un archivo completo.
- La base de datos completa.

Debemos considerar varias ventajas y desventajas al escoger el tamaño de los elementos de la base de datos. Analizaremos el tamaño de estos elementos en el contexto del bloque, aunque pueden presentarse argumentos similares para otras técnicas de control de concurrencia.

En primer lugar, observe que cuanto mayor sea el tamaño del elemento de información, menor será el grado de concurrencia permitido. Por ejemplo, si el tamaño del elemento es un bloque de disco, una transacción T que necesite bloquear un registro A deberá bloquear todo el bloque de disco X que contiene a A . Esto se debe a que un candado está asociado a todo el elemento de información X . Ahora bien, si otra transacción S desea bloquear un registro distinto B que por casualidad reside en el mismo bloque X con un modo de bloqueo en conflicto, se verá obligada a esperar hasta que la primera transacción libere el candado del bloque X . Si el tamaño del elemento de información fuera un solo registro, la transacción S podría continuar, ya que bloquearía un elemento diferente (el registro B) del bloqueado por T (el registro A). Por otro lado, cuanto menor sea el tamaño del elemento de información, más elementos habrá en la base de datos. Como cada elemento está asociado a un candado, el sistema tendrá un mayor número de candados activos que deberá manejar el gestor de bloqueo. Se efectuarán más operaciones de bloquear y desbloquear, dando lugar a un gasto extra más alto. Por añadidura, se requerirá más espacio de almacenamiento para la tabla de bloqueo. En el caso de las marcas de tiempo, se requiere almacenamiento para la $MT_lectura$ y $MT_escritura$ de cada elemento de información, y el gasto extra de manejar una gran cantidad de elementos es similar a aquel en que se incurre con el bloqueo.

A menudo se llama **granularidad de los elementos de información** al tamaño de dichos elementos. El término *granularidad fina* indica elementos de tamaño pequeño, en tanto que *granularidad gruesa* designa elementos grandes. Dadas las ventajas y desventajas anteriores, la pregunta obvia es: ¿cuál es el mejor tamaño para los elementos? La respuesta es que ello depende de los tipos de las transacciones implicadas. Si una transacción representativa tiene acceso a un número pequeño de registros, conviene hacer que la granularidad de los elementos sea un registro. Por otro lado, si una transacción suele tener acceso a muchos registros del mismo archivo, quizá sea mejor tener una granularidad de bloque o de archivo para que la transacción considere todos esos registros como un solo (o unos cuantos) elemento(s) de información.

En su mayoría, las técnicas para controlar la concurrencia tienen un tamaño de elemento de información uniforme. Sin embargo, se han propuesto algunas técnicas que permiten tamaños variables. En ellas, es posible cambiar el tamaño de los elementos de información a la granularidad más adecuada para las transacciones que se están ejecutando en el sistema.

18.6 Otras cuestiones de control de concurrencia

En esta sección analizaremos otros aspectos relacionados con el control de concurrencia. En la sección 18.6.1 hablaremos de los problemas asociados a la inserción y la eliminación de

registros y del llamado "problema del fantasma", que puede ocurrir cuando se insertan registros. Después, en la sección 18.6.2, trataremos problemas que pueden presentarse cuando una transacción envía datos a una terminal antes de confirmarse, y luego aborta.

18.6.1 Inserción, eliminación y registros fantasmas

Cuando se **inserta** un nuevo elemento de información en la base de datos, obviamente no es posible tener acceso a él antes de que se complete la operación de inserción. En un entorno de bloqueo, se puede crear un candado para ese elemento y ponerlo en modo exclusivo (candado de escritura); el candado puede liberarse en el mismo momento en que se liberarían otros candados de escritura, dependiendo del protocolo de control de concurrencia que se emplee. En el caso de un protocolo basado en marcas de tiempo, las marcas de tiempo de lectura y escritura del nuevo elemento serán la marca de tiempo de la transacción creadora.

Una operación de **eliminación** se aplica a un elemento de información ya existente. En los protocolos de bloqueo, también es preciso obtener un candado exclusivo (de escritura) para que la transacción pueda eliminar el elemento. Si se usa ordenamiento por marca de tiempo, el protocolo debe asegurarse de que ninguna transacción posterior haya leído o escrito el elemento antes de permitir que una transacción elimine dicho elemento.

Puede ocurrir el llamado **problema del fantasma** cuando un registro que está siendo insertado por alguna transacción T satisface una condición que deberá satisfacer un conjunto de registros a los que tenga acceso otra transacción T' . Por ejemplo, supongamos que la transacción T está insertando un nuevo registro EMPLEADO que pertenece al departamento número 5 (esto es, cuyo atributo ND es 5), y que la transacción T' está teniendo acceso a todos los registros EMPLEADO para los cuales $ND = 5$ a fin de sumar todos sus valores SALARIO (y así calcular el presupuesto del personal del departamento 5). Si el orden en serie equivalente es T seguida de T' , esta última deberá leer el nuevo registro EMPLEADO e incluir su SALARIO en el cálculo de la suma. Para el orden en serie equivalente T' seguida de T , el nuevo salario no deberá incluirse. Adviértase que, aun cuando las transacciones están en conflicto desde el punto de vista lógico, en el segundo caso no hay realmente ningún registro (elemento de información) en común entre las dos transacciones, ya que T' puede haber bloqueado todos los registros con $ND = 5$ antes de que T insertara el nuevo registro. Esto se debe a que el registro que causa el conflicto es un **registro fantasma** que ha aparecido repentinamente en la base de datos al ser insertado. Si otras operaciones de las dos transacciones están en conflicto, es posible que el protocolo de control de concurrencia no se percate del conflicto debido al registro fantasma.

Una solución al problema del registro fantasma es el llamado **bloqueo del índice**, que se puede usar junto con un protocolo de bloqueo de dos fases. Recuerde que en el capítulo 5 se dijo que un índice incluye entradas que tienen un valor de atributo, más un conjunto de apuntadores a todos los registros del archivo que tienen ese valor. Por ejemplo, un índice sobre el atributo ND del archivo EMPLEADO incluiría una entrada por cada valor distinto de ND, más un conjunto de apuntadores a todos los registros EMPLEADO con ese valor. Si la entrada del índice se bloquea antes de que se pueda tener acceso al registro mismo, será posible detectar el conflicto por el registro fantasma. La razón es que la transacción T' solicitaría un candado de lectura para la *entrada de índice* correspondiente a $ND = 5$ y T solicitaría un candado de escritura para esa misma entrada antes de poder solicitar

candados para los registros reales. Puesto que los bloqueos del índice están en conflicto, el problema del fantasma se detectaría. Una técnica más general, llamada **bloqueo de predicado**, bloquearía de manera similar el acceso a todos los registros que satisficieran un *predicado* (condición) *arbitrario*; sin embargo, no es fácil implementar los bloqueos de predicado de manera eficiente.

18.6.2 Transacciones interactivas

Otro problema surge cuando las transacciones interactivas leen entradas y escriben salidas en un dispositivo interactivo, como la pantalla de una terminal, antes de su confirmación. El problema es que un usuario puede introducir en una transacción T un valor de elemento de información basado en algún valor escrito en la pantalla por la transacción T' , que tal vez no se haya confirmado aún. El método de control de concurrencia del sistema no puede modelar la dependencia entre T y T' , ya que sólo se basa en la interacción entre el usuario y las dos transacciones.

Una estrategia para resolver este problema consiste en posponer las salidas de las transacciones a la pantalla hasta que hayan sido confirmadas.

18.7 Resumen

En este capítulo estudiamos técnicas de SGBD para el control de concurrencia. En la sección 18.1 vimos el protocolo de bloqueo de dos fases y algunas de sus variaciones. Presentamos los conceptos de candado compartido (de lectura) y exclusivo (de escritura), y mostramos cómo el bloqueo puede garantizar la seriabilidad cuando se usa junto con la regla de bloqueo de dos fases. Analizamos el protocolo básico de bloqueo de dos fases, así como el bloqueo de dos fases conservador y el estricto. También presentamos diversas técnicas para resolver el problema del bloqueo mortal, que puede presentarse con el bloqueo.

En la sección 18.2 examinamos el protocolo de ordenamiento por marca de tiempo, que asegura la seriabilidad con base en el orden de las marcas de tiempo de las transacciones. Las marcas de tiempo son identificadores de transacción únicos, generados por el sistema. Vimos la regla de escritura de Thomas, que mejora el rendimiento pero no garantiza la seriabilidad por conflictos. También presentamos el protocolo de ordenamiento por marca de tiempo estricto.

En la sección 18.3 tratamos dos protocolos de multiversión, que suponen que en la base de datos se pueden conservar versiones anteriores de los elementos de información. Una técnica llamada bloqueo de dos fases de multiversión supone que puede haber dos versiones de un elemento, e intenta aumentar la concurrencia haciendo compatibles los candados de lectura y de escritura (a expensas de introducir un modo de bloqueo de certificación adicional). También presentamos un protocolo de multiversión basado en el ordenamiento por marca de tiempo. En la sección 18.4 dimos un ejemplo de protocolo optimista, también conocido como protocolo de certificación o de validación. En la sección 18.5 hicimos un breve análisis de la granularidad de los elementos de información, y en la sección 18.6 presentamos el problema del fantasma y los problemas con las transacciones interactivas.

En el siguiente capítulo ofreceremos un panorama sobre las técnicas de recuperación.

Preguntas de repaso

- 18.1. ¿Qué es el protocolo de bloqueo de dos fases? ¿Cómo garantiza la seriabilidad?
- 18.2. Mencione algunas variaciones del protocolo de bloqueo de dos fases. ¿Por qué suele preferirse el bloqueo de dos fases estricto?
- 18.3. Analice los problemas de bloqueo mortal, espera indefinida e inanición, y las diferentes estrategias para resolver estos problemas.
- 18.4. Compare los candados binarios con los candados exclusivos/compartidos. ¿Por qué es preferible el segundo tipo de candados?
- 18.5. Describa los protocolos esperar-morir y herir-esperar para evitar el bloqueo mortal.
- 18.6. Describa los protocolos de espera cautelosa, no esperar y tiempo predefinido para prevenir el bloqueo mortal.
- 18.7. ¿Qué es una marca de tiempo? ¿Cómo genera el sistema las marcas de tiempo?
- 18.8. Explique el protocolo de ordenamiento por marca de tiempo para el control de concurrencia. ¿Qué diferencia hay entre el ordenamiento por marca de tiempo estricto y el ordenamiento por marca de tiempo básico?
- 18.9. Analice dos técnicas de multiversión para el control de concurrencia.
- 18.10. ¿Qué es un bloqueo de certificación? ¿Qué ventajas y desventajas tiene el uso de este tipo de bloqueos?
- 18.11. ¿Qué diferencia hay entre las técnicas de control de concurrencia optimistas y las de otra índole? ¿Por qué se les llama también técnicas de validación o de certificación? Analice las fases representativas de un método para el control de concurrencia optimista.
- 18.12. ¿De qué manera la granularidad de los elementos de información afecta el rendimiento del control de concurrencia? ¿Qué factores afectan la selección de la granularidad para los elementos de información?
- 18.13. ¿Qué tipos de bloqueos se necesitan para las operaciones de inserción y eliminación?
- 18.14. ¿Qué es un registro fantasma? Explique los problemas que puede provocar un registro fantasma para el control de la concurrencia.
- 18.15. ¿Cómo es que el bloqueo de índice resuelve el problema del fantasma?
- 18.16. ¿Qué es un bloqueo de predicado?

Ejercicios

- 18.17. Demuestre que el protocolo básico de bloqueo de dos fases garantiza la seriabilidad por conflictos de los planes. (Sugerencia: Demuestre que, si un grafo de seriabilidad de un plan tiene un ciclo, entonces por lo menos una de las transacciones que participan en el ciclo no obedece el protocolo de bloqueo de dos fases).
- 18.18. Modifique las estructuras de datos correspondientes a los candados de modo múltiple y los algoritmos para bloquear lectura (X), bloquear escritura (X) y desbloquear (X) de modo que sea posible la promoción y la degradación de candados. (Sugerencia: El candado necesita seguir la pista a los identificadores de las transacciones que poseen los candados, si las hay).

- 18.19. Demuestre que el bloqueo de dos fases estricto garantiza que los planes serán estrictos.
- 18.20. Demuestre que los protocolos esperar-morir y herir-esperar evitan el bloqueo mortal y la espera indefinida.
- 18.21. Demuestre que la espera cautelosa evita el bloqueo mortal.
- 18.22. Aplique el algoritmo de ordenamiento por marca de tiempo a los planes de las figuras 17.8(b) y (c), y determine si el algoritmo permite la ejecución de esos planes.
- 18.23. Repita el ejercicio 18.22, pero use el método de ordenamiento por marca de tiempo de multiversión.

Bibliografía selecta

El protocolo de bloqueo de dos fases, y el concepto de candados de predicado se presenta en Eswaran *et al.* (1976). Los libros de Bernstein *et al.* (1988), Gray y Reuter (1993) y Papadimitriou (1986) se dedican al control de concurrencia y a la recuperación. El bloqueo se estudia en Gray *et al.* (1975), Lien y Weinberger (1978), Kedem y Silberschatz (1980) y Korth (1983). Los bloqueos mortales y los grafos de espera se formalizaron en Holt (1972), y los esquemas de esperar-morir y herir-esperar aparecen en Rosenkrantz *et al.* (1978). La espera cautelosa se trata en Hsu *et al.* (1992). Helal *et al.* (1993) comparan diversas estrategias de bloqueo. Las técnicas para el control de concurrencia basadas en marcas de tiempo se explican en Bernstein y Goodman (1980) y en Reed (1983). El control de concurrencia optimista se analiza en Kung y Robinson (1981) y en Bessiouni (1988). Papadimitriou y Kallakis (1979) y Bernstein *et al.* (1983) analizan las técnicas de multiversión. El ordenamiento por marca de tiempo de multiversión fue propuesto en Reed (1978, 1983), y el bloqueo de dos fases de multiversión se examina en Lai y Wilkinson (1984). Un método de múltiples granularidades de bloqueo se propuso en Gray *et al.* (1975), y los efectos de estas granularidades se analizan en Ries y Stonebraker (1977). Bhargava y Reidl (1988) presenta una estrategia para escoger dinámicamente entre varios métodos de control de concurrencia y recuperación.

Otros trabajos recientes sobre control de concurrencia incluyen el control de concurrencia basado en semántica (Badrinath y Ramamritham 1992), los modelos de transacciones para actividades de larga ejecución (Dayal *et al.* 1991), y la gestión de transacciones de múltiples niveles (Hassse y Weikum 1991).

19.1 Conceptos de recuperación

19.1.1 Bosquejo de la recuperación

La recuperación de fallos en las transacciones casi siempre equivale a una *restauración* de la base de datos a algún estado anterior de modo que sea posible *reconstruir* un estado correcto —cercano al momento del fallo— a partir de tal estado anterior. Para lograr esto, el sistema debe conservar, fuera de la base de datos, información sobre las modificaciones hechas a los elementos de información al ejecutarse las transacciones. Por lo regular, dicha información se guarda en la **bitácora del sistema**, como se explicó en la sección 17.2.2. Una estrategia de recuperación representativa podría resumirse informalmente como sigue:

1. Si hay daños extensos en una porción considerable de la base de datos por algún fallo catastrófico, como un aterrizaje de las cabezas del disco, el método de recuperación restaurará una copia anterior de la base de datos que se habría *vacuado* en almacenamiento secundario (casi siempre cinta) y reconstruirá un estado más actualizado, volviendo a aplicar o *rehaciendo* las operaciones de las transacciones confirmadas asentadas en la bitácora hasta el momento del fallo.
2. Cuando la base de datos no presenta daños físicos pero se ha vuelto inconsistente debido a fallos no catastróficos de los tipos 1 a 4 (véase la Sec. 17.1.4), la estrategia consiste en invertir los cambios que provocaron la inconsistencia, *deshaciendo* algunas operaciones. También puede ser necesario *rehacer* algunas operaciones a fin de restaurar un estado consistente de la base de datos, como veremos. En este caso no necesitamos una copia completa de la base de datos; durante la recuperación sólo se consultan las entradas asentadas en la bitácora del sistema.

Podemos distinguir dos técnicas principales para recuperarse de fallos no catastróficos en las transacciones. Las técnicas de **actualización diferida** no actualizan en realidad la base de datos sino hasta *después* de que una transacción llega a su punto de confirmación; en ese momento, las actualizaciones se graban en la base de datos. Antes de ser confirmadas, todas las actualizaciones se asientan en el espacio de trabajo local de la transacción. Durante la confirmación, las actualizaciones se graban primero definitivamente en la bitácora y luego se escriben en la base de datos. Si una transacción falla antes de llegar a su punto de confirmación, no habrá modificado en absoluto la base de datos, por lo que no es preciso DESHACER. Puede ser necesario REHACER, a partir de la bitácora, el efecto de las operaciones de una transacción confirmada, ya que es posible que su efecto todavía no se haya registrado en la base de datos. Por ello, la actualización diferida se conoce también como **algoritmo de NO DESHACER/REHACER**, técnica que analizaremos en la sección 19.2.

En las técnicas de **actualización inmediata**, es posible que algunas operaciones de una transacción actualicen la base de datos *antes* de que la transacción llegue a su punto de confirmación. Sin embargo, estas operaciones casi siempre se asientan en la bitácora en disco mediante escritura forzada antes de aplicarse a la base de datos, lo que hace posible la recuperación. Si una transacción falla después de asentar algunos cambios en la base de datos pero antes de llegar a su punto de confirmación, será preciso anular el efecto de sus operaciones sobre la base de datos; esto es, la transacción deberá revertirse. En el caso general de la actualización inmediata, es preciso *deshacer* y *rehacer* durante la recuperación, y es por ello que se denomina **algoritmo de DESHACER/REHACER**. Una variación del algoritmo

CAPÍTULO 19

Técnicas de recuperación

En este capítulo estudiaremos algunas de las técnicas que pueden servir para recuperarse de fallos en las transacciones. Ya examinamos en la sección 17.1.4 las diferentes causas de los fallos (como caídas del sistema y errores en las transacciones). También hemos explicado ya, en la sección 17.2, muchos de los conceptos empleados por los procesos de recuperación, como son la bitácora del sistema, los puntos de control y los puntos de confirmación.

En primer término presentaremos algunos conceptos sobre la recuperación en la sección 19.1, para lo cual haremos un bosquejo de sus procedimientos y una clasificación de sus algoritmos. También estudiaremos cómo asentar anticipadamente en bitácoras, las actualizaciones en el lugar y las actualizaciones sombra, y el proceso de revertir (anular) el efecto de una transacción. En la sección 19.2 presentaremos técnicas de recuperación basadas en la actualización diferida, llamada también técnica NO DESHACER/REHACER. En la sección 19.3 veremos técnicas de recuperación basadas en la actualización inmediata, entre ellas los algoritmos DESHACER/REHACER y DESHACER/NO REHACER. En la sección 19.4 examinaremos la técnica conocida como paginación de sombra, que se puede clasificar como un algoritmo NO DESHACER/NO REHACER. En la sección 19.5 haremos un breve análisis de la recuperación en transacciones de múltiples bases de datos. Por último, en la sección 19.6, hablaremos de las técnicas para recuperarse de fallos catastróficos.

Las técnicas que analizaremos aquí no son descripciones de cómo algún sistema específico implanta la recuperación. Nuestra intención es describir conceptualmente varias estrategias distintas para la recuperación. Si el lector desea descripciones de las características de recuperación en sistemas específicos, deberá consultar las notas bibliográficas y los manuales de usuario de esos sistemas.

A menudo las técnicas de recuperación están imbricadas con los mecanismos de control de concurrencia. Algunas técnicas funcionan mejor con métodos específicos de control de concurrencia. Intentaremos analizar los conceptos de recuperación sin tomar en cuenta los mecanismos de control de concurrencia, pero señalaremos las circunstancias en las que, si se emplea un cierto protocolo de control de concurrencia, conviene usar un mecanismo específico de recuperación.

en la que todas las actualizaciones se asientan en la base de datos antes de que una transacción se confirme sólo requerirá *deshacer*, por lo cual se le conoce como **algoritmo de DESHACER/NO REHACER**. Estudiaremos estas técnicas en la sección 19.3.

19.1.2 Conceptos de sistema para la recuperación

El proceso de recuperación a menudo está íntimamente ligado con las funciones del sistema operativo, en particular con el almacenamiento intermedio y en memoria caché de páginas del disco en la memoria principal. Por lo regular, una o más páginas del disco que contienen el elemento de información que ha de actualizarse se colocan en un almacenamiento intermedio (*cache*) en memoria principal, donde se actualizan antes de escribirse otra vez en el disco. El almacenamiento en memoria caché de páginas de disco es tradicionalmente una función del sistema operativo, pero, debido a su importancia para que los procedimientos de recuperación sean eficientes, puede ser que esto lo maneje el SGBD llamando a rutinas de bajo nivel del sistema operativo. En general, para fines de recuperación, conviene considerar que cada elemento de información corresponde a una página de disco.

Por lo regular, un grupo de *buffers* dentro de la memoria, llamados *caché del SGBD*, está bajo el control del SGBD y sirve para almacenar elementos de la base de datos. Se utiliza un **directorio del caché** para seguir la pista a los elementos que están en los *buffers*. Este directorio puede ser una tabla de entradas <nombre de elemento, ubicación del *buffer*>. Cuando el SGBD necesita hacer algo con un elemento, primero examina el directorio para determinar si el elemento está en el *caché*. Si no es así, será preciso localizarlo en el disco y copiar las páginas de disco apropiadas en el *caché*. Puede ser necesario **desalojar** algunos de los *buffers* de *caché* para disponer de espacio para el nuevo elemento. Se puede usar alguna estrategia de reemplazo de páginas del sistema operativo, como “el menos recientemente usado” (LRU: *least recently used*) o “primero en entrar, primero en salir” (PEFS, o en inglés FIFO, *first-in-first-out*), para seleccionar el almacenamiento intermedio que se desalojará.

Cada elemento en el *caché* tiene asociado un **bit de modificación**, que se puede incluir en la entrada del directorio y que indica si el elemento ha sido modificado o no. Cuando se lee inicialmente el elemento y se coloca en almacenamiento intermedio, el directorio del *caché* se actualiza con el nombre del nuevo elemento, y se asigna cero al bit de modificación. Tan pronto como se modifica el elemento, el bit de modificación de la entrada del directorio correspondiente se cambia a 1 (uno). Cuando se desaloja un elemento, se escribirá en el disco sólo si su bit de modificación es 1.

Son dos las estrategias principales que se utilizan al desalojar hacia el disco un elemento de información modificado. La primera, denominada **actualización en el lugar**, escribe el elemento en la *misma ubicación en el disco*, sobrescribiendo así el valor anterior del elemento. Con ello, sólo se mantiene una copia de cada elemento en el disco. La segunda estrategia, llamada **creación de sombras**, escribe el nuevo elemento en un lugar diferente del disco, lo que hace posible mantener múltiples copias de un elemento de información. En general, el valor antiguo del elemento antes de la actualización se denomina **imagen “antes”** (BFIM: *before image*), y el nuevo valor después de la actualización se denomina **imagen “después”** (AFIM: *after image*). Con la creación de sombras, tanto la BFIM como la AFIM se conservan en disco; por tanto, no es estrictamente necesario mantener una bitácora para la recuperación. Analizaremos la recuperación basada en sombras en la sección 19.4.

Cuando se usa la actualización en el lugar, es necesario utilizar una bitácora (véase la Sec. 17.2.2) para la recuperación. En este caso, el mecanismo de recuperación debe cuidar que la BFIM del elemento de información se asiente en la entrada de bitácora apropiada, y que dicha entrada se desaloje al disco antes de que la BFIM se sobrescriba con la AFIM. En general, a este proceso se le conoce como **escritura anticipada en la bitácora**. Antes de poder describir un protocolo para la escritura anticipada debemos establecer las diferencias entre dos tipos de entradas de la bitácora: las que se necesitan para DESHACER y las que se necesitan para REHACER. Una **entrada de bitácora tipo REHACER** es una entrada correspondiente a una operación escribir elemento de alguna transacción que incluya el nuevo valor (AFIM) del elemento; se requiere una entrada de este tipo si la técnica de recuperación debe *rehacer* el efecto de la operación a partir de la bitácora asignando la BFIM otra vez al elemento en la base de datos. Además, cuando es posible la reversión en cascada, las entradas leer elemento de la bitácora se consideran entradas tipo DESHACER (véase la Sec. 19.1.3).

Para que sea posible la recuperación con la actualización en el lugar, las entradas apropiadas necesarias para la recuperación se deben asentar permanentemente en la bitácora en disco antes de aplicar modificaciones a la base de datos. Por ejemplo, consideremos el siguiente protocolo de **escritura anticipada en la bitácora** (WAL: *write-ahead logging*) para un algoritmo de recuperación que requiere tanto DESHACER como REHACER:

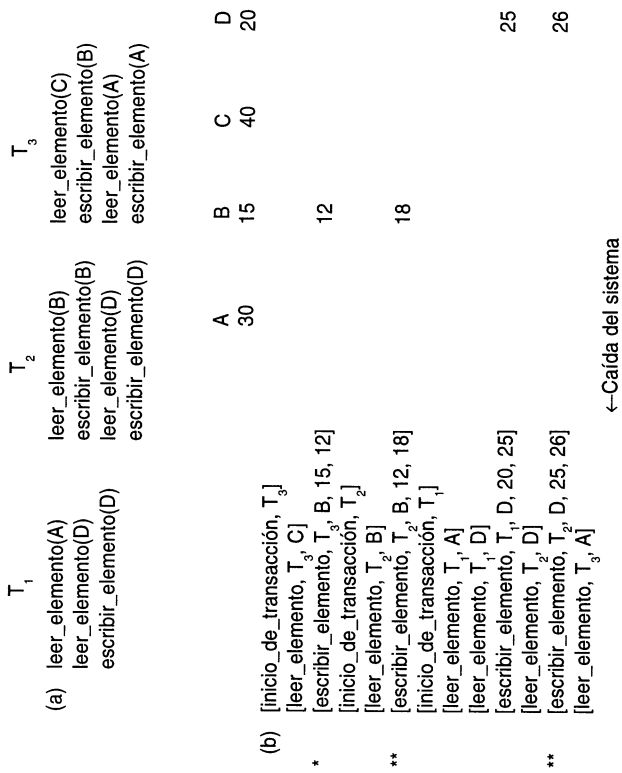
1. La imagen “antes” de un elemento no se puede sobrescribir con su imagen “después” en disco hasta que se haya forzado la escritura en disco de todas las entradas de bitácora tipo DESHACER para la transacción actualizadora (hasta el momento presente).
2. La operación confirmar de una transacción no se puede completar hasta que se haya forzado la escritura en disco de todas las entradas de bitácora tipo REHACER y tipo DESHACER para esa transacción.

Para facilitar el proceso de recuperación, el subsistema de recuperación del SGBD mantiene varias listas relacionadas con las transacciones que se están procesando en el sistema. Éstas incluyen una lista de **transacciones activas** que se han iniciado pero todavía no se han confirmado, una lista de todas las **transacciones confirmadas** desde el último punto de control y una lista de **transacciones abortadas** desde el último punto de control. Mantener estas listas hace más eficiente el proceso de recuperación.

19.1.3 Reversión de transacciones

Si por cualquier razón una transacción falla después de actualizar la base de datos, puede ser necesario **revertir** o **DESHACER** la transacción. Cualesquier valores de elementos de información que la transacción haya alterado deberán restablecerse a sus valores anteriores (BFIMs). Las entradas de la bitácora sirven para recuperar los valores antiguos de los elementos que deben revertirse.

Si una transacción *T* se revierte, cualquier transacción *S* que haya leído mientras tanto el valor de algún elemento de información *X* escrito por *T* también deberá revertirse; de



* T_3 se revierte porque no llegó a su punto de confirmación.

** T_2 se revierte porque lee el valor del elemento B escrito por T_3 .

El resto de las entradas [escribir, ...] de la bitácora se rehacen.

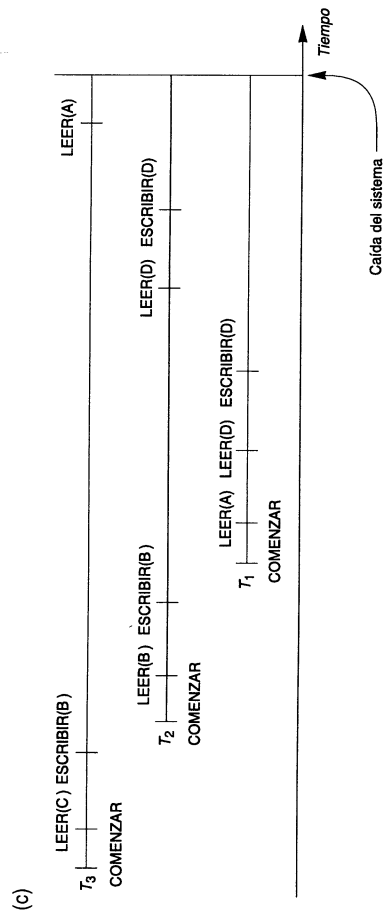


Figura 19.1 Reversión en cascada. (a) Las operaciones de lectura y de escritura de tres transacciones. (b) Bitácora del sistema en el momento de la caída. (c) Operaciones antes de la caída.

manera similar, una vez que S se revierta, cualquier transacción R que haya leído el valor de algún elemento de información Y escrito por S deberá revertirse también, y así sucesivamente. Este fenómeno se denomina **reversión en cascada** y puede ocurrir cuando el protocolo de recuperación garantiza que los planes sean *recuperables* pero no que sean *estrictos* o *sin cascada* (véase la Sec. 17.4.2). Naturalmente, la reversión en cascada puede consumir bastante tiempo, y es por ello que casi todos los mecanismos de recuperación se diseñan de modo que la reversión en cascada *nunca sea necesaria*.

La figura 19.1 muestra un ejemplo en el que se requiere la reversión en cascada. Las operaciones de lectura y escritura de tres transacciones individuales aparecen en la figura 19.1(a). La figura 19.1(b) muestra la bitácora en el momento de una caída del sistema para un determinado plan de ejecución de dichas transacciones. Los valores de los elementos de información A, B, C y D, que las transacciones utilizan, aparecen a la derecha de las entradas de la bitácora. Suponemos que los valores originales de los elementos, que se muestran en la primera línea, son $A = 30$, $B = 15$, $C = 40$ y $D = 20$. En el momento en que falló el sistema, la transacción T_3 no había llegado a su conclusión y deberá revertirse. Las operaciones ESCRIBIR de T_3 , marcadas con un solo * en la figura 19.1(b), son las operaciones de T_3 que se deshacen durante la reversión de la transacción. La figura 19.1(c) muestra gráficamente las operaciones de las diferentes transacciones a lo largo del eje del tiempo.

Ahora debemos verificar si hay reversión en cascada. En la figura 19.1(c) vemos que la transacción T_2 lee el valor del elemento B que escribió la transacción T_3 ; esto puede determinarse también examinando la bitácora. Puesto que T_3 se revierte, T_2 deberá revertirse también. Las operaciones ESCRIBIR de T_2 , marcadas con ** en la bitácora, son las que se deshacen. Observe que sólo es necesario deshacer las operaciones escribir_elemento durante la reversión de transacciones; las operaciones leer_elemento se asientan en la bitácora sólo para determinar si es necesaria la reversión en cascada de transacciones adicionales.

Si el método de recuperación *nunca* requiere reversión de transacciones (porque dicho método garantiza planes sin cascada o estrictos), podemos conservar *información más limitada* en la bitácora del sistema. Tampoco es preciso asentar operaciones leer_elemento en la bitácora, porque éstas sólo son necesarias para determinar la reversión en cascada.

19.2 Técnicas de recuperación basadas en actualización diferida

Las técnicas de actualización diferida se basan en la idea de diferir o posponer cualesquier actualizaciones a la base de datos misma hasta que la transacción complete con éxito su ejecución y alcance su punto de confirmación. Durante la ejecución de las transacciones, las actualizaciones sólo se asientan en la bitácora y en el espacio de trabajo de la transacción. Una vez que esta última llega a su punto de confirmación y se fuerza la escritura de la bitácora en el disco, las actualizaciones se asientan en la base de datos misma. Si una transacción falla antes de llegar a su punto de confirmación, no hay necesidad de deshacer ninguna operación, porque la transacción no ha afectado la base de datos en ningún sentido.

Podemos expresar un protocolo representativo de actualización diferida como sigue:

1. Una transacción no puede modificar la base de datos antes de llegar a su punto de confirmación.

2. Una transacción no llega a su punto de confirmación antes de asentar todas sus operaciones de actualización en la bitácora y forzar la escritura de la bitácora en el disco.

Observe que el paso 2 de este protocolo es una variación del protocolo de escritura anticipada en la bitácora (WAL). Como la base de datos nunca se actualiza antes de confirmarse la transacción, nunca hay necesidad de DESHACER operaciones. Por ello, esta técnica se conoce como algoritmo de NO DESHACER/REHACER. Es necesario REHACER si el sistema falla después de confirmarse la transacción pero antes de que todos sus cambios queden asentados en la base de datos. En este caso, las operaciones de la transacción se rehacen a partir de las entradas de la bitácora.

Por lo regular, el método de recuperación de fallos está muy relacionado con el método de control de concurrencia en sistemas multiusuario. Primero analizaremos la recuperación en sistemas monousuario, donde, claro está, no se requiere control de concurrencia, para entender el proceso de recuperación independientemente de cualquier método de control de concurrencia. Luego veremos cómo el control de concurrencia puede afectar el proceso de recuperación.

19.2.1 Recuperación por actualización diferida en un entorno monousuario

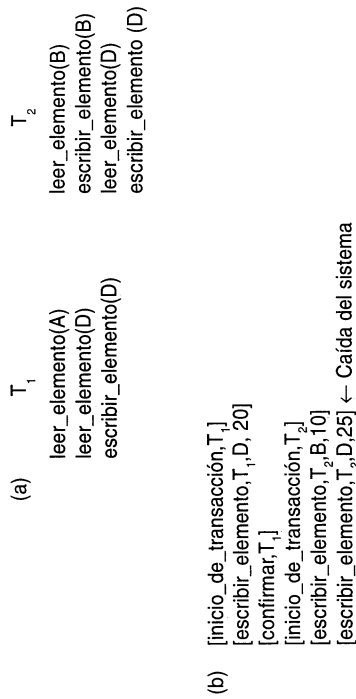
En un entorno monousuario, el algoritmo de recuperación puede ser bastante simple. El algoritmo RAD_1 (recuperación por actualización diferida en un entorno de un solo usuario) utiliza un procedimiento de REHACER, que damos a continuación, para rehacer ciertas operaciones escribir_elemento; funciona como sigue:

PROCEDIMIENTO RAD_1 Usar dos listas de transacciones: las transacciones confirmadas desde el último punto de control y las transacciones activas (cuando más una transacción pertenecerá a esta categoría, porque el sistema es monousuario). A partir de la bitácora, aplicar la operación REHACER a todas las operaciones escribir_elemento de las transacciones confirmadas en el orden en que se escribieron en la bitácora. Reiniciar las transacciones activas.

El procedimiento REHACER se define como sigue:

REHACER(OP_ESCRITURA) Rehacer una operación escribir_elemento (OP_ESCRITURA) consiste en examinar su entrada de bitácora [escribir_elemento, T, X, nuevo_valor] y asignar nuevo_valor al elemento X de la base de datos; esto es, la imagen "después" (AFIM).

La operación REHACER debe ser **idempotente**; es decir, ejecutarla una y otra vez equivale a ejecutarla una sola vez. De hecho, todo el proceso de recuperación debe ser idempotente. La razón es que, si el sistema llega a fallar durante el proceso de recuperación, el siguiente intento de recuperación podría REHACER ciertas operaciones escribir_elemento que ya hubieran sido restauradas durante el primer proceso. El resultado de la recuperación de una caída durante la recuperación deberá ser el mismo que el resultado de recuperarse cuando no hay caída durante la recuperación.



Las operaciones [escribir_elemento, ...] de T_1 se rehacen.
El proceso de recuperación ignora las entradas de bitácora de T_2 .

Figura 19.2 Recuperación por actualización diferida en un entorno monousuario. (a) Las operaciones de lectura y de escritura de dos transacciones. (b) Bitácora del sistema en el momento de la caída.

Observe que la transacción en la lista activa no habrá tenido ningún efecto sobre la base de datos debido al protocolo de actualización diferida, y el proceso de recuperación la ignora por completo. Se revierte implícitamente, porque ninguna de sus operaciones se reflejaron en la base de datos. Sin embargo, ahora es necesario reiniciar la transacción, ya sea que lo haga automáticamente el proceso de recuperación o que lo haga el usuario manualmente.

La figura 19.2 muestra un ejemplo de recuperación en un entorno monousuario, donde el primer fallo ocurre cuando se ejecuta la transacción T_2 , como se aprecia en la figura 19.2(b). El proceso de recuperación rehacerá la entrada [escribir_elemento, T_2 , D, 20] de la bitácora asignando al elemento D el valor 20 (su nuevo valor). El proceso de recuperación ignora las entradas [escribir_elemento, T_2 , ...] de la bitácora porque T_2 no ha sido confirmada. Si ocurre un segundo fallo durante la recuperación del primer fallo, se repetirá el mismo proceso de recuperación de principio a fin, con idénticos resultados.

19.2.2 Actualización diferida con ejecución concurrente en un entorno multiusuario

En los sistemas multiusuario con control de concurrencia, el proceso de recuperación puede ser más complejo, dependiendo de los protocolos empleados para dicho control. En muchos casos, los procesos de recuperación y de control de concurrencia están interrelacionados. En general, cuanto mayor sea el grado de concurrencia que deseemos alcanzar, más difícil se volverá la tarea de recuperación.

Consideremos un sistema en el que el control de concurrencia utiliza bloqueo de dos fases y evita el bloqueo mortal preasignando todos los bloqueos de elementos que una transacción necesita antes que ésta inicie su ejecución. Si queremos combinar el método de recuperación por actualización diferida con esta técnica de control de concurrencia, podemos mantener vigentes todos los bloqueos de elementos hasta que la transacción llegue a su punto

de confirmación. Después de ello, los bloqueos pueden liberarse. Esto garantiza planes estrictos y seriales. Si suponemos que en la bitácora se incluyen entradas [punto de control], el siguiente sería un posible algoritmo de recuperación para tal caso; lo llamaremos RAD_M (recuperación por actualización diferida en un entorno multiusuario). Este procedimiento utiliza el procedimiento REHACER que dimos antes.

PROCEDIMIENTO RAD_M Usar dos listas de transacciones mantenidas por el sistema (CON PUNTOS DE CONTROL) tema: las transacciones T confirmadas desde el último punto de control y las transacciones activas T'. REHACER, a partir de la bitácora, todas las operaciones de ESCRITURA de las transacciones confirmadas, en el orden en que se escribieron en la bitácora. Las transacciones que están activas y no han sido confirmadas se cancelan efectivamente y deberán reintroducirse.

La figura 19.3 muestra un posible plan de transacciones en ejecución. Cuando se marcó el punto de control en el momento t_1 , la transacción T_1 se había confirmado, pero no las transacciones T_3 ni T_4 . Antes de la caída del sistema en el momento t_2 , T_3 y T_2 se habían confirmado, pero no T_4 ni T_5 . Según el método RAD_M, no hay necesidad de rehacer las operaciones escribir elemento de la transacción T_1 , ni de cualquier otra transacción confirmada antes del último punto de control t_1 . En cambio, las operaciones escribir elemento de T_3 y T_2 sí se deben rehacer, porque ambas transacciones llegaron a su punto de confirmación después del último punto de control. Recuerde que se fuerza la escritura de la bitácora antes de confirmar una transacción. Las transacciones T_4 y T_5 se ignoran: se han cancelado o revertido efectivamente porque ninguna de sus operaciones escribir elemento se asentaron en la base de datos por el protocolo de actualización diferida. Nos referiremos posteriormente a la figura 19.3 para ilustrar otros protocolos de recuperación.

Podemos hacer más eficiente el algoritmo de recuperación NO DESHACER/REHACER si observamos que, si un elemento de información X fue actualizado más de una vez por transacciones confirmadas, sólo será necesario REHACER la última actualización de X a partir de la bitácora durante la recuperación, ya que de todos modos esta última operación REHACER sobreescribiría todas las demás actualizaciones. En este caso comenzamos desde el final de la

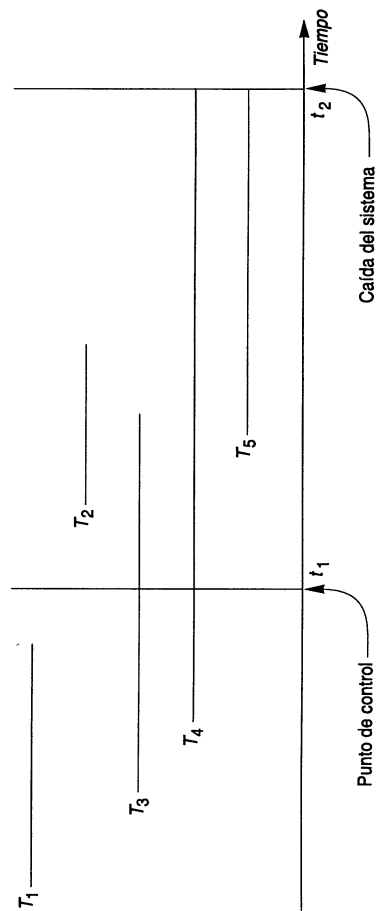


Figura 19.3 Ejemplo de recuperación en un entorno multiusuario.

bitácora; luego, cada vez que se rehace una actualización, se añade a una lista de elementos reactualizados. Antes de aplicar REHACER a cualquier elemento, hay que examinar la lista; si el elemento aparece en ella, no se rehace, pues su valor más reciente ya ha sido recuperado.

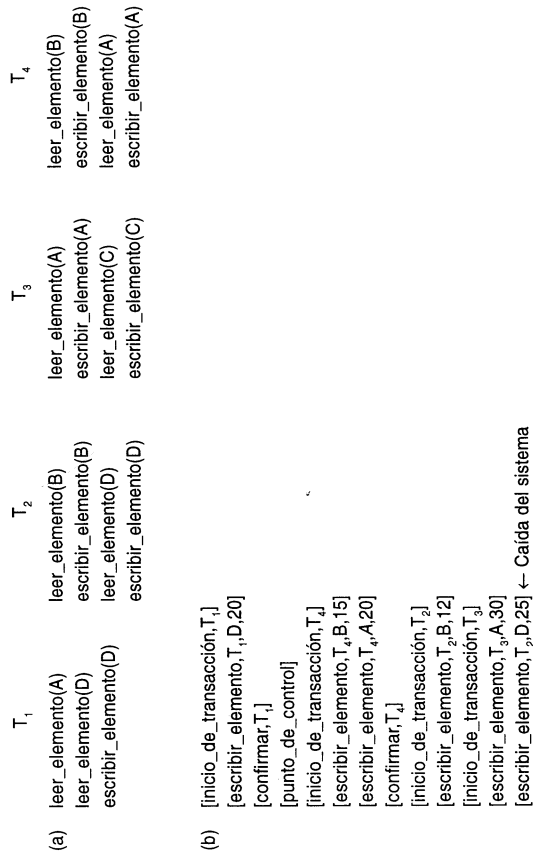
Una desventaja del método descrito aquí es que limita la ejecución concurrente de transacciones porque todos los elementos permanecen bloqueados hasta que la transacción llega a su punto de confirmación. Su principal ventaja es que nunca es preciso deshacer operaciones de las transacciones, por dos razones:

1. Una transacción no asienta sus cambios en la base de datos sino hasta que llega a su punto de confirmación; es decir, hasta que completa con éxito su ejecución. Por ello, ninguna transacción se revierte por haber fallado durante su ejecución.
2. Una transacción nunca leerá el valor de un elemento escrito por una transacción no confirmada, porque los elementos permanecen bloqueados hasta que esta última llega a su punto de confirmación. Por ello, nunca hay reversión en cascada.

La figura 19.4 muestra un ejemplo de recuperación para un sistema multiusuario que utiliza los métodos de recuperación y de control de concurrencia que acabamos de describir.

19.2.3 Acciones de las transacciones que no afectan la base de datos

En general, una transacción incluirá acciones que no afectan la base de datos, como por ejemplo, generar e imprimir mensajes o informes a partir de información obtenida de la base



T_2 y T_3 se ignoran porque no llegaron a su punto de confirmación. T_4 se rehace porque su punto de confirmación fue después del último punto de control del sistema.

Figura 19.4 Recuperación por actualización diferida con transacciones concurrentes. (a) Las operaciones de lectura y de escritura de cuatro transacciones. (b) Bitácora del sistema en el momento de la caída.

de datos. Si una transacción falla antes de completarse, tal vez no deseemos que el usuario reciba estos informes, pues la transacción no logró concluir. Por esto, tales informes deben generarse sólo después de que la transacción haya llegado a su punto de confirmación. Un método común para manejar acciones de este tipo consiste en emitir las órdenes que generan los informes pero mantenerlas como trabajos por lotes. Estos trabajos sólo se ejecutarán si la transacción llega a su punto de confirmación. Si por un fallo esto no sucede, los trabajos por lotes se cancelarán.

19.3 Técnicas de recuperación basadas en actualización inmediata★

En estas técnicas, cuando una transacción emite una orden de actualización, la base de datos se puede actualizar "inmediatamente" sin tener que esperar que la transacción llegue a su punto de confirmación. Sin embargo, en muchas de estas técnicas las operaciones de actualización tendrán que asentarse de todos modos en la bitácora (en disco) antes de aplicarse a la base de datos, a fin de que podamos recuperarlos, en caso de fallo, con el protocolo de escritura anticipada en la bitácora (véase la Sec. 19.1.2).

Si se permite la actualización inmediata, deben tomarse medidas para deshacer el efecto de las operaciones de actualización sobre la base de datos, porque una transacción puede fallar después de haber aplicado algunas actualizaciones a la base de datos misma. Por ello, los esquemas de recuperación basados en actualización inmediata deben incluir la capacidad de hacer revertir una transacción deshaciendo el efecto de sus operaciones escribir_elemento.

En general, podemos distinguir dos categorías principales de algoritmos de actualización inmediata. Si la técnica de recuperación se asegura de que todas las actualizaciones de una transacción se asienten en la base de datos en disco antes de que la transacción se confirme, nunca habrá necesidad de REHACER operaciones de las transacciones confirmadas. Estos algoritmos se denominan DESHACER/NO REHACER. Por otro lado, si se permite a la transacción confirmarse antes de que todos sus cambios se asienten en la base de datos, tendremos el algoritmo de recuperación más general, conocido como DESHACER/REHACER. Ésta es también la técnica más compleja. A continuación veremos dos ejemplos de algoritmos DESHACER/REHACER y dejaremos el desarrollo de la variación DESHACER/NO REHACER como ejercicio para el lector.

19.3.1 Recuperación DESHACER/REHACER basada en actualización inmediata en un entorno monousuario

En primer término consideraremos un sistema monousuario para poder examinar el proceso de recuperación independientemente del control de la concurrencia. Si ocurre un fallo en un sistema monousuario, la transacción que se estaba ejecutando en el momento del fallo puede haber asentado algunos cambios en la base de datos, y el efecto de tales operaciones deberá anularse como parte del proceso de recuperación. Así, el algoritmo de recuperación necesita un procedimiento DESHACER, que se describe más adelante, para anular el efecto de ciertas operaciones escribir_elemento que se aplicaron a la base de datos, después de examinar la entrada correspondiente en la bitácora del sistema. El algoritmo de recuperación RAI_1 (recuperación por actualización inmediata en un entorno de un solo usuario) también utiliza el procedimiento REHACER que definimos antes.

PROCEDIMIENTO RAI_1

1. Usar dos listas de transacciones mantenidas por el sistema: las transacciones confirmadas desde el último punto de control y las transacciones activas (cuando más una transacción pertenecerá a esta categoría, porque el sistema es monousuario).
2. Deshacer, con base en la bitácora, todas las operaciones escribir_elemento de la transacción activa, usando el procedimiento DESHACER que se describe en seguida.
3. Rehacer, con base en la bitácora, todas las operaciones escribir_elemento de las transacciones confirmadas, en el orden en que se escribieron en la bitácora, usando el procedimiento REHACER.

El procedimiento DESHACER se define como sigue:

DESHACER(OP_ESCRITURA) Deshacer una operación escribir_elemento (OP_ESCRITURA) consiste en examinar su entrada de bitácora [escribir_elemento, T, X, valor antiguo, valor_nuevo] y asignar valor_antiguo al elemento X en la base de datos, que es la imagen "antes" (BFIM). La anulación de operaciones escribir_elemento de una o más transacciones, a partir de la bitácora, debe proceder en el orden inverso al orden en que se asentaron las operaciones en la bitácora.

19.3.2 DESHACER/REHACER con actualización inmediata y ejecución concurrente

Cuando se permite la ejecución concurrente, el proceso de recuperación depende una vez más de los protocolos empleados para el control de concurrencia. El procedimiento RAI_M (recuperación por actualización inmediata para un entorno multiusuario) bosqueja una técnica de recuperación para transacciones concurrentes con actualización inmediata. Supongamos que la bitácora incluye puntos de control y que el protocolo de control de concurrencia produce planes estrictos, como lo hace, por ejemplo, el protocolo de bloqueo de dos fases estricto. Recuerde que un plan estricto no permite a una transacción leer o escribir un elemento a menos que la última transacción que haya escrito el elemento se haya confirmado. Sin embargo, puede haber bloqueos mortales en el bloqueo de dos fases estricto, por lo que se requiere DESHACER transacciones. En el caso de un plan estricto, DESHACER una operación requiere reasignar al elemento su antiguo valor (BFIM).

PROCEDIMIENTO RAI_M

1. Usar dos listas de transacciones mantenidas por el sistema: las transacciones confirmadas desde el último punto de control y las transacciones activas.
2. Deshacer todas las operaciones escribir_elemento de las transacciones activas, mediante el procedimiento DESHACER. Las operaciones deberán deshacerse en el orden opuesto a aquel en que se asentaron en la bitácora.
3. Rehacer, con base en la bitácora, todas las operaciones escribir_elemento de las transacciones confirmadas, en el orden en que se escribieron en la bitácora.

19.4 Paginación de sombra★

Este esquema de recuperación no requiere el empleo de una bitácora en un entorno monousuario. En un entorno multiusuario, puede requerir una bitácora si el método de control de concurrencia la necesita.¹ La paginación de sombra considera que la base de datos está compuesta por varias páginas (o bloques) de disco de tamaño fijo —digamos, n — para fines de recuperación. Se construye una **tabla (o directorio) de páginas** con n entradas, donde la i -ésima entrada de la tabla apunta a la i -ésima página de la base de datos en el disco. La tabla de páginas se mantiene en la memoria principal si no es demasiado grande, y todas las referencias —lecturas o escrituras— a las páginas de la base de datos en el disco pasan por la tabla de páginas. Cuando una transacción comienza a ejecutarse, la **tabla de páginas actual** —cuyas entradas apuntan a las páginas más recientes o actuales de la base de datos en el disco— se copia en una **tabla de páginas sombra**, la cual se guarda en disco mientras la transacción usa la tabla de páginas actual.

Durante la ejecución de transacciones, la tabla de páginas sombra *nunca* se modifica. Cuando se efectúa una operación `escribir_elemento`, se crea una nueva copia de la página de base de datos modificada, pero *no se sobrescribe* la copia antigua de esa página. En vez de ello, la nueva página se escribe en algún otro sitio; por ejemplo, en un bloque de disco desocupado. La entrada de la tabla de páginas actual se modifica de modo que apunte al nuevo bloque de disco, pero la tabla de páginas sombra no se modifica y sigue apuntando al antiguo bloque de disco, no modificado. La figura 19.5 ilustra los conceptos de tabla de páginas sombra y de tabla de páginas actual. En el caso de las páginas actualizadas por las

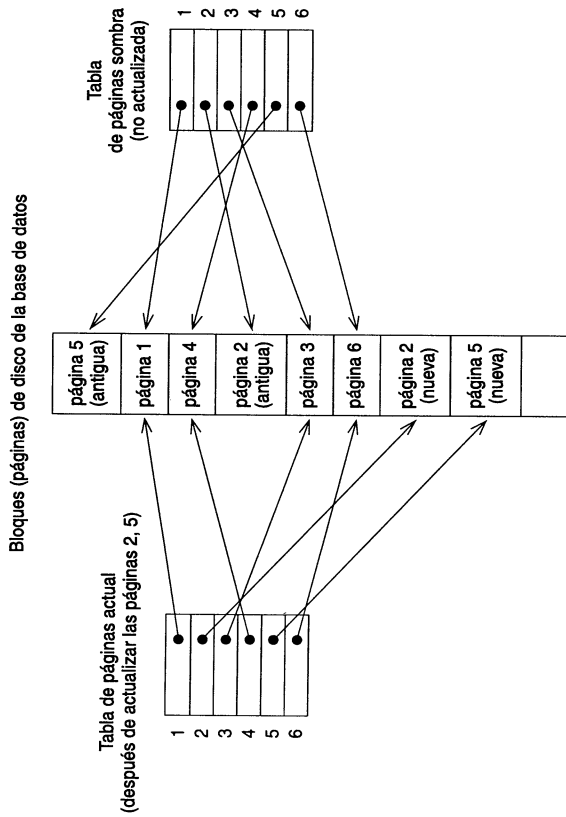


Figura 19.5 Paginación de sombra.

¹Por ejemplo, el SGBD experimental System R utiliza paginación de sombra además de puntos de control y bitácoras.

transacciones, se conservan dos versiones. La tabla de páginas sombra hace referencia a la versión antigua, y la tabla de páginas actual se refiere a la versión nueva.

Para recuperarse de un fallo mientras se ejecuta una transacción basta con liberar las páginas modificadas de la base de datos y desechar la tabla de páginas actual. El estado de la base de datos antes de ejecutarse la transacción está disponible a través de la tabla de páginas sombra, y se recupera haciendo que dicha tabla sea otra vez la tabla de páginas actual. Así, la base de datos vuelve a su estado previo a la transacción que se estaba ejecutando cuando ocurrió la caída, y cualesquier páginas que se hayan modificado serán desechadas. A la confirmación de una transacción corresponde desechar la tabla de páginas sombra previa y liberar las páginas antiguas a las que hace referencia. Como la recuperación no implica anular ni rehacer operaciones, esta técnica de recuperación puede clasificarse como NO DESHACER/NO REHACER.

La ventaja de la paginación de sombra es que simplifica bastante la anulación del efecto de la transacción en ejecución. No hay necesidad de anular ni de rehacer operaciones de las transacciones. En un entorno multiusuario con transacciones concurrentes es preciso incorporar bitácoras y puntos de control a la técnica de paginación de sombra. Una desventaja de este método es que las páginas actualizadas de la base de datos cambian de lugar en el disco, y esto dificulta mucho mantener juntas las páginas relacionadas sin emplear estrategias de manejo de almacenamiento muy complejas. Además, si la tabla (directorio) de páginas es grande, el gasto extra de escribir tablas de páginas sombra en disco cada vez que se confirmen las transacciones será significativo. Una complicación adicional es la forma de realizar la **recolección de basura** cuando una transacción se confirma. Las páginas viejas a las que hace referencia la tabla de páginas sombra y que se han actualizado deberán liberarse y añadirse a una lista de páginas libres para su uso posterior. Estas páginas ya no se necesitarán después de que la transacción se confirme, y la tabla de páginas actual sustituye a la tabla de páginas sombra convirtiéndose en la tabla de páginas válida.

19.5 Recuperación en transacciones de múltiples bases de datos★

Hasta ahora hemos supuesto implícitamente que una transacción tiene acceso a una sola base de datos. En algunos casos, una sola transacción, llamada **transacción de multibases de datos**, puede requerir acceso a varias bases de datos. Éstas podrían incluso estar almacenadas en diferentes tipos de SGBD; por ejemplo, algunos SGBD podrían ser relacionales, y otros jerárquicos o de red. En tal caso, cada uno de los SGBD implicados en la transacción tendrá su propia técnica de recuperación y un gestor de transacciones independiente de los de los otros SGBD. Esta situación es un tanto similar al caso de un sistema de gestión de base de datos distribuida (véase el Cap. 23), donde partes de la base de datos residen en diferentes sitios conectados por una red de comunicaciones.

Para mantener la atomicidad de una transacción de multibases de datos, es necesario contar con un mecanismo de recuperación de dos niveles. Se requiere un **gestor de recuperación global**, o **coordinador**, además de los gestores de recuperación locales. El coordinador suele seguir un protocolo llamado **protocolo de confirmación de dos fases**, las cuales pueden definirse como sigue:

FASE 1: Cuando todas las bases de datos participantes le indican al coordinador que la parte de la transacción en la que interviene cada una ha concluido, el coordinador envía el

mensaje "prepárese para confirmar" a cada participante a fin de que se apresten a confirmar la transacción. Cada una de las bases de datos que reciban ese mensaje forzará la escritura de todos los registros de la bitácora en disco y después enviará una señal de "lista para confirmar" o "correcto" al coordinador. Si falla la escritura forzada a disco o la transacción local no puede confirmarse por alguna razón, la base de datos participante envía una señal "imposible confirmar" o "incorrecto" al coordinador. Si éste no recibe respuesta de una base de datos dentro de un cierto intervalo de tiempo, supone que la respuesta es "incorrecto".

FASE 2: Si todas las bases de datos participantes contestan "correcto", la transacción tiene éxito y el coordinador envía a las participantes una señal "confirmar" para esa transacción. Como todos los efectos locales de la transacción se han asentado en las bitácoras de las bases de datos participantes, ya es posible recuperarse de un fallo. Cada base de datos participante completa la confirmación de la transacción escribiendo en la bitácora una entrada [confirmar] para esa transacción y actualizando permanentemente la base de datos si es necesario. Por otro lado, si una o más de las bases de datos participantes envió una respuesta "incorrecto" al coordinador, la transacción habrá fallado, y el coordinador enviará a cada participante un mensaje de "revertir" o DESHACER el efecto local de la transacción. La anulación se efectúa usando la bitácora. ■

El efecto neto del protocolo de confirmación de dos fases es que o bien todas las bases de datos participantes confirman el efecto de la transacción o ninguna de ellas lo hace. Si llegara a fallar cualquiera de las participantes —o el coordinador— siempre será posible recuperarse hasta un estado en el que la transacción se haya confirmado o bien se haya revertido. Un fallo durante la fase 1 o antes casi siempre obliga a hacer revertir la transacción, pero un fallo durante la fase 2 significa que una transacción exitosa puede recuperarse y confirmarse.

19.6 Respaldo de bases de datos y recuperación de fallos catastróficos

Hasta aquí, todas las técnicas que hemos estudiado se aplican a fallos no catastróficos. Una suposición clave ha sido que la bitácora del sistema se mantiene en disco y no se pierde como consecuencia del fallo. De manera similar, la tabla de páginas sombra se debe almacenar en disco para hacer posible la recuperación cuando se use la paginación de sombra. Las técnicas de recuperación que hemos visto usan las entradas de la bitácora del sistema o la tabla de páginas sombra para recuperarse de un fallo llevando de nuevo la base de datos a un estado consistente.

El gestor de recuperación de un SGRD debe estar equipado también para manejar fallos más catastróficos, como son las caídas de disco. La técnica principal para manejar tales caídas es la de **respaldo de la base de datos**. La base de datos completa y la bitácora se copian periódicamente en un medio de almacenamiento económico como las cintas magnéticas. En caso de un fallo catastrófico del sistema, la copia de respaldo más reciente se puede cargar de la cinta al disco, y el sistema podrá reiniciarse.

Para evitar la pérdida de todos los efectos de las transacciones que se han ejecutado desde el último respaldo, se acostumbra respaldar la bitácora del sistema copiándola periódicamente en cinta magnética. La bitácora suele ser bastante más pequeña que la base de datos misma y por tanto se puede respaldar con mayor frecuencia. Si se respalda la bitácora,

los usuarios no pierden todas las transacciones que han efectuado desde el último respaldo de la base de datos. Es posible reconstruir el efecto sobre la base de datos de todas las transacciones confirmadas asentadas en la parte de la bitácora que se respaldó. Se inicia una nueva bitácora después de cada operación de respaldo de la base de datos. Así, para recuperarse de un fallo de disco, lo primero que se hace es recrear la base de datos en disco a partir de su última copia de seguridad en cinta. Después, se reconstruyen los efectos de todas las transacciones confirmadas cuyas operaciones se hayan asentado en la copia de respaldo de la bitácora del sistema.

19.7 Resumen

En este capítulo estudiamos las técnicas para recuperarse de fallos de las transacciones. El objetivo principal de la recuperación es asegurarse de que las transacciones tengan la propiedad de atomicidad. Si una transacción falla antes de completar su ejecución, el mecanismo de recuperación deberá cerciorarse de que no tenga efectos permanentes sobre la base de datos. Primero presentamos un panorama sobre un proceso de recuperación y luego analizamos conceptos del sistema relacionados con la recuperación, como son el almacenamiento *cache*, la actualización en el lugar y la creación de sombras, las imágenes "antes" y "después" de un elemento de información, las operaciones de recuperación DESHACER y REHACER, y el protocolo de escritura anticipada en la bitácora.

Después vimos dos diferentes estrategias de recuperación: actualización diferida y actualización inmediata. Las técnicas de actualización diferida posponen la actualización real de la base de datos hasta que una transacción llega a su punto de confirmación. La transacción fuerza la escritura de la bitácora en disco antes de grabar las actualizaciones en la base de datos. Esta estrategia, cuando se emplea con ciertos métodos de control de concurrencia, está diseñada de modo que nunca sea necesario revertir las transacciones, y la recuperación consiste simplemente en rehacer, a partir de la bitácora, las operaciones de las transacciones confirmadas después del último punto de control. La actualización diferida puede dar lugar a un algoritmo de recuperación conocido como NO DESHACER/REHACER. Las técnicas de actualización inmediata pueden aplicar cambios a la base de datos antes de que la transacción llegue a buen término. En algunos protocolos, cualesquier cambios que se apliquen a la base de datos deben asentarse primero en la bitácora, la cual se graba de inmediato en disco para que estas operaciones se puedan anular en caso de ser necesario. También presentamos a grandes rasgos un algoritmo de recuperación para actualización inmediata denominado DESHACER/REHACER. También puede usarse otro algoritmo, conocido como DESHACER/NO REHACER, para la actualización inmediata si todas las acciones de las transacciones se graben en la base de datos antes de la confirmación.

Analizamos la técnica de paginación de sombra para la recuperación, que sigue la pista a las páginas viejas de la base de datos mediante una tabla de páginas sombra. Esta técnica, que se clasifica como NO DESHACER/NO REHACER, no requiere una bitácora en los sistemas monousuario pero sí en los de múltiples usuarios. Después examinamos el protocolo de confirmación de dos fases, que sirve para recuperarse de fallos en los que intervienen transacciones de multibases de datos. Por último, hablamos de la recuperación de fallos catastróficos, que por lo regular se efectúa respaldando la base de datos y la bitácora en cinta. La bitácora puede respaldarse con mayor frecuencia que la base de datos, y la copia de seguridad de la bitácora puede servir para rehacer las operaciones posteriores al último respaldo de la base de datos.

Preguntas de repaso

- 19.1. Analice los diferentes tipos de fallos de las transacciones. ¿Qué significa fallo catastrófico?
- 19.2. Explique las acciones que realizan las operaciones leer_elemento y escribir_elemento sobre una base de datos.
- 19.3. (Repaso del capítulo 17.) ¿Para qué se usa la bitácora del sistema? ¿Qué tipo de entradas suele contener una bitácora? ¿Qué son los puntos de control, y por qué son importantes? ¿Qué son los puntos de confirmación de las transacciones, y por qué son importantes?
- 19.4. ¿Cómo usa el subsistema de recuperación las técnicas de almacenamiento intermedio y caché?
- 19.5. ¿Qué es la imagen “antes” (BFIM) y la imagen “después” (AFIM) de un elemento de información? ¿Qué diferencia hay entre la actualización en el lugar y la creación de sombras, en relación con su manejo de las BFIM y las AFIM?
- 19.6. ¿Qué son las entradas de bitácora tipo DESHACER y tipo REHACER?
- 19.7. Describa el protocolo de escritura anticipada en la bitácora.
- 19.8. Identifique tres listas de transacciones que el subsistema de recuperación mantenga.
- 19.9. ¿Qué significa la reversión de transacciones? ¿Por qué es necesario verificar si hay reversión en cascada? ¿Cuáles técnicas de recuperación no requieren reversión?
- 19.10. Analice las operaciones DESHACER y REHACER y las técnicas de recuperación que las utilizan.
- 19.11. Explique la técnica de recuperación de actualización diferida. ¿Cuál es su principal ventaja? ¿Por qué se le llama método de NO DESHACER/REHACER?
- 19.12. ¿Cómo puede la recuperación manejar las operaciones que no afectan la base de datos, como la impresión de informes por parte de la transacción?
- 19.13. Explique la técnica de actualización inmediata tanto en entornos monousuario como multiusuario. ¿Cuáles son las ventajas y desventajas de la actualización inmediata?
- 19.14. ¿Qué diferencia hay entre los algoritmos DESHACER/REHACER y DESHACER/NO REHACER para la recuperación con actualización inmediata? Bosqueje un algoritmo de DESHACER/NO REHACER.
- 19.15. Describa la técnica de recuperación con paginación de sombra. ¿En qué circunstancias no requiere una bitácora?
- 19.16. Describa el protocolo de confirmación de dos fases para las transacciones de multibases de datos.
- 19.17. Explique cómo se maneja la recuperación de fallos catastróficos.

Ejercicios

- 19.18. Suponga que el sistema se cae antes de que la entrada [leer_elemento, T₃, A] se asiente en la bitácora de la figura 19.1(b); ¿afectará esto el proceso de recuperación?

```
[inicio_de_transacción, T1]
[leer_elemento, T1, A]
[escribir_elemento, T1, D, 20]
[confirmar, T1]
[punto_de_control]
[inicio_de_transacción, T2]
[leer_elemento, T2, B]
[escribir_elemento, T2, B, 12]
[inicio_de_transacción, T4]
[leer_elemento, T4, B]
[escribir_elemento, T4, B, 15]
[inicio_de_transacción, T3]
[escribir_elemento, T3, A, 30]
[leer_elemento, T3, A]
[escribir_elemento, T3, A, 20]
[confirmar, T3]
[leer_elemento, T2, D]
[escribir_elemento, T2, D, 25] ← Caída del sistema
```

Figura 19.6 Ejemplo de plan y su bitácora correspondiente.

- 19.19. Suponga que el sistema se cae antes de que la entrada [escribir_elemento, T₃, D, 25, 26] se asiente en la bitácora de la figura 19.1(b); ¿afectará esto el proceso de recuperación?
- 19.20. La figura 19.6 muestra la bitácora correspondiente a un cierto plan de las transacciones T₁, T₂, T₃ y T₄ de la figura 19.4, en el momento de caerse el sistema. Suponga que se usa el *protocolo de actualización inmediata* con puntos de control. Describa el proceso de recuperación de la caída del sistema. Especifique cuáles transacciones se hacen revertir, cuáles operaciones de la bitácora se rehacen y cuáles (si existen) se deshacen, y si hay reversión en cascada.
- 19.21. Suponga que se usa el protocolo de actualización diferida con el ejemplo de la figura 19.6. Indique las diferencias que habría en la bitácora si, en el caso de la actualización diferida, se eliminaran las entradas innecesarias; luego describa el proceso de recuperación, empleando la bitácora modificada. Suponga que sólo se aplican operaciones REHACER, y especifique cuáles operaciones de la bitácora se rehacen y cuáles se ignoran.

Bibliografía selecta

Los libros de Bernstein *et al.* (1987) y Papadimitriou (1986) están dedicados a la teoría y los principios del control de concurrencia y la recuperación. El de Gray y Reuter (1993) es una obra enciclopédica sobre control de concurrencia, recuperación y otros aspectos del procesamiento de transacciones.

Verhoefstad (1978) presenta una explicación didáctica y un repaso de las técnicas de recuperación en los sistemas de bases de datos. La clasificación de los algoritmos basada en sus características

de DESHACER/REHACER se analiza en Haerder y Reuter (1983) y en Bernstein *et al.* (1983). Gray (1978) trata la recuperación, junto con otros aspectos de la implementación de sistemas operativos para bases de datos. La técnica de paginación de sombra se estudia en Lorie (1977), Verhoefstad (1978) y Reuter (1980). Gray *et al.* (1981) analiza el mecanismo de recuperación en System R. Lockeman y Knutsen (1968), Davies (1972) y Bjork (1973) se cuentan entre los primeros artículos que se ocupan de la recuperación. Chandy *et al.* (1975) trata la reversión de transacciones. Lilien y Bhargava (1985) analiza el concepto de bloque de integridad y su empleo para mejorar la eficiencia de la recuperación.

La recuperación por escritura anticipada en bitócora se explica en Jhingran y Khedkar (1992) y se utiliza en el sistema ARIES (Mohan *et al.* 1992a). Entre los trabajos más recientes sobre recuperación están las transacciones compensadoras (Korth *et al.* 1990) y la recuperación de bases de datos en memoria principal (Kumar 1991). Los algoritmos de recuperación de ARIES (Mohan *et al.* 1992) han tenido un gran éxito en la práctica. Franklin *et al.* (1992) analiza la recuperación en el sistema EXODUS.

CAPÍTULO 20

Seguridad y autorización en bases de datos

En este capítulo estudiaremos las técnicas empleadas para proteger la base de datos contra personas que no estén autorizadas para tener acceso a una parte de la base de datos o a toda. La sección 20.1 ofrece una introducción a los problemas de seguridad y un panorama sobre los temas que se cubrirán en el resto del capítulo. En la sección 20.2 se analizan los mecanismos empleados para otorgar y revocar privilegios en los sistemas de bases de datos relacionales y en SQL; estos mecanismos suelen denominarse *control de acceso discrecional*. La sección 20.3 ofrece un panorama sobre los mecanismos para imponer múltiples niveles de seguridad: un aspecto más reciente de la seguridad de los sistemas de bases de datos que se conoce como *control de acceso obligatorio*. En la sección 20.4 se explica brevemente el problema de la seguridad en las bases de datos estadísticas.

Los lectores que estén interesados únicamente en los mecanismos básicos de seguridad tendrán suficiente si estudian las secciones 20.1 y 20.2.

20.1 Introducción a los problemas de seguridad en las bases de datos

20.1.1 Tipos de seguridad

La seguridad de las bases de datos es un área amplia que abarca varios temas, entre ellos los siguientes:

- Cuestiones éticas y legales relativas al derecho a tener acceso a cierta información. Es posible que parte de ésta se considere privada y que las personas no autorizadas no puedan tener acceso a ella legalmente. En Estados Unidos, los gobiernos federal y de varios estados tienen leyes sobre la confidencialidad de la información.

- Cuestiones de política en el nivel gubernamental, institucional o corporativo, relacionadas con las clases de información que no deben estar disponibles para el público; por ejemplo, clasificaciones de crédito y expedientes médicos personales.
- Cuestiones relacionadas con el sistema, como los *niveles del sistema* en los que deben manejarse las diversas funciones de seguridad; por ejemplo, el nivel del hardware físico, el nivel del sistema operativo o el nivel del SGBD.
- La necesidad en algunas organizaciones de identificar múltiples *niveles de seguridad* y de clasificar los datos y los usuarios según estos niveles; por ejemplo, secreto máximo, secreto, confidencial y no clasificado. La política de seguridad de la organización relacionada con el permiso para tener acceso a las diversas clasificaciones de los datos se debe hacer cumplir.

En un sistema multiusuario, el SGBD debe proveer técnicas que permitan a ciertos usuarios o grupos de usuarios tener acceso a porciones selectas de una base de datos sin tener acceso al resto. Esto es importante sobre todo cuando muchos usuarios distintos dentro de la misma organización necesitan usar una gran base de datos integrada. La información confidencial, como los salarios de los empleados, se debe ocultar a la mayor parte de los usuarios del sistema. Por lo regular, un SGBD cuenta con un **subsistema de seguridad y autorización de la base de datos** que se encarga de garantizar la seguridad de porciones de la base de datos contra el acceso no autorizado.

Actualmente se acostumbra hablar de dos tipos de mecanismos de seguridad en las bases de datos:

- Los **mecanismos de seguridad discrecionales** se usan para otorgar privilegios a los usuarios, incluida la capacidad de tener acceso a archivos, registros o campos de datos específicos en un determinado modo (como modo de lectura, de escritura o de actualización).
- Los **mecanismos de seguridad obligatorios** sirven para imponer seguridad de múltiples niveles clasificando los datos y los usuarios en varias clases (o niveles) de seguridad e implementando después la política de seguridad apropiada de la organización. Por ejemplo, una política de seguridad común consiste en permitir a los usuarios de un cierto nivel de clasificación ver sólo los elementos de información clasificados en el mismo nivel que el usuario (o en un nivel inferior).

Hablaremos de la seguridad discrecional en la sección 20.2 y de la seguridad obligatoria en la sección 20.3.

Otro problema de seguridad común a todos los sistemas de cómputo es el de evitar que personas no autorizadas tengan acceso al sistema mismo, ya sea para obtener información o para efectuar cambios mal intencionados en una porción de la base de datos. El mecanismo de seguridad de un SGBD debe incluir formas de restringir el acceso al sistema como un todo. Esta función se denomina **control de acceso** y se pone en práctica creando cuentas de usuarios y contraseñas para que el SGBD controle el proceso de entrada al sistema. Estudiaremos las técnicas de control de acceso en la sección 20.1.3.

Un tercer problema de seguridad es el de controlar el acceso a una **base de datos estadística**, la cual sirve para proporcionar información estadística o resúmenes de valores a partir de diversos criterios. Por ejemplo, una base de datos de información demográfica podría suministrar información estadística basada en grupos de edad, niveles de ingreso, tamaño de

la familia, niveles de educación y otros criterios. Los usuarios de bases de datos estadísticas, como los actuarios gubernamentales o las empresas de investigación de mercados, están autorizados para usarlas para obtener información estadística sobre una población, pero no para tener acceso a información confidencial detallada sobre individuos específicos. La seguridad en las bases de datos estadísticas debe cuidar que la información sobre individuos no sea accesible. En ocasiones es posible deducir ciertos hechos relativos a los individuos a partir de consultas en las que intervienen sólo datos sinópticos de grupos, así que tampoco esto debe permitirse. Este problema, llamado **seguridad en bases de datos estadísticas**, se analiza brevemente en la sección 20.4.

Otra técnica de seguridad es el **cifrado de datos**, que sirve para proteger datos confidenciales que se transmiten por satélite o por algún otro tipo de red de comunicaciones. Así mismo, el cifrado puede proveer protección adicional a secciones confidenciales de una base de datos. Los datos se **codifican** mediante algún algoritmo de codificación. Un usuario no autorizado que tenga acceso a datos codificados tendrá problemas para descifrarlos, pero un usuario autorizado contará con algoritmos (o claves) de decodificación o descifrado para descifrarlos. Se han creado técnicas de cifrado que son muy difíciles de descifrar sin la clave, para aplicaciones militares. No hablaremos aquí de los algoritmos de cifrado.

Un análisis completo sobre la seguridad en los sistemas de cómputo y en las bases de datos rebasa el alcance de este libro. Sólo presentaremos un breve bosquejo de las técnicas de seguridad de bases de datos; el lector interesado puede consultar alguno de los textos citados en la bibliografía al final del capítulo si desea un tratamiento más completo.

20.1.2 La seguridad de la base de datos y el DBA

Como vimos en el capítulo 1, el administrador de bases de datos (DBA) es la autoridad central que controla un sistema de este tipo. Entre las obligaciones del DBA están otorgar privilegios a los usuarios que necesitan usar el sistema y clasificar los usuarios y los datos de acuerdo con la política de la organización. El DBA tiene una **cuenta privilegiada** en el SGBD, a veces denominada **cuenta del sistema**, que confiere capacidades extraordinarias no disponibles para las cuentas y usuarios ordinarios de la base de datos. Esta cuenta es similar a las cuentas *raíz* (*root*) o *superusuario* (*superuser*) que se dan a los administradores de los sistemas de cómputo y que les permiten usar órdenes restringidas del sistema operativo. Las órdenes privilegiadas del DBA incluyen órdenes para otorgar o revocar privilegios a cuentas individuales, usuarios o grupos de usuarios, y para efectuar los siguientes tipos de acciones:

1. **Creación de cuentas:** Esta acción crea una nueva cuenta y contraseña para un usuario o grupo de usuarios, a fin de que puedan tener acceso al SGBD.
2. **Concesión de privilegios:** Esta acción permite al DBA otorgar ciertos privilegios a ciertas cuentas.
3. **Revocación de privilegios:** Esta acción permite al DBA revocar (cancelar) ciertos privilegios que se habían concedido previamente a ciertas cuentas.
4. **Asignación de niveles de seguridad:** Esta acción consiste en asignar cuentas de usuario al nivel apropiado de clasificación de seguridad.

El DBA es responsable de la seguridad global del sistema de base de datos. La acción 1 de la lista sirve para controlar el acceso al SGBD en general, en tanto que las acciones 2 y 3 se usan para controlar las autorizaciones discrecionales, y con la acción 4 se controla la autorización obligatoria.

20.1.3 Protección de acceso, cuentas de usuarios y auditorías de la base de datos

Siempre que una persona o grupo de personas necesite tener acceso a un sistema de bases de datos, deberá solicitar primero una cuenta de usuario; el DBA creará entonces un nuevo **número de cuenta** y **contraseña** para el usuario si la necesidad de acceso es legítima. El usuario debe **ingresar** al SGBD introduciendo el número de cuenta y la contraseña siempre que requiera acceso a la base de datos. El SGBD verifica que el número y la contraseña sean válidos; si lo son, se permite al usuario utilizar el SGBD y tener acceso a la base de datos. Los programas de aplicación también se pueden considerar como usuarios y se les puede exigir que proporcionen contraseñas.

No es difícil llevar el control de los usuarios de la base de datos y sus cuentas y contraseñas; para ello se crea una tabla o archivo cifrado con dos campos: **Número_de_cuenta** y **Contraseña**. El SGBD puede mantener sin dificultad esta tabla. Siempre que se crea una nueva cuenta, se inserta un nuevo registro en la tabla. Cuando se cancela una cuenta, su registro correspondiente debe eliminarse de la tabla.

El sistema de base de datos también debe llevar el control de todas las operaciones que un usuario determinado aplica a la base de datos durante cada **sesión de trabajo**, la cual consiste en la secuencia de interacciones con la base de datos que un usuario realiza desde el momento en que ingresa hasta el momento en que sale. Cuando un usuario ingresa, el SGBD puede asentar su número de cuenta y asociarlo a la terminal desde la cual ingresó. Todas las operaciones que se apliquen desde esa terminal se atribuirán a la cuenta del usuario hasta que éste salga del sistema. Es en extremo importante mantenerse al tanto de las operaciones de actualización que se aplican a la base de datos para que, si ésta es alterada indebidamente, el DBA pueda averiguar cuál usuario lo hizo.

A fin de llevar un control de cada una de las actualizaciones aplicadas a la base de datos y del usuario específico que las aplicó, podemos modificar la bitácora del sistema. Recuerde que la **bitácora del sistema** incluye una entrada por cada operación aplicada a la base de datos y que puede requerirse para la recuperación de una falla de transacción o de una caída del sistema. Podemos expandir las entradas de la bitácora de modo que incluyan también el número de cuenta del usuario y la identificación de la terminal en línea que aplicó cada operación. Si se sospecha una alteración indebida de la base de datos, se efectúa una **auditoría de base de datos**, que consiste en examinar en la bitácora todos los accesos y operaciones aplicadas a la base de datos durante un cierto periodo. Cuando encuentra una operación ilegal o no autorizada, el DBA puede determinar qué número de cuenta se usó para efectuar dicha operación. Estas auditorías son importantes sobre todo en los casos de bases de datos confidenciales actualizadas por muchas transacciones y usuarios, como las bancarias, que son actualizadas por muchos cajeros. Una bitácora de base de datos que se usa principalmente para fines de seguridad suele recibir el nombre de **registro de intervenciones**.

20.2 Control de acceso discrecional basado en privilegios

El método más común para imponer el control de acceso discrecional en un sistema de bases de datos consiste en otorgar y revocar privilegios. Consideraremos los privilegios en el contexto de un SGBD relacional. En particular, analizaremos un sistema de privilegios un tanto parecido al que se creó originalmente para el lenguaje SQL (véase el Cap. 7). Muchos

SGBD relacionales actuales utilizan alguna variación de esta técnica. La idea principal es incluir enunciados adicionales en el lenguaje de consulta con los que el DBA y los usuarios seleccionados puedan otorgar y revocar privilegios.

20.2.1 Tipos de privilegios discrecionales

En SQL2, con el concepto de *identificador de autorización* se hace referencia, aproximadamente, a una cuenta de usuario. Aquí utilizaremos la palabra *usuario* o *cuenta* informalmente en lugar de identificador de autorización; por tanto, para nosotros serán intercambiables los términos *usuario* y *cuenta*. El SGBD debe ofrecer acceso selectivo a cada relación de la base de datos según cuentas específicas. También es posible controlar las operaciones, así que tener una cuenta no necesariamente confiere a su titular toda la funcionalidad que puede ofrecer el SGBD. En términos informales, hay dos niveles de asignación de privilegios para usar el sistema de base de datos:

1. *El nivel de cuenta*: En este nivel, el DBA especifica los privilegios particulares que tiene cada usuario, independientemente de las relaciones de la base de datos.
2. *El nivel de relación*: En este nivel, podemos controlar el privilegio para tener acceso a cada relación o vista individual de la base de datos.

Los privilegios en el **nivel de cuenta** se aplican a las capacidades conferidas a la cuenta misma y pueden incluir los siguientes privilegios: CREATE SCHEMA o CREATE TABLE, para crear un esquema o una relación base; CREATE VIEW, para crear una vista; ALTER, para agregar o eliminar atributos de las relaciones; DROP, para eliminar relaciones o vistas; MODIFY, para insertar, eliminar o modificar tuplas, y SELECT, para obtener información de la base de datos con una consulta SELECT. Cabe indicar que estos privilegios de cuenta se aplican a la cuenta en general. Si una cuenta dada no tiene el privilegio CREATE TABLE, no será posible crear relaciones con ella. Los privilegios en el nivel de cuenta no se definen como parte de SQL2; su definición se deja a los implementadores del SGBD. En versiones anteriores de SQL había un privilegio CREATE TAB para dar a una cuenta el privilegio de crear tablas (relaciones).

El segundo nivel de privilegios se aplica a las relaciones individuales, sin importar si son relaciones base o virtuales (vistas). Estos privilegios *sí están* definidos para SQL2. En el análisis que sigue, el término *relación* se puede referir a una relación base o bien a una vista, a menos que especifiquemos explícitamente una u otra. Los privilegios en el nivel de relación especifican para cada usuario las relaciones individuales a las que se puede aplicar cada tipo de instrucción. Algunos privilegios se refieren también a columnas (atributos) individuales de las relaciones. Las órdenes de SQL2 confieren privilegios *únicamente a nivel de relación y de atributo*. Aunque esto es bastante general, dificulta la creación de cuentas con privilegios limitados. La concesión y revocación de privilegios sigue por lo general un modelo de autorización para privilegios discrecionales denominado **modelo de matriz de acceso**, donde las filas de una matriz M representan sujetos (usuarios, cuentas, programas) y las columnas representan objetos (relaciones, registros, columnas, vistas, operaciones). Cada posición $M(i, j)$ de la matriz representa los tipos de privilegios (lectura, escritura, actualización) que el sujeto i tiene para el objeto j .

Para controlar la concesión y revocación de privilegios de relación, a cada relación R de una base de datos se le asigna una **cuenta propietario**, que por lo regular es la cuenta con la que se creó inicialmente. El propietario de una relación posee *todos* los privilegios para esa

relación. En SQL2, el DBA puede asignar un propietario a todo un esquema, para lo cual crea el esquema y asocia el identificador de autorización apropiado a ese esquema mediante la orden CREATE SCHEMA (véase la Sec. 7.1.1). El poseedor de la cuenta propietario puede a su vez dar privilegios para cualquiera de las relaciones de su propiedad a otros usuarios otorgando privilegios a sus cuentas. En SQL es posible otorgar los siguientes tipos de privilegios para cada relación individual R:

- Privilegio SELECT (obtención) para R: Confiere a la cuenta el privilegio de obtención; esto es, la cuenta puede usar la instrucción SELECT en SQL para obtener tuplas de R.
- Privilegios MODIFY para R: Confiere a la cuenta la capacidad de modificar tuplas de R. En SQL este privilegio se subdivide en privilegios UPDATE, DELETE e INSERT para aplicar la orden correspondiente de SQL a R. Por añadidura, los privilegios INSERT y UPDATE pueden especificar que la cuenta sólo puede actualizar ciertos atributos de R.
- Privilegio REFERENCES para R: Éste confiere a la cuenta la capacidad de hacer referencia a la relación R al especificar restricciones de integridad. Este privilegio también se puede restringir a atributos específicos de R.

Cabe señalar que, para crear una vista, la cuenta debe poseer el privilegio SELECT para todas las relaciones que intervienen en la definición de la vista.

20.2.2 Cómo especificar autorizaciones empleando vistas

Por sí mismas, las vistas constituyen un importante mecanismo de autorización discrecional. Por ejemplo, si el propietario A de una relación R desea que otra cuenta B pueda leer únicamente ciertos campos de R, A puede crear una vista V de R que incluya sólo esos atributos, y después otorgar a B el privilegio SELECT para V. Lo mismo se aplica cuando se desea limitar a B a la lectura de sólo ciertas tuplas de R; se puede crear una vista V definiéndola por medio de una consulta que seleccione sólo las tuplas de R que A desea poner al alcance de B. Ilustraremos nuestro análisis con un ejemplo en la sección 20.2.5.

20.2.3 Revocación de privilegios

En algunos casos es deseable otorgar temporalmente algún privilegio a un usuario. Por ejemplo, el propietario de una relación quizá quiera otorgar el privilegio SELECT a un usuario para una tarea específica y luego revocárselo una vez que haya completado la tarea. Por tanto, se necesita un mecanismo para **revocar** privilegios. SQL incluye una orden REVOKE para cancelar privilegios. Pronto veremos cómo utilizarla en nuestro ejemplo (Sec. 20.2.5).

20.2.4 Propagación de privilegios y GRANT OPTION

Siempre que el propietario A de una relación R otorga un privilegio para R a otra cuenta B, el privilegio puede darse a B con la opción de otorgar (GRANT OPTION) o sin ella. Si se concede esta opción, significa que B también podrá otorgar ese privilegio para R a otras cuentas. Supongamos que A otorga a B la GRANT OPTION y que B otorga luego el privilegio para R a una tercera cuenta C, también con GRANT OPTION. De esta manera, los privilegios para R se pueden **propagar** a otras cuentas sin que lo sepa el propietario de R. Si la

cuenta propietario A revoca el privilegio otorgado a B, el sistema deberá revocar automáticamente todos los privilegios que B propagó con base en ese privilegio. Así pues, un SGBD que permita la propagación de privilegios deberá seguir la pista a la concesión de privilegios de modo que la revocación pueda hacerse de manera correcta y completa.

Se han creado técnicas para limitar la propagación de privilegios, aunque todavía no se han implementado en casi ningún SGBD. Limitar la **propagación horizontal** a un número entero i significa que una cuenta B que posee la GRANT OPTION puede otorgar el privilegio a cuando más i usuarios adicionales. La **propagación vertical** es más complicada; limita la profundidad de la concesión de privilegios. Otorgar un privilegio con una propagación vertical de cero equivale a otorgarlo sin GRANT OPTION. Si la cuenta A otorga un privilegio a la cuenta B con una propagación vertical j, con j > 0, esto significa que la cuenta B tendrá GRANT OPTION para ese privilegio, pero B sólo podrá otorgar el privilegio a otras cuentas con una propagación vertical menor que j. En efecto, la propagación vertical limita la secuencia de opciones de concesión que se pueden dar de una cuenta a la siguiente con base en una sola concesión original del privilegio. Ilustraremos esto en el ejemplo que sigue.

20.2.5 Un ejemplo

Supongamos que el DBA crea cuatro cuentas —A1, A2, A3 y A4— y desea que sólo A1 pueda crear relaciones base; en tal caso, el DBA deberá emitir la siguiente orden GRANT (otorgar) en SQL:

GRANT CREATETAB TO A1;

El privilegio CREATETAB (crear tabla) confiere a la cuenta A1 la capacidad de crear nuevas tablas (relaciones base) de la base de datos, y es por tanto un *privilegio de cuenta*. Este privilegio formaba parte de versiones anteriores de SQL, pero ahora su definición se deja a cada implementación individual del sistema. En SQL2, puede lograrse el mismo efecto si el DBA emite una orden CREATE SCHEMA, como sigue:

CREATE SCHEMA EJEMPLO AUTHORIZATION A1;

Ahora la cuenta usuario A1 puede crear tablas dentro del esquema llamado EJEMPLO. También es posible introducir un *privilegio de cuenta* CREATE SCHEMA definido por el sistema que sustituya a CREATETAB. CREATE SCHEMA conferiría al usuario el privilegio de crear un esquema de base de datos, y por tanto implicaría el privilegio CREATETAB.

Ahora, para continuar nuestro ejemplo, supongamos que A1 crea las dos relaciones base EMPLEADO y DEPARTAMENTO que se muestran en la figura 20.1; con ello, A1 es el propietario de estas dos relaciones y por tanto tiene todos los *privilegios de relación* para las dos. A continuación, supongamos que la cuenta A1 desea otorgar a la cuenta A2 el privilegio

EMPLEADO				DEPARTAMENTO			
NOMBRE	NSS	FECHAN	DIRECCIÓN	SEXO	SALARIO	ND	
				NÚMEROD	NOMBRED	NSSGTE	

Figura 20.1 Las dos relaciones EMPLEADO y DEPARTAMENTO.

de insertar y eliminar tuplas en estas dos relaciones. Sin embargo, A1 no desea que A2 pueda propagar estos privilegios a cuentas adicionales. A1 puede entonces emitir la siguiente orden:

GRANT INSERT, DELETE ON EMPLEADO, DEPARTAMENTO TO A2;

Observe que la cuenta propietario de una relación tiene automáticamente GRANT OPTION, y puede otorgar privilegios para la relación a otras cuentas. Sin embargo, la cuenta A2 no puede otorgar privilegios INSERT y DELETE para las tablas EMPLEADO y DEPARTAMENTO, porque no recibió la GRANT OPTION en la orden anterior. Ahora, supongamos que A1 desea permitir a la cuenta A3 obtener información de cualquiera de las dos tablas y también propagar el privilegio SELECT a otras cuentas. A1 puede entonces emitir la siguiente orden:

GRANT SELECT ON EMPLEADO, DEPARTAMENTO TO A3 WITH GRANT OPTION;

La cláusula "WITH GRANT OPTION" significa que A3 puede propagar el privilegio a otras cuentas empleando GRANT. Por ejemplo, A3 puede otorgar a A4 el privilegio SELECT para la relación EMPLEADO emitiendo la siguiente orden:

GRANT SELECT ON EMPLEADO TO A4;

Observe que A4 no puede propagar el privilegio SELECT a otras cuentas, porque no se le otorgó la GRANT OPTION. Supongamos ahora que A1 decide revocar el privilegio SELECT que A3 tiene para la relación EMPLEADO; A1 puede emitir esta orden:

REVOKE SELECT ON EMPLEADO FROM A3;

El SCBD deberá ahora revocar automáticamente el privilegio SELECT para EMPLEADO que tiene A4, porque A3 le otorgó ese privilegio a A4 pero A3 ya no lo tiene. A continuación, supongamos que A1 quiere devolver a A3 una capacidad de selección limitada para la relación EMPLEADO y desea que A3 pueda propagar ese privilegio. La limitación es que sólo podrá obtener los atributos NOMBRE, FECHAN y DIRECCIÓN, y sólo las tuplas con ND = 5. Así pues, A1 puede crear esta vista:

**CREATE VIEW A3EMPLEADO AS
SELECT NOMBRE, FECHAN, DIRECCIÓN
FROM EMPLEADO
WHERE ND=5;**

Una vez creada la vista, A1 puede otorgar a A3 el privilegio SELECT para la vista A3EMPLEADO como sigue:

GRANT SELECT ON A3EMPLEADO TO A3 WITH GRANT OPTION;

Por último, supongamos que A1 desea dar permiso a A4 de actualizar sólo el atributo SALARIO de EMPLEADO; A1 puede entonces emitir la siguiente orden:

GRANT UPDATE ON EMPLEADO (SALARIO) TO A4;

El privilegio UPDATE o INSERT puede especificar atributos particulares que pueden ser actualizados o insertados en una relación. Otros privilegios (SELECT, DELETE) no pueden limitarse a ciertos atributos, pero su especificidad puede controlarse fácilmente creando las vistas apropiadas que incluyan sólo los atributos deseados. Sin embargo, como no siempre es posible actualizar las vistas, los privilegios UPDATE e INSERT tienen la opción de especificar atributos particulares de una relación base que sí es actualizable.

Un usuario puede recibir un cierto privilegio de dos o más fuentes. Por ejemplo, A4 puede recibir un cierto privilegio UPDATE R tanto de A2 como de A3. En un caso así, aun- que A2 llegara a revocar el privilegio que otorgó a A4, ésta seguiría teniendo el privilegio en virtud de que A3 se lo concedió. Si A3 revoca posteriormente el privilegio de A4, ésta perderá por completo el privilegio.

Ahora ilustraremos brevemente los límites de propagación horizontal y vertical, que actualmente no están disponibles en SQL ni en otros sistemas relacionales. Supongamos que A1 otorga el privilegio SELECT a A2 para la relación EMPLEADO con propagación horizontal = 1 y propagación vertical = 2. En tal caso, A2 puede otorgar el privilegio SELECT a cuan- do más una cuenta, porque la propagación horizontal es 1. Además, A2 no puede otorgar el privilegio a otra cuenta si no es con propagación vertical = 0 (sin GRANT OPTION) o 1, porque el límite de la propagación vertical es 2. Esto significa que A2 deberá reducir la propagación vertical en cuando menos 1 al pasar el privilegio a otras. Con este ejemplo queda claro que las técnicas de propagación horizontal y vertical están diseñadas para li- mitar la propagación de los privilegios.

20.3 Control de acceso obligatorio para seguridad multinivel*

La técnica de control de acceso discrecional para otorgar y revocar privilegios en relacio- nes ha sido, desde siempre, el principal mecanismo de seguridad en los sistemas de bases de datos. Éste es un método de todo o nada: un usuario tiene un cierto privilegio o bien no lo tiene. En muchas aplicaciones se necesita una política de seguridad adicional que clasifica los datos y los usuarios de acuerdo con ciertas clases de seguridad. Esta estrategia, llamada **control de acceso obligatorio**, se combina con los mecanismos de control de ac- ceso discrecional descritos en la sección 20.2. Es importante señalar que la mayor parte de los SCBD comerciales actualmente ofrecen sólo mecanismos para el control de acceso dis- crecional. Sin embargo, la necesidad de una seguridad multinivel existe en aplicaciones gubernamentales, militares y de espionaje, así como en muchas aplicaciones industria- les y corporativas.

Las **clases de seguridad** usuales son secreto máximo (TS: top secret), secreto (S), con- fidencial (C) y no clasificado (U: unclassified), donde TS es el nivel más alto y U el más bajo. Existen otros esquemas de clasificación de seguridad más complejos, en los que las clases de seguridad se organizan en una matriz. Por sencillez, usaremos el sistema con cuatro ni- veles de clasificación de seguridad, donde $TS > S > C > U$, para ilustrar nuestro análisis. El modelo que suele usarse para la seguridad multinivel, denominado modelo Bell-LaPadula, asigna a cada **sujeto** (usuario, cuenta, programa) y **objeto** (relación, tupla, columna, vis- ta, operación) una de las clasificaciones de seguridad TS, S, C o U. Nos referiremos a la cla- sificación de un sujeto S como **clasif(S)** y a la clasificación de un objeto O como **clasif(O)**. Se imponen dos restricciones al acceso a los datos según las clasificaciones sujeto/objeto:

1. Un sujeto S no puede tener acceso de lectura a un objeto O si $\text{clasif(S)} \geq \text{clasif(O)}$. Esto se conoce como la *propiedad de seguridad simple*.
2. Un sujeto S no puede tener acceso de escritura a un objeto O si $\text{clasif(S)} \leq \text{clasif(O)}$. Esto se conoce como la *propiedad ** (o *propiedad estrella*).

La primera restricción hace cumplir la regla obvia de que ningún sujeto puede leer un objeto cuya clasificación de seguridad sea más alta que la del sujeto. La segunda restricción prohíbe a un sujeto escribir un objeto que tenga una clasificación de seguridad menor que la del sujeto. La violación de esta regla permitiría el flujo de información de clasificaciones más altas a clasificaciones más bajas, lo cual viola un precepto básico de la seguridad multinivel.

Para incorporar las ideas de la seguridad multinivel al modelo relacional de bases de datos, se acostumbra considerar los valores de los atributos y las tuplas como objetos de datos. Así, cada atributo A está asociado a un **atributo de clasificación** C en el esquema, y cada valor de atributo de una tupla está asociado a una clasificación de seguridad correspondiente. Además, en algunos modelos se añade un atributo de **clasificación de tupla** CT a los atributos de la relación para contar con una clasificación para la tupla completa. Así, un esquema de relación **multinivel** R con n atributos se representaría como:

$$R(A_1, C_1, A_2, C_2, \dots, A_n, C_n, CT)$$

donde cada C_i representa el atributo de clasificación asociado al atributo A_i .

El valor del atributo CT de cada tupla provee una clasificación general para la tupla misma, en tanto que cada C_i provee una clasificación de seguridad más fina para cada valor de atributo dentro de la tupla. El valor de CT dentro de una tupla t deberá ser el **más alto** de todos los valores de los atributos de clasificación dentro de t . La **clave aparente** de una relación multinivel es el conjunto de atributos que habrían formado la clave primaria en una relación normal (de un solo nivel). A sujetos (usuarios) con diferentes niveles de clasificación les parecerá que una relación multinivel contiene diferentes datos. En algunos casos, es posible almacenar una sola tupla de la relación con un nivel de clasificación más alto y producir las tuplas correspondientes con una clasificación de nivel más bajo a través de un proceso llamado **filtración**. En otros casos, es necesario almacenar dos o más tuplas con diferentes niveles de clasificación pero con el mismo valor de la **clave aparente**. Esto da pie al concepto de **creación de múltiples ejemplares**, donde varias tuplas pueden tener el mismo valor de la clave aparente pero diferentes valores de atributos para usuarios con diferentes niveles de clasificación.

Ilustraremos estos conceptos con el ejemplo simple de relación multinivel de la figura 20.2(a), donde se muestran los valores de los atributos de clasificación junto al valor de cada atributo. Supongamos que el atributo Nombre es la clave aparente, y consideremos la consulta `SELECT * FROM EMPLEADO`. Un usuario con clasificación de seguridad S verá la misma relación que se muestra en la figura 20.2(a), ya que todas las clasificaciones de las tuplas son inferiores a S o iguales. Sin embargo, un usuario con clasificación de seguridad C no podría ver el valor de Salario de Bravo ni el valor de Rendimiento de Silva, pues tienen una clasificación más alta. Las tuplas se filtrarían, apareciendo como se muestra en la figura 20.2(b). En el caso de un usuario con clasificación de seguridad U , la filtración sólo permitirá que aparezca el atributo Nombre de Silva (Fig. 20.2(c)). Observe cómo la filtración introduce valores nulos para los valores de atributos cuya clasificación de seguridad es más alta que la del usuario.

En general, la regla de **integridad de entidades** para las relaciones multinivel establece que ningún atributo que sea miembro de la clave aparente podrá ser nulo, y que todos estos atributos deberán tener la **misma** clasificación de seguridad dentro de cada tupla individual. Por añadidura, todos los demás valores de atributos en la tupla deberán tener una clasificación de seguridad mayor que la de la clave o igual a ella. Esta restricción asegura que

(a)	EMPLEADO			
	Nombre	Salario	Rendimiento	CT
	Silva U Bravo C	40000 C 80000 S	Regular S Bueno C	S S
(b)	EMPLEADO			
	Nombre	Salario	Rendimiento	CT
	Silva U Bravo C	40000 C nulo S	nulo C Bueno C	C C
(c)	EMPLEADO			
	Nombre	Salario	Rendimiento	CT
	Silva U U	nulo U	nulo U	U
(d)	EMPLEADO			
	Nombre	Salario	Rendimiento	CT
	Silva U Silva U Bravo C	40000 C 40000 C 80000 S	Regular S Excelente C Bueno C	S C S

Figura 20.2 Una relación de múltiples niveles. (a) Las tuplas EMPLEADO originales. (b) Aspecto de EMPLEADO después de filtrar los usuarios de clasificación C. (c) Aspecto de EMPLEADO después de filtrar los usuarios de clasificación U. (d) Creación de múltiples ejemplares de la tupla Silva.

un usuario podrá ver la clave si puede ver alguna parte de la tupla. Otras reglas de integridad, llamadas **integridad de nulos e integridad entre ejemplares**, aseguran informalmente que, si un valor de tupla con un cierto nivel de seguridad se puede filtrar (derivar) de una tupla con más alta clasificación, bastará con almacenar la tupla de más alta clasificación en la relación multinivel.

A fin de ilustrar la creación de múltiples ejemplares, supongamos que un usuario con *clasificación de seguridad* C trata de actualizar el valor de Rendimiento de Silva, cambiándolo a "Excelente"; esto corresponde a la siguiente consulta en SQL:

```
UPDATE EMPLEADO
SET Rendimiento = "Excelente"
WHERE Nombre = "Silva";
```

Como la vista que se presenta a los usuarios con clasificación de seguridad C (Fig. 20.2(b)) permite tal actualización, el sistema no deberá rechazarla; si así lo hiciera, el usuario podría inferir que existe algún valor no nulo para el atributo Rendimiento de Silva, y no el valor nulo que aparece. Esto es un ejemplo de inferencia de información a través de lo que se conoce como **canal furtivo**, cosa que no debe permitirse en los sistemas muy seguros.

No obstante, el usuario no deberá poder sobrescribir el valor que Rendimiento tiene en el nivel de clasificación superior. La solución es crear *múltiples ejemplares* de la tupla Silva en el nivel de clasificación inferior, C, como se muestra en la figura 20.2(d). Esto es necesario porque la nueva tupla no se puede filtrar a partir de la tupla existente con clasificación S.

Las operaciones básicas de actualización del modelo relacional (insertar, eliminar, modificar) se deben cambiar para manejar ésta y otras situaciones similares, pero este aspecto del problema rebasa el alcance de nuestra exposición. El lector interesado puede consultar la bibliografía con que se cierra el capítulo si desea mayores detalles.

Para poder lograr un sistema verdaderamente seguro, el Departamento de la Defensa (DoD) de Estados Unidos ha creado escalas para los **niveles de seguridad** que alcanzan diversos sistemas. Estos niveles se denominan A1, B3, B2, B1, C2, C1 y D, siendo A1 el más seguro y D el que lo es menos. Los sistemas de nivel C1 y C2 deben ofrecer control de acceso discrecional, y los de nivel B1 deben ofrecer además control de acceso obligatorio. Los sistemas de niveles superiores son seguros contra canales furtivos, y los sistemas A1 deben proveer seguridad verificable. Se ha propuesto una arquitectura de implementación con un **kernel de seguridad** (o **monitor de referencia**) —que se encarga de todas las acciones de seguridad básicas pero que es lo bastante pequeño como para verificar los niveles de seguridad alcanzados— para garantizar el concepto; se denomina **base de cómputo de confianza** (TCB: *trusted computing base*).

20.4 Seguridad de bases de datos estadísticas *

Las bases de datos estadísticas sirven principalmente para producir cifras estadísticas sobre diversas poblaciones. Pueden contener datos confidenciales sobre muchos individuos, y estos datos deben protegerse para que los usuarios no tengan acceso a ellos. Sin embargo, hay usuarios que están autorizados para obtener información estadística sobre las poblaciones, como medias, conteos, sumas y desviaciones estándar. Las técnicas creadas para proteger la confidencialidad de la información individual en las bases de datos estadísticas quedan fuera del alcance de este libro; sólo ilustraremos de manera muy breve el problema con un ejemplo simple. El lector interesado puede consultar la bibliografía para encontrar textos que ofrecen un análisis completo de las bases de datos estadísticas y de su seguridad. Ilustraremos el problema con la relación PERSONA que se muestra en la figura 20.3, con atributos NOMBRE, NSS, INGRESO, DOMICILIO, CIUDAD, ESTADO, CP, SEXO y ÚLTIMO_GRADO.

Una **población** es un conjunto de tuplas de un archivo que satisfacen alguna condición de selección. Así pues, cada condición de selección sobre la relación PERSONA especificará una determinada población de tuplas PERSONA. Por ejemplo, la condición SEXO = 'M' especifica la población masculina, la condición (SEXO = 'F' AND (ÚLTIMO_GRADO = 'MC' OR ÚLTIMO_GRADO = 'DR')) especifica la población femenina que tiene un grado de maestro en ciencias o de doctor como grado más alto, y la condición CIUDAD = 'Higueras' especifica la población que vive en Higueras.

PERSONA

NOMBRE	NSS	INGRESO	DIRECCIÓN	CIUDAD	ESTADO	CP	SEXO	ÚLTIMO_GRADO
--------	-----	---------	-----------	--------	--------	----	------	--------------

Figura 20.3 La relación PERSONA.

En las consultas estadísticas se aplican funciones estadísticas a una población de tuplas. Por ejemplo, podríamos querer obtener el número de individuos que tiene una población o el ingreso medio de la población. Sin embargo, los usuarios estadísticos no tienen permiso de obtener datos individuales, como el ingreso de una persona específica. Las técnicas de **seguridad de bases de datos estadísticas** deben prohibir la obtención de datos individuales. Para controlar esto, se prohíbe que las consultas obtengan valores de los atributos y sólo se permiten las consultas que contengan funciones agregadas estadísticas como COUNT, SUM, MIN, MAX, AVERAGE y STANDARD DEVIATION. Tales consultas suelen recibir el nombre de **consultas estadísticas**.

En algunos casos es posible **deducir** los valores de tuplas individuales a partir de una secuencia de consultas estadísticas. Esto sucede sobre todo cuando las condiciones producen una población que contiene muy pocas tuplas. Como ilustración, supongamos que usamos estas dos consultas estadísticas:

C1: SELECT COUNT(*) FROM PERSONA
WHERE <condición>

C2: SELECT AVERAGE(INGRESO) FROM PERSONA
WHERE <condición>

Supongamos ahora que nos interesa conocer el SALARIO de 'Rebeca Silva', y sabemos que tiene un doctorado y que vive en la ciudad de Belén, Estado de México. Emitiremos la consulta estadística C1 con la siguiente condición:

(ÚLTIMO_GRADO='DR' AND SEXO='F' AND CIUDAD='Belén' AND ESTADO='México')

Si obtenemos un resultado de 1 para esta consulta, podemos emitir C2 con la misma condición y obtener el ingreso de 'Rebeca Silva'. Incluso si con la condición anterior el resultado de C1 no es 1 pero sí un número pequeño —digamos, 2 o 3—, podemos emitir consultas estadísticas usando las funciones MAX, MIN y AVERAGE para identificar el posible intervalo de valores para el INGRESO de 'Rebeca Silva'.

La posibilidad de deducir información individual a partir de consultas estadísticas se reduce si no se permiten dichas consultas en los casos en que el número de tuplas de la población especificada por la condición de selección sea menor que cierto umbral. Otra técnica para prohibir la obtención de información individual es prohibir secuencias de consultas que hagan referencia repetidamente a la misma población de tuplas. También es posible introducir deliberadamente pequeñas inexactitudes o "ruido" en los resultados de las consultas estadísticas, a fin de dificultar la deducción de información individual a partir de los resultados. El lector interesado puede consultar la bibliografía si desea un análisis de estas técnicas.

20.5 Resumen

En este capítulo estudiamos varias técnicas para imponer seguridad en los sistemas de bases de datos. La imposición de la seguridad implica controlar el acceso a la base de datos como un todo y controlar la autorización para tener acceso a porciones específicas de una base de datos. Lo primero suele hacerse asignando cuentas con contraseña a los usuarios. Lo segundo puede lograrse empleando un sistema de concesión y revocación de privilegios a

cuentas individuales para tener acceso a partes específicas de la base de datos. Esta estrategia, generalmente denominada *seguridad discrecional*, se trató en la sección 20.2. Vimos algunas órdenes de SQL para otorgar y revocar privilegios, y se ilustró su uso con ejemplos. Después, en la sección 20.3, presentamos un panorama sobre los mecanismos de seguridad obligatoria que imponen la seguridad multinivel. Éstos requieren la clasificación de los usuarios y de los valores de los datos en clases de seguridad, e imponen las reglas que prohíben el flujo de información a niveles de seguridad más bajos. Examinamos algunos de los conceptos clave en los que se basa el modelo relacional multinivel, como la filtración y la creación de múltiples ejemplares. Por último, en la sección 20.4, analizamos brevemente el problema de controlar el acceso a las bases de datos estadísticas, a fin de proteger la confidencialidad de la información individual y, al mismo tiempo, proporcionar acceso estadístico a poblaciones de registros.

Preguntas de repaso

- 20.1. Analice el significado de los siguientes términos: *autorización de base de datos*, *control de acceso*, *cifrado de datos*, *cuenta privilegiada (del sistema)*, *auditoría de base de datos*, *registro de intervenciones*.
- 20.2. Explique qué son los tipos de privilegios en el nivel de cuenta y en el nivel de relación.
- 20.3. ¿Cuál cuenta se designa como propietario de una relación? ¿Qué privilegios tiene el propietario de una relación?
- 20.4. ¿Cómo se usa el mecanismo de vistas como mecanismo de autorización?
- 20.5. ¿Qué significa otorgar un privilegio?
- 20.6. ¿Qué significa revocar un privilegio?
- 20.7. Analice el sistema de propagación de privilegios y las restricciones que imponen los límites de propagación horizontal y vertical.
- 20.8. Mencione los tipos de privilegios disponibles en SQL.
- 20.9. ¿Qué diferencia hay entre el control de acceso discrecional y el obligatorio?
- 20.10. ¿Cuáles son las clasificaciones de seguridad más comunes? Analice la propiedad de seguridad simple y la propiedad *, y explique la justificación de estas reglas para imponer la seguridad multinivel.
- 20.11. Describa el modelo relacional multinivel. Defina los siguientes términos: *clave primaria*, *creación de múltiples ejemplares*, *filtración*.
- 20.12. ¿Qué es una base de datos estadística? Analice el problema de la seguridad de bases de datos estadísticas.

Ejercicios

- 20.13. Considere el esquema de base de datos relacional de la figura 6.5. Suponga que el usuario X creó todas las relaciones (y por tanto es su propietario), y X desea otorgar los siguientes privilegios a las cuentas de usuario A, B, C, D y E:

- a. La cuenta A puede leer o modificar cualquier relación excepto DEPENDIENTE y puede otorgar cualquiera de estos privilegios a otros usuarios.
 - b. La cuenta B puede leer todos los atributos de EMPLEADO y de DEPARTAMENTO, excepto SALARIO, NSSGTE y FECHAINICGTE.
 - c. La cuenta C puede leer o modificar TRABAJA_EN pero sólo puede leer los atributos NOMBREP, INIC, APELLIDO y NSS de EMPLEADO, y los atributos NOMBREPR y NÚMERO de PROYECTO.
 - d. La cuenta D puede leer cualquier atributo de EMPLEADO o de DEPENDIENTE, y puede modificar DEPENDIENTE.
 - e. La cuenta E puede leer cualquier atributo de EMPLEADO pero sólo en las tuplas de EMPLEADO en las que ND = 3.
- Escriba enunciados en SQL para otorgar estos privilegios. Utilice vistas en los casos apropiados.
- 20.14. Suponga que se va a otorgar el privilegio a del ejercicio 20.13 con GRANT OPTION, pero con la condición de que la cuenta A pueda otorgarlo a cuando más cinco cuentas, y que cada una de estas cuentas pueda propagar el privilegio a otras cuentas pero sin la GRANT OPTION. ¿Cuáles serían los límites de propagación horizontal y vertical en este caso?
 - 20.15. Considere la relación de la figura 20.2(d). ¿Cómo la vería un usuario con clasificación U? Suponga que un usuario con clasificación U trata de cambiar el salario de "Silva" a \$50 000; ¿cuál sería el resultado de esta acción?

Bibliografía selecta

- La autorización basada en la concesión y revocación de privilegios se propuso para el SGBD experimental System R y se presenta en Griffiths y Wade (1976). Varios libros tratan la seguridad de las bases de datos y de los sistemas de cómputo en general, como los de Leiss (1982a) y Fernandez et al. (1981). Denning y Denning (1979) es un artículo didáctico sobre seguridad de los datos.
- Muchos artículos analizan diferentes técnicas para diseñar y proteger bases de datos estadísticas. Entre ellos están McLeish (1989), Chin y Ozsoyoglu (1981), Leiss (1982), Wong (1984) y Denning (1980). Ghosh (1984) estudia el empleo de bases de datos estadísticas para el control de calidad. También hay muchos artículos que analizan la criptografía y el cifrado de datos, como Diffie y Hellman (1979), Rivest et al. (1978) y Akl (1983).
- La seguridad multinivel se examina en Jajodia y Sandhu (1991), Denning et al. (1987), Smith y Winslett (1992), Stachour y Thuraisingham (1990) y Lunt et al. (1990). Lunt y Fernandez (1990) y Jajodia y Sandhu (1990) presentan panoramas sobre los temas de investigación en torno a la seguridad en las bases de datos. Los efectos de la seguridad multinivel sobre el control de concurrencia se analizan en Kogan y Jajodia (1990).
- En fechas recientes se han publicado varios trabajos que estudian la seguridad en bases de datos de la siguiente generación, semánticas y orientadas a objetos (véase el Cap. 22), como Bertino (1992), Rabbitti et al. (1991) y Smith (1990).

Conceptos avanzados de modelado de datos

Los conceptos de modelado ER que vimos en el capítulo 3 bastan para representar muchos esquemas de base de datos en las aplicaciones tradicionales, entre las que destacan sobre todo las aplicaciones de procesamiento de datos en los negocios y la industria. Sin embargo, desde finales de los años setenta se han generalizado nuevas aplicaciones de estas tecnologías; entre éstas: las bases de datos de diseño para ingeniería (CAD/CAM)[†], bases de datos de imágenes y gráficos, bases de datos cartográficas y geológicas, bases de datos multimedia^{††} y bases de conocimientos para aplicaciones de inteligencia artificial. Estos tipos de bases de datos tienen requerimientos más complejos que las aplicaciones tradicionales. A fin de presentar estos requerimientos de la manera más exacta y explícita posible, los diseñadores deben utilizar conceptos adicionales de modelado "semántico". Se han propuesto varios modelos semánticos de datos en la literatura. En este capítulo describiremos muchas de las características que se han incorporado en estos modelos. Comenzamos en la sección 21.1 mejorando el modelo ER (véase el Cap. 3) con conceptos adicionales de especialización, generalización, herencia y categorías, dando lugar al modelo ER *extendido* o modelo EER (*enhanced entity-relationship*). Después de presentar los conceptos del modelo EER, mostramos en la sección 21.2 cómo se puede establecer una transformación de estos conceptos al modelo relacional; esto constituye una ampliación del análisis de la transformación de los conceptos en el modelo ER normal que hicimos en la sección 6.8. En la sección 21.3 veremos las abstracciones fundamentales en que se apoyan muchos de los modelos de datos semánticos. En la sección 21.4 clasificaremos los diferentes tipos de restricciones de integridad que se usan en el modelado de datos. La sección 21.5 describe las operaciones del modelo EER

[†]CAD/CAM es la abreviatura en inglés de diseño asistido por computador/fabricación asistida por computador.

^{††}Las bases de datos multimedia almacenan datos que representan entidades tradicionales, además de datos no estructurados como texto, imágenes y grabaciones de voz.

y muestra cómo pueden usarse para diseñar conceptualmente las transacciones de una aplicación de bases de datos. Por último, la sección 21.6 presenta un bosquejo breve de otros modelos de datos conceptuales, los modelos funcional, relacional anidado, estructural y semántico.

Si lo desea, el lector puede pasar por alto algunas de las secciones finales de este capítulo (Secs. 21.2 a 21.6), o todas. Además, la temática de la sección 21.1 puede cubrirse inmediatamente después de completar el capítulo 3, si se prefiere.

21.1 Conceptos del modelo ER extendido (EER)

El modelo EER abarca todos los conceptos de modelado del modelo ER que se presentaron en el capítulo 3. Además, incluye los conceptos de **subclase** y **superclase** y los conceptos relacionados de **especialización** y **generalización**. Otro concepto que tiene el modelo EER es el de **categoría**. Asociado a estos conceptos está el importante mecanismo de **herencia de atributos**. Desafortunadamente, no existe una terminología estándar para estos conceptos, de modo que utilizaremos los términos de uso más frecuente. La terminología alternativa se dará en notas a pie de página. También describiremos una técnica de diagramación para mostrar estos conceptos cuando aparecen en un esquema EER. A los diagramas de esquema resultantes los llamamos **diagramas ER-extendido** o **diagramas EER**.

21.1.1 Subclases, superclases y especialización

Subclases y superclases. El primer concepto del modelo EER que trataremos es el de subclase de un tipo de entidades. Como vimos en el capítulo 3, un tipo de entidades sirve para representar un conjunto de entidades del mismo tipo, como el conjunto de entidades EMPLEADO en la base de datos de una compañía. En muchos casos, un tipo de entidades tiene varias subagrupaciones adicionales de sus entidades que son significativas y que deben representarse explícitamente por su importancia para la aplicación de base de datos. Por ejemplo, las entidades que son miembros del tipo de entidades EMPLEADO pueden agruparse en entidades SECRETARIA, INGENIERO, GERENTE, TÉCNICO, EMPLEADO_ASALARIADO, EMPLEADO_POR_HORA, etc. El conjunto de entidades de cada una de estas agrupaciones es un subconjunto de las entidades que pertenecen al tipo EMPLEADO, lo que significa que toda entidad que sea miembro de una de estas subagrupaciones también será un empleado. Cada una de estas subagrupaciones es una **subclase** del tipo de entidades EMPLEADO, y EMPLEADO es la **superclase** de cada una de estas subclases.

Llamamos a la relación entre una superclase y cualquiera de sus subclases un **vínculo superclase/subclase**, o simplemente **vínculo clase/subclase**.[‡] En nuestro ejemplo anterior, EMPLEADO/SECRETARIA y EMPLEADO/TÉCNICO son dos vínculos clase/subclase. Observe que un ejemplar de entidad miembro de la subclase representa a la *misma entidad del mundo real* que algún miembro de la superclase; por ejemplo, una entidad SECRETARIA 'Juana Luján' es también el EMPLEADO 'Juana Luján'. Así pues, el miembro de la subclase es igual a la entidad de la superclase, pero tiene un *papel específico* distinto. Cuando implementamos un vínculo de la superclase/subclase en un sistema de base de datos, empero, tal vez representemos un

[‡]Es común llamar a un vínculo clase/subclase un vínculo ES-UN (o ES-UNA), por la forma como nos referimos al concepto. Decimos "una SECRETARIA ES-UN EMPLEADO", "un TÉCNICO ES-UN EMPLEADO", etcétera.

miembro de la subclase como un objeto distinto de la base de datos; digamos, un registro distinto relacionado a través del atributo clave con su entidad de superclase. En la sección 21.2 analizaremos varias opciones para representar vínculos superclase/subclase en las bases de datos relacionales.

Una entidad no puede existir en la base de datos simplemente por ser miembro de una subclase; también debe ser miembro de la superclase. Una entidad así se puede incluir opcionalmente como miembro en cualquier cantidad de subclases. Por ejemplo, un empleado asalariado que también es un ingeniero pertenece a las dos subclases INGENIERO y EMPLEADO_ASALARIADO del tipo de entidades EMPLEADO. Sin embargo, no es necesario que toda entidad de una superclase sea miembro de alguna subclase.

Herencia de atributos en los vínculos superclase/subclase. Un importante concepto asociado a las subclases es el de **herencia de atributos**. Como una entidad de la subclase representa la misma entidad del mundo real de la superclase, debe poseer valores para sus atributos específicos además de valores de sus atributos como miembro de la superclase. Decimos que una entidad que es miembro de una subclase **hereda** todos los atributos de la entidad como miembro de la superclase. La entidad también hereda todos los ejemplares de vínculo de los tipos de vínculos en los que participa la superclase. Observe que una subclase, junto con todos los atributos que hereda de la superclase, es un *tipo de entidades* por derecho propio.

Especialización. La **especialización** es el proceso de definir un conjunto de subclases de un tipo de entidades; este tipo de entidades se denomina la **superclase** de la especialización. El conjunto de las subclases que forman una especialización se define a partir de alguna característica distintiva de las entidades de la superclase. Por ejemplo, el conjunto de subclases {SECRETARIA, INGENIERO, TÉCNICO} es una especialización de la superclase EMPLEADO que las distingue entre las entidades EMPLEADO según el *tipo de trabajo* de cada entidad. Podemos tener varias especializaciones del mismo tipo de entidades basadas en diferentes características distintivas. Por ejemplo, *otra especialización* del tipo EMPLEADO puede originar el conjunto de subclases {EMPLEADO_ASALARIADO, EMPLEADO_POR_HORA}; esta especialización distingue entre los empleados basándose en el *método de compensación*.

Diagramas ER-extendido (EER). La figura 21.1 muestra cómo representamos una especialización gráficamente en un **diagrama EER**. Las subclases que definen una especialización se conectan con líneas a un círculo, el cual se conecta a la superclase. El símbolo de subconjunto sobre la línea que conecta una subclase al círculo indica la dirección del vínculo superclase/subclase. Cualesquier atributos que se apliquen sólo a entidades de una determinada subclase —como RapidezTeclado de SECRETARIA— se conectan al rectángulo que representa esa subclase. Éstos se denominan **atributos específicos** de la subclase. De manera similar, una subclase puede participar en **tipos de vínculos específicos**, como la participación de EMPLEADO_POR_HORA en AFILIADO_A en la figura 21.1. Un poco más adelante explicaremos el símbolo **d** que aparece en los círculos de la figura 21.1 y otros aspectos de la notación de los diagramas EER.

La figura 21.2 muestra unos cuantos ejemplares de entidades que pertenecen a subclases de la especialización {SECRETARIA, INGENIERO, TÉCNICO}. Una vez más, observe que una entidad que pertenece a una subclase representa la *misma entidad del mundo real* que la entidad con la que está conectada en la superclase EMPLEADO, aunque la misma entidad se muestre dos veces; por ejemplo, en la figura 21.2 *e*, aparece tanto en EMPLEADO como en SECRETARIA.

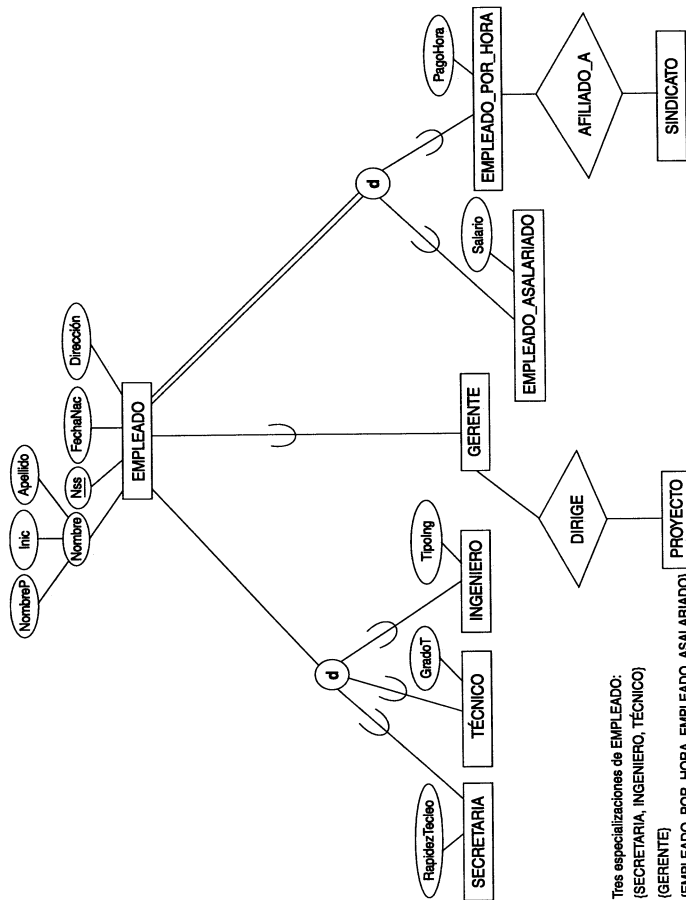


Figura 21.1 Diagrama EER para representar especialización y subclases.

Como lo sugiere esta figura, un vínculo superclase/subclase como EMPLEADO/SECRETARIA se parece un poco a un vínculo 1:1 en el nivel de ejemplares (véase la Fig. 3.12). La diferencia principal es que, en un vínculo 1:1, dos entidades distintas están relacionadas. Podemos considerar que una entidad de la subclase es lo mismo que la entidad de la superclase pero con un *papel especializado*; por ejemplo, un EMPLEADO especializado en el papel de SECRETARIA, o un EMPLEADO especializado en el papel de TÉCNICO.

Empleo de subclases en el modelado de datos. Hay dos razones principales para incluir vínculos clase/subclase en un modelo de datos. La primera es que ciertos atributos pueden aplicarse a algunas de las entidades del tipo de entidades (superclase), pero no a todas. Se define una subclase para agrupar las entidades a las que se aplican estos atributos. Los miembros de la subclase pueden compartir la mayor parte de sus atributos con los demás miembros de la superclase. Por ejemplo, la subclase SECRETARIA puede tener un atributo RapidezTeclado, en tanto que la subclase INGENIERO puede tener un atributo TipoIngeniero, pero SECRETARIA e INGENIERO comparten sus demás atributos como miembros del tipo EMPLEADO.

La segunda razón para usar subclases es que en algunos tipos de vínculos sólo pueden participar entidades que sean miembros de la subclase. Por ejemplo, si sólo los empleados a los que se les paga por hora pueden estar afiliados a un sindicato, podemos representar ese hecho creando la subclase EMPLEADOS_POR_HORA de EMPLEADO y relacionándola con un tipo de entidades SINDICATO a través del tipo de vínculos AFILIADO_A, como se ilustra en la figura 21.1.

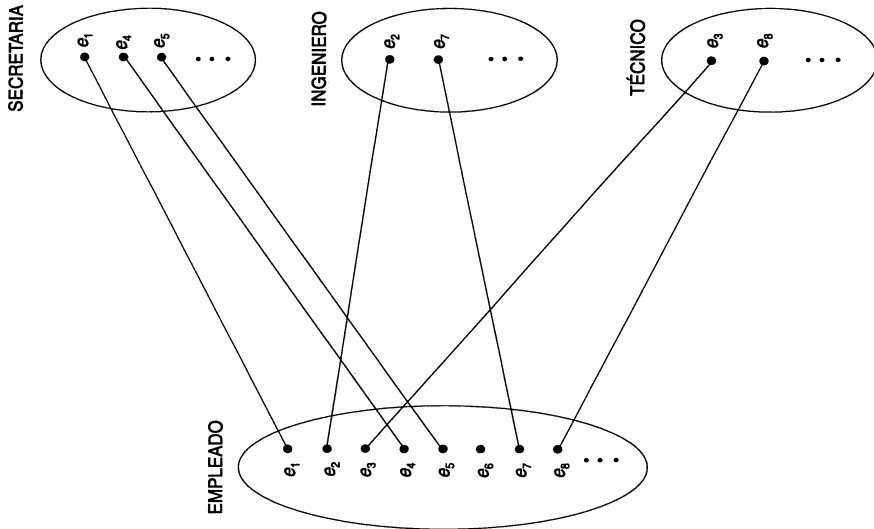


Figura 21.2 Algunos ejemplares de la especialización de EMPLEADO en el conjunto {SECRETARIA, INGENIERO, TÉCNICO} de subclases.

21.1.2 Generalización

El proceso de especialización examinado en la subsección anterior nos permite:

- Definir un conjunto de subclases de un tipo de entidades.
- Asociar atributos específicos adicionales a cada subclase.
- Establecer tipos de vínculos específicos adicionales entre cada subclase y otros tipos de entidades, u otras subclases.

Podemos concebir un *proceso inverso* de abstracción en el que suprimimos las diferencias entre varios tipos de entidades, identificamos sus rasgos comunes y los **generalizamos** para formar una sola **superclase** de la cual los tipos de entidades originales sean **subclases especiales**. Por ejemplo, consideremos los tipos de entidades COCHE y CAMIÓN de la figura 21.3(a); se

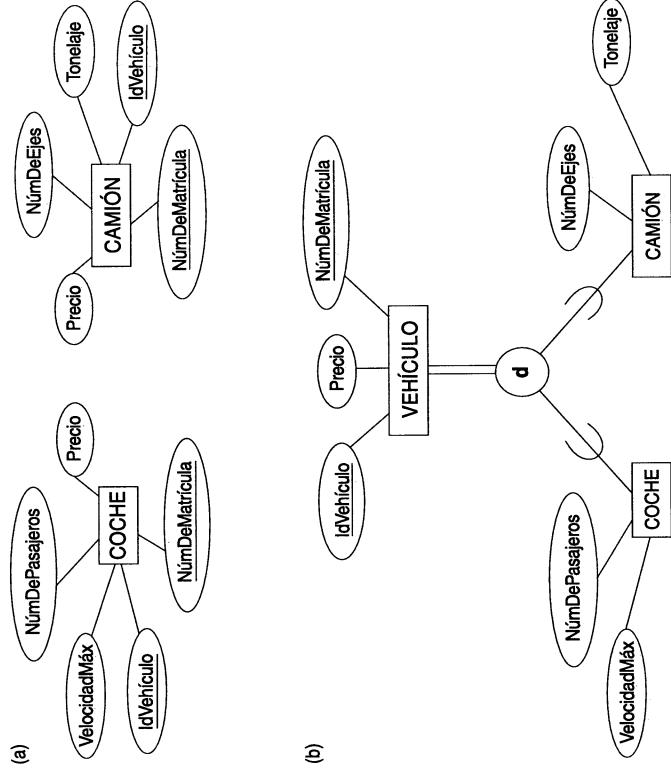


Figura 21.3 Ejemplos de generalización. (a) Dos tipos de entidades COCHE y CAMIÓN. (b) Generalización de COCHE y CAMIÓN en VEHÍCULO.

pueden generalizar en el tipo de entidades VEHÍCULO, como se aprecia en la figura 21.3(b). Tanto COCHE como CAMIÓN son ahora subclases de la **superclase generalizada** VEHÍCULO. Usamos el término **generalización** para referirnos al proceso de definir un tipo de entidades generalizado a partir de los tipos de entidades dados.

Observe que el proceso de generalización puede considerarse como el inverso funcional del proceso de especialización. Por ello, en la figura 21.3 podemos ver {COCHE, CAMIÓN} como una especialización de VEHÍCULO, en vez de ver VEHÍCULO como una generalización de COCHE y CAMIÓN. De manera similar, en la figura 21.1 podemos ver EMPLEADO como una generalización de SECRETARIA, TÉCNICO e INGENIERO. En la práctica a veces se usa una notación diagramática para distinguir entre generalización y especialización. Una flecha que apunta a la superclase generalizada representa una generalización, y las flechas que apuntan a las subclases especializadas representan una especialización. No usaremos esta notación porque la decisión respecto a cuál proceso es el más apropiado en una situación dada suele ser subjetiva. El apéndice A presenta algunas de las notaciones diagramáticas alternativas que se sugieren.

21.1.3 Modelado de datos con especialización y generalización

Ya hemos estudiado los conceptos de subclase y de vínculos superclase/subclase, así como los procesos de especialización y generalización. Por lo regular, una superclase o subclase

representa un conjunto de entidades y por tanto también es un *tipo de entidades*; es por ello que las superclases y subclases (al igual que los tipos de entidades) se representan como rectángulos en los diagramas EER. Ahora veremos con más detalle las propiedades de las especializaciones y generalizaciones.

Restricciones sobre la especialización y la generalización. En los párrafos siguientes examinaremos restricciones que se aplican a una sola especialización o a una sola generalización; pero para no hacer más largo el análisis, éste se referirá sólo a la especialización, aunque se aplique *tanto a la generalización como a la especialización*.

En general, puede haber varias especializaciones definidas sobre el mismo tipo de entidades (superclase), como en la figura 21.1. En un caso así, las entidades pueden pertenecer a subclases en cada una de las especializaciones. Sin embargo, una especialización puede consistir en una sola subclase, como la especialización {GERENTE} de la figura 21.1; en tal caso, no usaremos la notación del círculo.

En algunas especializaciones podemos determinar con exactitud las entidades que se convertirán en miembros de cada subclase, para lo cual especificamos una condición en términos del valor de algún atributo de la superclase. Tales subclases se llaman **subclases definidas por predicado** (o **definidas por condición**). Por ejemplo, si el tipo de entidades EMPLEADO tiene un atributo TipoTrabajo, como se muestra en la figura 21.4, podemos especificar la condición de pertenencia a la subclase SECRETARIA mediante el predicado (TipoTrabajo = 'Secretaria'), al cual llamamos **predicado de definición** de la subclase. Esta condición es una *restricción* que especifica que los miembros de la subclase SECRETARIA deben satisfacer el predicado y que todas las entidades del tipo EMPLEADO cuyo valor para el atributo TipoTrabajo sea 'Secretaria' deben pertenecer a la subclase. Para representar las subclases definidas por predicado escribimos la condición del predicado junto a la línea que conecta la subclase a su superclase.

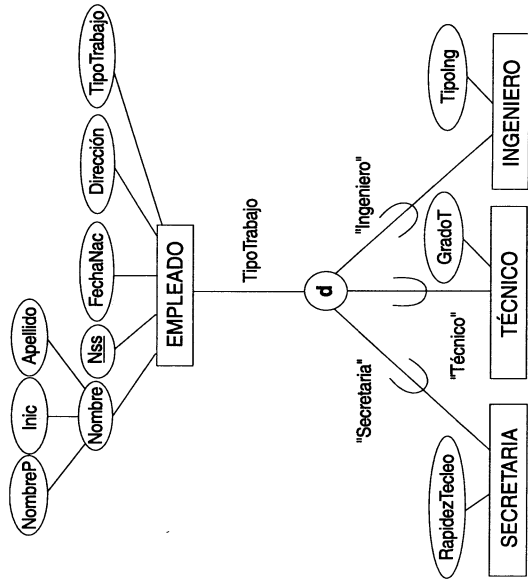


Figura 21.4 Especialización definida por atributo sobre el atributo TipoTrabajo de EMPLEADO.

Si la condición de pertenencia de *todas las subclases* de una especialización están definidas en términos del *mismo atributo* de la superclase, se dice que la especialización misma es una **especialización definida por atributo**, y el atributo se denomina **atributo de definición** de la especialización. Las especializaciones definidas por atributo se representan como en la figura 21.4, colocando el nombre del atributo de definición junto a la línea que va del círculo a la superclase.

Cuando no tenemos una condición que determine la pertenencia, se dice que la subclase está **definida por el usuario**. La pertenencia a tales subclases la determinan los usuarios de la base de datos cuando aplican la operación de añadir una entidad a la subclase; así pues, *el usuario especifica individualmente la pertenencia para cada entidad*; no la especifica una condición que pueda evaluarse automáticamente.

Se pueden aplicar otras restricciones a una especialización. La primera es la **restricción de disyunción**, que especifica que las subclases de una especialización deben ser disjuntas. Esto significa que una entidad puede ser miembro de *cualquier una* de las subclases de la especialización. Una especialización definida por atributo implica la restricción de disyunción si el atributo con que se define el predicado de pertenencia es monovaluado. La figura 21.4 muestra este caso, donde la **d** dentro del círculo significa disyunción. También usamos la notación **d** para especificar la restricción de que las subclases de una especialización definida por el usuario deben ser disjuntas, como se ilustra con la especialización {EMPLEADO_ASALARIADO, EMPLEADO_POR_HORA} de la figura 21.1. Si las subclases no son disjuntas, sus conjuntos de entidades pueden **traslaparse**; esto es, la misma entidad puede ser miembro de más de una subclase de la especialización. Éste, que es el caso por omisión, se indica colocando una **o** en el círculo, como en el ejemplo de la figura 21.5.

La segunda restricción sobre la especialización se denomina **restricción de completación**, la cual puede ser total o parcial. Una restricción de **especialización total** especifica que toda entidad de la superclase debe ser miembro de alguna subclase de la especialización. Por ejemplo, si todo EMPLEADO debe ser un EMPLEADO_POR_HORA o bien un EMPLEADO_ASALARIADO, la especialización {EMPLEADO_POR_HORA, EMPLEADO_ASALARIADO} de la figura 21.1 es una

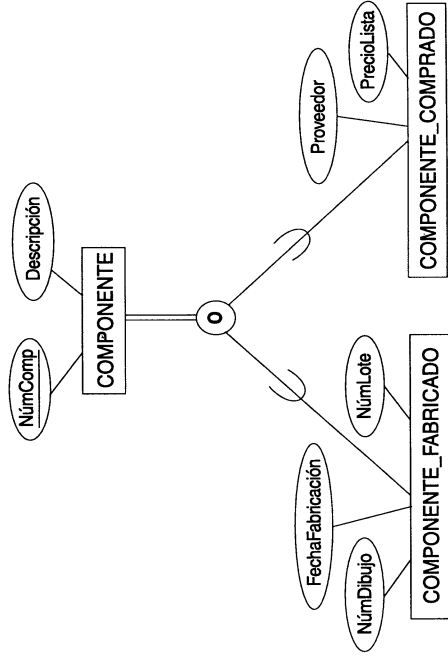


Figura 21.5 Especialización con subclases no disjuntas (traslapadas).

especialización total de EMPLEADO; eso se indica en los diagramas EER con una línea doble que conecte la superclase al círculo. Se usa una línea sencilla para indicar una **especialización parcial**, que permite que una entidad no pertenezca a ninguna de las subclases. Por ejemplo, si algunas entidades EMPLEADO no pertenecen a ninguna de las subclases {SECRETARIA, INGENIERO, TÉCNICO} de las figuras 21.1 y 21.4, esa especialización será parcial. Esta notación es similar a la de participación total de un tipo de entidades en un tipo de vínculos en el modelo ER que presentamos en el capítulo 3. Observe que las restricciones de disyunción y de completación son *independientes*. Así pues, tenemos estos cuatro tipos de especialización:

- disyunta, total
- disyunta, parcial
- traslapada, total
- traslapada, parcial

Desde luego, la restricción correcta la determina el significado que cada especialización tenga en el mundo real. Sin embargo, una superclase de generalización suele ser **total**, porque la superclase se *deriva* de las subclases y por tanto contiene sólo las entidades que están en las subclases.

Reglas de inserción y eliminación para la especialización y la generalización. Ciertas reglas de inserción y eliminación se aplican a la especialización (y a la generalización) como consecuencia de las restricciones que acabamos de especificar. Algunas de ellas son:

- La eliminación de una entidad de una superclase implica que automáticamente se le elimina de todas las subclases a las que pertenece.
- La inserción de una entidad en una superclase implica que la entidad se inserta por fuerza en todas las subclases *definidas por predicado* para las cuales la entidad satisficce el predicado de definición.
- La inserción de una entidad en una superclase de una *especialización total* implica que la entidad se insertará por fuerza en por lo menos una de las subclases de la especialización.

Se sugiere al lector que elabore una lista completa de reglas de inserción y de eliminación para los diversos tipos de especializaciones.

Jerarquías de especialización, retículas de especialización y herencia múltiple. Es posible especificar subclases de una subclase, formando una jerarquía o retícula de especializaciones. Por ejemplo, en la figura 21.6 INGENIERO es una subclase de EMPLEADO y también es superclase de GERENTE_DE_INGENIERÍA; esto representa la restricción del mundo real según la cual todo gerente de ingeniería debe ser un ingeniero. Una **jerarquía de especialización** tiene la restricción de que toda subclase participa (como subclase) en un *vínculo* clase/subclase; en contraste, en una **retícula de especialización** una subclase puede ser subclase en *más de un vínculo* clase/subclase. Por tanto, la figura 21.6 es una retícula.

La figura 21.7 muestra otra retícula de especialización de más de un nivel. Ésta puede ser parte de un esquema conceptual de una base de datos UNIVERSIDAD. Observe que esta organización habría sido una jerarquía si no fuera por la subclase ESTUDIANTE_ASISTENTE, que es subclase en dos vínculos clase/subclase distintos. Todas las entidades persona representadas en la base de datos son miembros del tipo de entidades PERSONA, que se especializa para

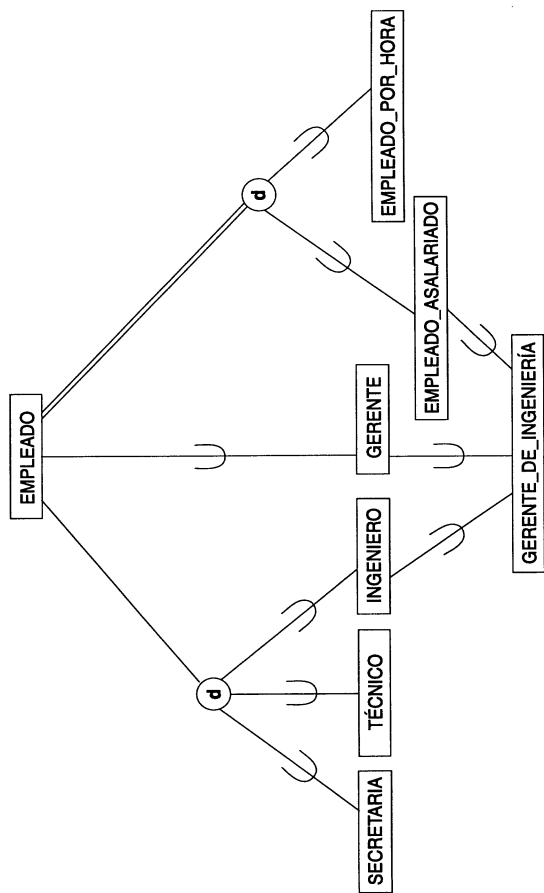


Figura 21.6 Retícula de especialización con la subclase compartida GERENTE_DE_INGENIERÍA.

dar las subclases {EMPLEADO, EXALUMNO, ESTUDIANTE}. Esta especialización es traslapada; por ejemplo, un exalumno también puede ser un empleado, o un estudiante que busca un grado avanzado después de recibir la licenciatura. La subclase ESTUDIANTE es superclase de la especialización {ESTUDIANTE_POSGRADO, ESTUDIANTE_LICENCIATURA}, mientras que EMPLEADO es superclase de la especialización {ESTUDIANTE_ASISTENTE, PERSONAL, PROFESORADO}. Observe que ESTUDIANTE_ASISTENTE también es una subclase de ESTUDIANTE. Por último, ESTUDIANTE_ASISTENTE es superclase de la especialización {ASISTENTE_INVESTIGADOR, ASISTENTE_DOCENTE}.

En una retícula o jerarquía de especialización como ésta, una subclase hereda los atributos no sólo de su superclase directa, sino también de todas sus superclases predecesoras, *incluida la raíz*. Por ejemplo, una entidad de ESTUDIANTE_POSGRADO hereda todos los valores de atributos de esa entidad como ESTUDIANTE y como PERSONA. Advierta que una entidad puede existir en varios nodos hoja de la jerarquía; por ejemplo, un miembro de ESTUDIANTE_POSGRADO puede ser también miembro de ASISTENTE_INVESTIGADOR.

Una subclase con *más de una* superclase se denomina **subclase compartida**. Por ejemplo, si todo GERENTE_DE_INGENIERÍA debe ser un INGENIERO pero también debe ser un EMPLEADO_ASALARIADO y un GERENTE, entonces GERENTE_DE_INGENIERÍA debería ser una subclase compartida de las tres superclases (Fig. 21.6). Esto lleva al concepto denominado **herencia múltiple**, ya que la subclase compartida GERENTE_DE_INGENIERÍA hereda directamente atributos y vínculos de múltiples subclases. Observe que las subclases compartidas dan origen a una retícula; si no existieran subclases compartidas, tendríamos una jerarquía en vez de una retícula.

Aunque hemos usado la especialización para ilustrar nuestro análisis, conceptos similares se aplican de *igual manera* a la generalización, como mencionamos al principio de esta subsección. Así, podemos hablar también de **jerarquías de generalización y retículas de**

generalización. En la siguiente subsección hablaremos más sobre las diferencias entre los procesos de especialización y de generalización.

Diseño conceptual descendente y ascendente. En el proceso de especialización, por lo regular comenzamos con un tipo de entidades y luego definimos subclases de él por especialización

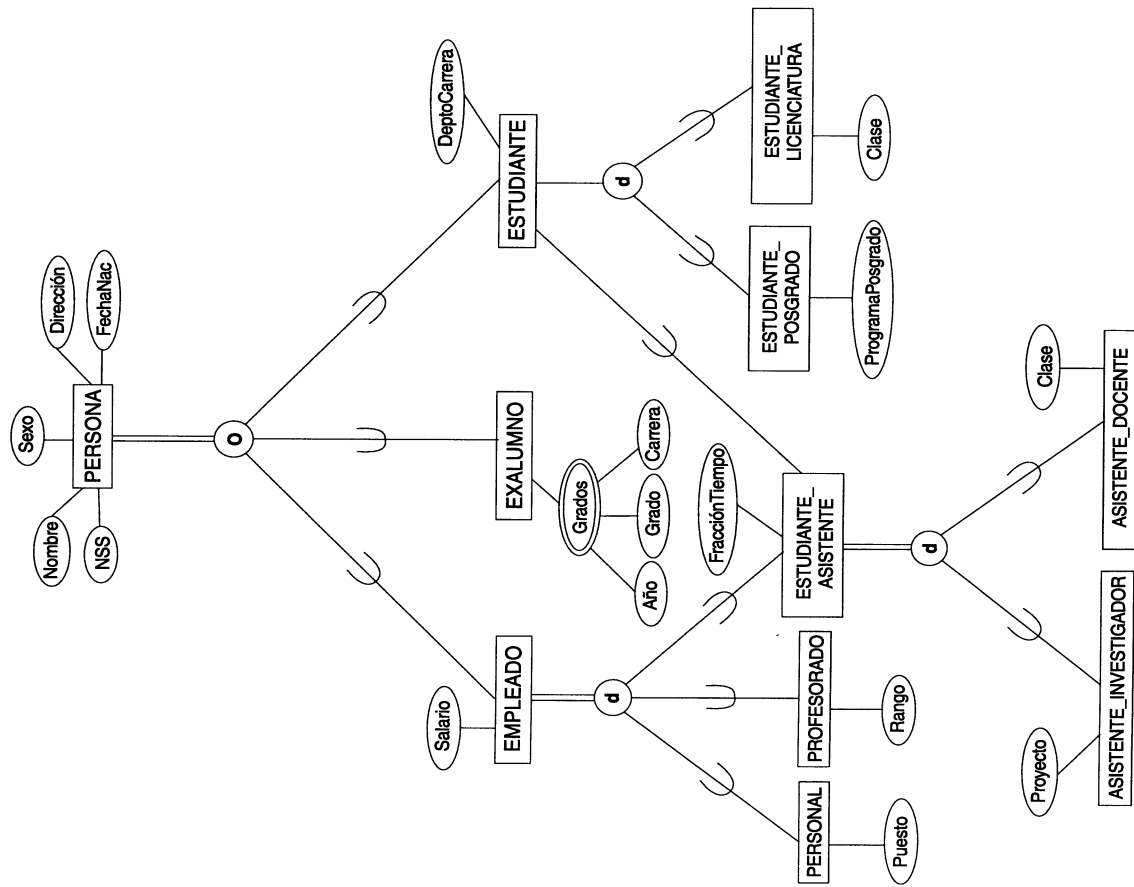


Figura 21.7 Retícula de especialización para una base de datos UNIVERSIDAD.

sucesiva; esto es, definimos repetidamente agrupaciones más específicas del tipo de entidades. Por ejemplo, al diseñar la retícula de especialización de la figura 21.7, podríamos primero especificar un tipo de entidades PERSONA para una base de datos universitaria. Luego descubrimos que se representarán tres tipos de personas en la base de datos: EMPLEADO, EXALUMNO, ESTUDIANTE para este fin, y escogemos la especialización traslapada porque una persona puede pertenecer a más de una de las subclases. En seguida especializamos EMPLEADO a {PERSONAL, PROFESORADO, ESTUDIANTE_ASISTENTE} y especializamos ESTUDIANTE a {ESTUDIANTE_POSGRADO, ESTUDIANTE_LICENCIATURA}. Por último, especializamos ESTUDIANTE a {ESTUDIANTE_ASISTENTE, ASISTENTE_INVESTIGADOR, ASISTENTE_DOCENTE}. Esta especialización sucesiva corresponde a un proceso de **refinación conceptual descendente** durante el diseño conceptual del esquema. Hasta aquí, tenemos una jerarquía; luego descubrimos que ESTUDIANTE_ASISTENTE es una subclase compartida, ya que también es una subclase de ESTUDIANTE, lo que da lugar a la retícula.

Es posible llegar a la misma jerarquía o retícula desde la dirección opuesta. En un caso así, el proceso implica generalización más que especialización y corresponde a una **síntesis conceptual ascendente**. En términos estructurales, las jerarquías o retículas que resultan de cualquiera de los dos procesos pueden ser idénticas; la única diferencia tiene que ver con la manera o el orden en que se especificaron las superclases y subclases del esquema.

En la práctica, es probable que no se sigan estrictamente ni el proceso de generalización ni el de especialización, sino que se utilice una combinación de ambos. En tal caso, se incorporan continuamente nuevas clases en una jerarquía o retícula conforme se hacen obvias para los usuarios y diseñadores. Cabe señalar que la noción de representar los datos y conocimientos mediante jerarquías y retículas de superclases/subclases es muy común en los sistemas basados en conocimientos y en los sistemas expertos, que combinan la tecnología de bases de datos con técnicas de inteligencia artificial. Por ejemplo, los esquemas de representación de conocimientos basados en marcos se parecen mucho a las jerarquías de clases.

21.1.4 Categorías y categorización*

Todos los vínculos superclase/subclase que hemos visto hasta ahora tienen *una sola superclase*. Incluso una subclase compartida como GERENTE DE INGENIERÍA en la retícula de la figura 21.6 es subclase en tres vínculos superclase/subclase *distintos*, y cada uno de los vínculos tiene *una sola superclase*. Sin embargo, en algunos casos es necesario modelar un solo vínculo superclase/subclase con *más de una superclase*, donde las superclases representan diferentes tipos de entidades. En este caso decimos que la *subclase* es una **categoría**.¹

Por ejemplo, supongamos que tenemos tres tipos de entidades: PERSONA, BANCO y COMPANHIA. En una base de datos para registro de vehículos, el dueño de un vehículo puede ser una persona, un banco (con un embargo preventivo sobre un vehículo) o una compañía. Necesitamos crear una clase que incluya entidades de los tres tipos para desempeñar el papel de dueño del vehículo. Para este fin se crea una categoría DUEÑO que es una *subclase de la unión* de las tres clases PERSONA, BANCO y COMPANHIA. En los diagramas EER las categorías se representan como se muestra en la figura 21.8. Las superclases COMPANHIA, BANCO y PERSONA se

¹Nuestro empleo del término *categoría* se basa en el modelo ECR (Elmasri et al. 1985).

conectan al círculo con el símbolo **U**, que representa la operación de unión de conjuntos. Una arista con el símbolo de subconjunto conecta el círculo a la categoría (subclase) DUEÑO. Si se necesita un predicado de definición, se muestra junto a la línea que proviene de la superclase a la que se aplica el predicado. En la figura 21.8 tenemos dos categorías: DUEÑO, que es una subclase de la unión de PERSONA, BANCO y COMPAÑÍA; y VEHÍCULO_REGISTERADO, que es una subclase de la unión de COCHE y CAMIÓN.

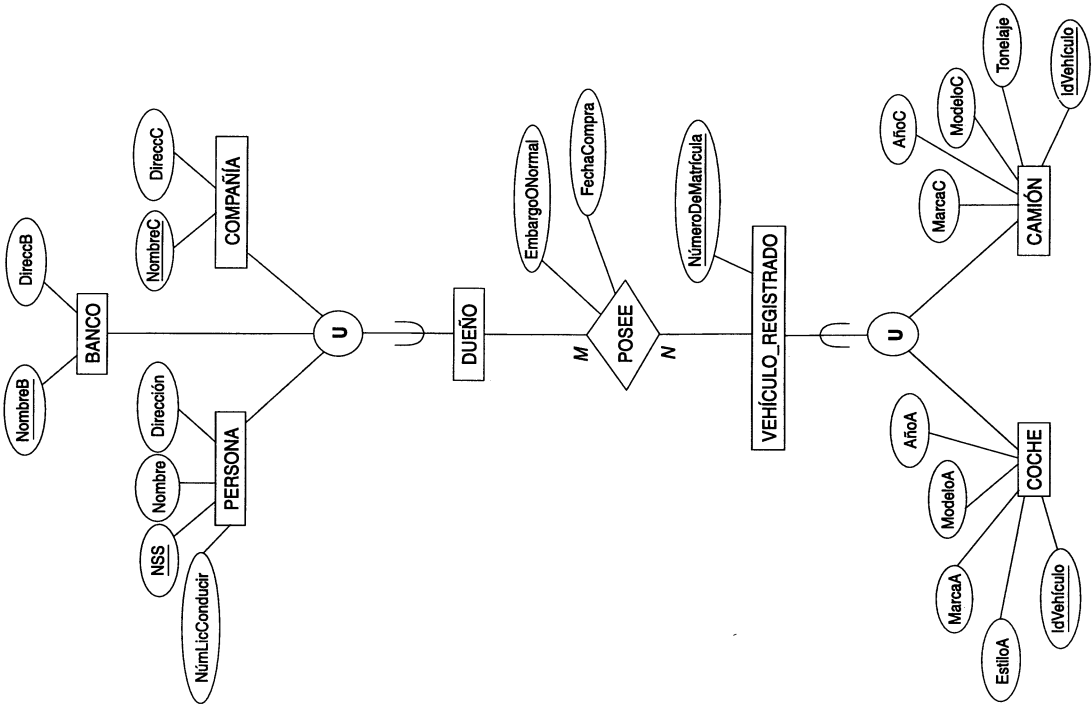


Figura 21.8 Dos categorías: DUEÑO y VEHÍCULO_REGISTERADO.

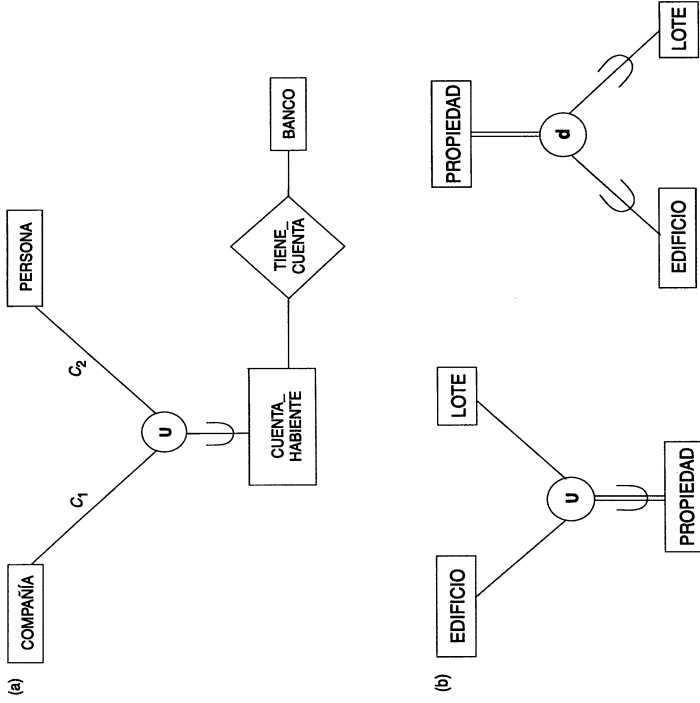


Figura 21.9 Categorías. (a) Categoría parcial CUENTAHABIENTE que es un subconjunto de la unión de dos tipos de entidades, COMPAÑÍA y PERSONA. (b) Categoría total PROPIEDAD y una generalización similar.

Una categoría tiene dos o más superclases que pueden representar distintos tipos de entidades, mientras que otros vínculos superclase/subclase siempre tienen una sola superclase. Podemos comparar una categoría, como DUEÑO en la figura 21.8, con la subclase comparada GERENTE_DE_INGENIERÍA de la figura 21.6. La segunda es una subclase de cada una de las tres superclases INGENIERO, GERENTE y EMPLEADO_ASALARIADO, así que una entidad que sea miembro de GERENTE_DE_INGENIERÍA deberá existir en las tres. Esto representa la restricción de que un gerente de ingeniería debe ser un INGENIERO, un GERENTE y un EMPLEADO_ASALARIADO; es decir, GERENTE_DE_INGENIERÍA es un subconjunto de la intersección de las tres superclases. Por otro lado, una categoría es un subconjunto de la unión de sus superclases. Por ello, una entidad que sea miembro de DUEÑO deberá existir en por lo menos una de las superclases pero no tiene que ser miembro de todas ellas. Esto representa la restricción de que un DUEÑO puede ser una COMPAÑÍA, un BANCO o una PERSONA en la figura 21.8. En este ejemplo, como en la mayor parte de los casos en los que se usan categorías, una entidad de la categoría es miembro de exactamente una de las superclases.

La herencia de atributos funciona de manera más selectiva en las categorías. Por ejemplo, en la figura 21.8 cada entidad DUEÑO hereda los atributos de una COMPAÑÍA, una PERSONA o un BANCO, dependiendo de la superclase a la que pertenezca la entidad. Esto se denomina

herencia selectiva. Por otro lado, una subclase compartida como GERENTE_DE_INGENIERÍA (Fig. 21.6) hereda todos los atributos de sus superclases EMPLEADO, ASALARIADO, INGENIERO y GERENTE.

Resulta interesante observar la diferencia entre la categoría VEHÍCULO_REGISTRADO (Fig. 21.8) y la superclase generalizada VEHÍCULO (Fig. 21.3(b)). En esta última, cada coche y cada camión es un VEHÍCULO; pero en la figura 21.8 la categoría VEHÍCULO_REGISTRADO contiene algunos coches y algunos camiones, pero no necesariamente todos ellos (por ejemplo, podría ser que algunos coches o camiones no estuvieran registrados). En general, una especialización o generalización como la de la figura 21.3(b), si fuera *parcial*, no impediría que VEHÍCULO contuviera otros tipos de entidades, como las motocicletas. Sin embargo, una categoría como VEHÍCULO_REGISTRADO en la figura 21.8 implica que sólo los coches y los camiones, y ningún otro tipo de entidades, pueden ser miembros de VEHÍCULO_REGISTRADO.

Una categoría puede ser *total* o *parcial*. Por ejemplo, CUENTAHABIENTE es una categoría parcial definida por predicado en la figura 21.9(a), donde c_1 y c_2 son condiciones de predicado que especifican cuáles entidades COMPañÍA y PERSONA, respectivamente, son miembros de CUENTAHABIENTE. No obstante, la categoría PROPIEDAD de la figura 21.9(b) es total porque todo edificio y lote debe ser miembro de ella; esto se indica con una línea doblada que conecta la categoría al círculo. Las categorías parciales se indican con una línea sencilla que conecta la categoría y el círculo, como en las figuras 21.8 y 21.9(a).

Las superclases de una categoría pueden tener diferentes atributos clave, como lo demuestra la categoría DUEÑO de la figura 21.8; o bien pueden tener el mismo atributo clave, como lo demuestra la categoría VEHÍCULO_REGISTRADO. Observe que, en el caso de una categoría *total* (no parcial), ésta puede representarse también como una especialización (o generalización), como se ilustra en la figura 21.9(b). En este caso la elección de qué representación utilizar es subjetiva. Si las dos clases representan el mismo tipo de entidades y comparten numerosos atributos, como los mismos atributos clave, es preferible la especialización/generalización; en caso contrario, la categorización es más apropiada.

21.1.5 Definiciones formales*

En las subsecciones precedentes hemos extendido el modelo básico de entidad-vínculo con los conceptos de subclase, vínculo clase/subclase, especialización, generalización y categorización. Llamamos al modelo resultante modelo ER extendido o modelo EER. En esta sección resumiremos estos conceptos y los definiremos formalmente de la misma manera como definimos formalmente los conceptos del modelo ER en el capítulo 3.

Una *clase*[†] es un conjunto de entidades; esto incluye cualquiera de los elementos de los esquemas EER que agrupan entidades como tipos de entidades, subclases, superclases y categorías. Una *subclase* S es una clase cuyas entidades siempre deben ser un subconjunto de las entidades de otra clase, llamada la **superclase** C del vínculo **superclase/subclase** (o ES-UN). Denotamos un vínculo así con C/S . Para un vínculo superclase/subclase como éste siempre debe cumplirse

$$S \subseteq C$$

[†]El empleo de la palabra *clase* en el modelado conceptual difiere de su uso en los lenguajes de programación orientados a objetos, como C++; una clase es una definición de tipo estructurado junto con sus funciones (operaciones) aplicables.

Una **especialización** $Z = \{S_1, S_2, \dots, S_n\}$ es un conjunto de subclases que tienen la misma superclase G ; esto es, G/S_i es un vínculo superclase/subclase para $i = 1, 2, \dots, n$. Se dice que G es un **tipo de entidades generalizado** (o la **superclase** de la especialización, o una **generalización** de las subclases $\{S_1, S_2, \dots, S_n\}$). Se dice que Z es **total** si siempre se cumple

$$\bigcup_{i=1}^n S_i = G$$

de lo contrario, se dice que Z es **parcial**. Se dice que Z es **disjunta** si siempre se cumple

$$S_i \cap S_j = \emptyset \quad \text{para } i \neq j$$

En caso contrario, se dice que Z es **traslapada**.

Se dice que una subclase S de C está **definida por predicado** si se usa un predicado p sobre los atributos de C para especificar cuáles entidades de C son miembros de S ; esto es, $S = C[p]$, donde $C[p]$ es el conjunto de entidades de C que satisfacen p . Una subclase que no está definida por un predicado se denomina **definida por el usuario**.

Se dice que una especialización Z (o generalización G) está **definida por atributo** si se usa un predicado ($A = c_i$), donde A es un atributo de G y c_i es un valor constante del dominio de A , para especificar la pertenencia a cada subclase S_j de Z . Cabe señalar que, si $c_i \neq c_j$ para $i \neq j$, y A es un atributo monovaluado, la especialización es disjunta.

Una **categoría** T es una clase que es un subconjunto de la unión de n superclases de finidoras $D_1, D_2, \dots, D_n$, con $n > 1$, y se especifica formalmente como sigue:

$$T \subseteq (D_1 \cup D_2 \cup \dots \cup D_n)$$

Se puede usar un predicado p_i sobre los atributos de D_i para especificar los miembros de cada D_i que son miembros de T . Si se especifica un predicado sobre cada D_i , tenemos

$$T = (D_1[p_1] \cup D_2[p_2] \cup \dots \cup D_n[p_n])$$

Ahora debemos extender la definición de **tipo de vínculos** dada en el capítulo 3 para que cualquier clase —no sólo a cualquier tipo de entidades— pueda participar en un vínculo. Para ello, deberemos sustituir las palabras *tipo de entidades* por *clase* en esa definición. La notación gráfica de EER es consistente con ER porque todas las clases se representan con rectángulos.

21.1.6 Ejemplo de esquema de base de datos en el modelo EER

En esta sección daremos un ejemplo de esquema de base de datos en el modelo EER para ilustrar el empleo de los diversos conceptos estudiados aquí y en el capítulo 3. Consideremos una base de datos UNIVERSIDAD que mantiene información sobre los estudiantes, sus carreras, sus boletas de notas y su registro, así como de los cursos que ofrece la universidad. La base de datos también contiene información sobre los proyectos de investigación patrocinados del profesorado y de los estudiantes graduados. Este esquema se muestra en la figura 21.10. A continuación analizamos los requerimientos que dieron origen a este esquema.

Para cada persona en la base de datos, ésta mantiene información sobre su nombre [Nombre], número de seguro social [Nss], dirección [Dirección], sexo [Sexo] y fecha de nacimiento [FechaN]. Se identificaron dos subclases del tipo de entidades PERSONA: PROFESORADO y ESTUDIANTE. Los atributos específicos de PROFESORADO son el rango [Rango] (asistente, asociado, adjunto, investigador, visitante, etc.), oficina [OficinaP], teléfono de

comunes; hay otras posibles transformaciones. En seguida veremos las condiciones en las que deberá usarse cada opción. Usaremos $\text{Atrs}(R)$ para denotar los atributos de la relación R y $\text{CLP}(R)$ para denotar la clave primaria de R .

PASO 8: Convertir cada especialización con m subclases $\{S_1, S_2, \dots, S_m\}$ y superclase (generalizada) C , donde los atributos de C son $\{k, a_1, \dots, a_n\}$ y k es la clave (primaria), en esquemas de relación empleando una de estas cuatro opciones:

Opción 8A: Crear una relación L para C con atributos $\text{Atrs}(L) = \{k, a_1, \dots, a_n\}$ y $\text{CLP}(L) = k$. Crear una relación L_i por cada subclase S_i , $1 \leq i \leq m$, con los atributos $\text{Atrs}(L_i) = \{k\} \cup \{\text{atributos de } S_i\}$ y $\text{CLP}(L_i) = k$.

Opción 8B: Crear una relación L_i por cada subclase S_i , $1 \leq i \leq m$, con los atributos $\text{Atrs}(L_i) = \{\text{atributos de } S_i\} \cup \{k, a_1, \dots, a_n\}$ y $\text{CLP}(L_i) = k$.

Opción 8C: Crear una sola relación L con atributos $\text{Atrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{atributos de } S_1\} \cup \dots \cup \{\text{atributos de } S_m\} \cup \{t\}$ y $\text{CLP}(L) = k$. Esta opción es para una especialización cuyas subclases son *disjuntas*, y t es un atributo de **tipo** que indica la subclase a la que pertenece cada tupla, si la hay. Esta opción tiene el potencial de crear un gran número de valores nulos.

Opción 8D: Crear un solo esquema de relación L con atributos $\text{Atrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{atributos de } S_1\} \cup \dots \cup \{\text{atributos de } S_m\} \cup \{t_1, t_2, \dots, t_m\}$ y $\text{CLP}(L) = k$. Esta opción es para una especialización cuyas subclases se *traslapan* (no son disjuntas) y cada t_i , $1 \leq i \leq m$, es un atributo booleano que indica si una tupla pertenece o no a una subclase S_i . ■

La opción 8A crea una relación L para la superclase C y sus atributos, más una relación L_i para cada subclase S_i ; cada L_i incluye los atributos específicos de S_i , más la clave primaria de la superclase C , que se propaga a L_i y se convierte en su clave primaria. Una operación de EQUIRREUNIÓN sobre la clave primaria entre cualquier L_i y L produce todos los atributos específicos y heredados de las entidades de S_i . Esta opción se ilustra en la figura 21.11(a) para el esquema EER de la figura 21.4. La opción 8A funciona para cualesquier restricciones sobre la especialización: disjunta o traslapada, total o parcial. Observe que la

$$\pi_{\langle L_i \rangle}(L_i) \subseteq \pi_{\langle L_i \rangle}(L)$$

se debe cumplir para cada L_i . Esto especifica una *dependencia de inclusión* $L_i.k < L.k$ (véase la Sec. 13.4).

En la opción 8B, la operación de EQUIRREUNIÓN está *integrada* al esquema y no se necesita la relación L , como se ilustra en la figura 21.11(b) para la especialización EER de la figura 21.3(b). Esta opción sólo funciona bien con ambas restricciones, disjunta y total. Si la especialización no es total, una entidad que no pertenezca a ninguna de las subclases S_i se perderá. Si la especialización no es disjunta, una entidad que pertenezca a más de una subclase tendrá almacenados redundantemente en más de una L_i sus atributos heredados de la superclase C . Con la opción 8B, ninguna relación contiene todas las entidades de la superclase C ; en consecuencia, deberemos aplicar una operación de UNIÓN EXTERNA a las relaciones L_i para obtener todas las entidades de C . El resultado de la unión externa será similar a las relaciones producidas con las opciones 8C y 8D, excepto que faltarán los campos de

(a) EMPLEADO

NSS	NombreP	Ini	Apellido	FechaNac	Dirección	TipoTrabajo
-----	---------	-----	----------	----------	-----------	-------------

SECRETARIA

NSS	RapidezTeclado
-----	----------------

TÉCNICO

NSS	GradoT
-----	--------

INGENIERO

NSS	Tipolog
-----	---------

(b) COCHE

IdVehículo	NúmeroDeMatrícula	Precio	VelocidadMáx	NúmDePasajeros
------------	-------------------	--------	--------------	----------------

CAMIÓN

IdVehículo	NúmeroDeMatrícula	Precio	NúmDeEjes	Tonelaje
------------	-------------------	--------	-----------	----------

(c) EMPLEADO

NSS	NombreP	Ini	Apellido	FechaNac	Dirección	TipoTrabajo	RapidezTeclado	GradoT	Tipolog
-----	---------	-----	----------	----------	-----------	-------------	----------------	--------	---------

(d) COMPONENTE

NúmComp	Descripción	SeñalF	NúmDibujo	FechaFabricación	NúmLote	SeñalC	Proveedor	PrecioLista
---------	-------------	--------	-----------	------------------	---------	--------	-----------	-------------

Figura 21.11 Opciones para transformar especializaciones (o generalizaciones) a relaciones.

- (a) Transformación del esquema EER de la figura 21.4 a relaciones usando la opción 8A. (b) Transformación del esquema EER de la figura 21.3(b) a relaciones empleando la opción 8B. (c) Transformación del esquema EER de la figura 21.4 usando la opción 8C, donde TipoTrabajo desempeña el papel de atributo de tipo. (d) Transformación del esquema EER de la figura 21.5 empleando la opción 8D, con dos campos de tipo booleanos SeñalF y SeñalC.

tipo. Siempre que busquemos una entidad arbitraria en C , deberemos buscar en todas las m relaciones L_i .

Las opciones 8C y 8D crean una sola relación para representar la superclase C y todas sus subclases. Una entidad que no pertenezca a alguna de las subclases tendrá valores nulos para los atributos específicos de esas subclases. Por esto, las opciones mencionadas no se recomiendan si se han definido muchos atributos específicos para las subclases. Sin embargo, si hay pocos atributos específicos, estas transformaciones son preferibles a las opciones 8A y 8B porque hacen innecesario especificar operaciones de EQUIRREUNIÓN y de UNIÓN EXTERNA y por tanto pueden producir una implementación más eficiente. La opción 8C sirve para manejar subclases disjuntas mediante la inclusión de un solo **atributo de tipo** (o de **ingen**) t para indicar la subclase a la que cada tupla pertenece; así, el dominio de t podría ser $\{1, 2, \dots, m\}$. Si la especialización es parcial, t puede tener valores nulos en las tuplas que no pertenezcan a ninguna subclase. Si la especialización está definida por atributo, ese atributo hará las funciones de t y éste no será necesario; esta opción se ilustra en la figura 21.11(c) para la especialización EER de la figura 21.4. Con la opción 8D se manejan subclases traslapadas mediante la inclusión de m campos de tipo *booleanos*, uno para *cada* subclase.

Cada campo de tipo *t*_i puede tener un dominio {sí, no}, donde un valor de sí indica que la tupla es miembro de la subclase *S*_i. Esta opción se ilustra en la figura 21.11(d) para la especialización EER de la figura 21.5, donde SeñalF y SeñalC son los campos de tipo. Observe que también es posible crear un solo campo de tipo con *m* bits, en vez de los *m* campos de tipo.

Cuando tenemos una jerarquía o retícula de especialización (o generalización) de múltiples niveles, no tenemos que seguir la misma opción de transformación para todas las especializaciones; podemos usar una opción de transformación para una parte de la jerarquía o retícula y otras opciones para otras partes. La figura 21.12 muestra una posible transformación a relaciones de la retícula de la figura 21.7. Aquí usamos la opción 8A para PERSONA/EMPLEADO, EXALUMNO, ESTUDIANTE, la opción 8C para EMPLEADO/PERSONAL, PROFESORADO, ESTUDIANTE_ASISTENTE y para ESTUDIANTE/ESTUDIANTE_ASISTENTE, y la opción 8D para ESTUDIANTE_ASISTENTE/ASISTENTE_INVESTIGADOR, ASISTENTE_DOCENTE y para ESTUDIANTE/ESTUDIANTE_POSGRADO, ESTUDIANTE_LICENCIATURA}. En la figura 21.12, todos los atributos que comienzan con 'Tipo' o 'Señal' son campos de tipo.

21.2.2 Transformación de subclases compartidas

Una subclase compartida, como GERENTE_DE_INGENIERÍA en la figura 21.6, es una subclase de varias superclases. Todas estas clases deben tener el mismo atributo clave; de lo contrario, la subclase compartida se modelaría como categoría. Podemos aplicar cualquiera de las opciones analizadas en el paso 8 a una subclase compartida, aunque por lo regular se utiliza la opción 8A. En la figura 21.12, se usó la opción 8D para la subclase compartida ESTUDIANTE_ASISTENTE.

21.2.3 Transformación de categorías

Una categoría es una subclase de la unión de dos o más superclases que pueden tener diferentes claves porque pueden ser de diferentes tipos de entidades. Un ejemplo es la categoría DUEÑO que se muestra en la figura 21.8, que es un subconjunto de la unión de tres tipos de entidades, PERSONA, BANCO y COMPAÑÍA. La otra categoría de la figura 21.8, VEHÍCULO_REGISTERADO, tiene dos superclases con el mismo atributo clave.

PERSONA

NSS	Nombre	FechaNac	Sexo	Dirección
-----	--------	----------	------	-----------

EMPLEADO

NSS	Salario	TipoEmpleado	Puesto	Rango	FracciónTiempo	SeñalAI	SeñalAD	Proyecto	Curso
-----	---------	--------------	--------	-------	----------------	---------	---------	----------	-------

EXALUMNO GRADOS_EXALUMNO

NSS	Señal	Año	Grado	Carrera
-----	-------	-----	-------	---------

ESTUDIANTE

NSS	DeptoCarrera	SeñalPosg	SeñalLic	ProgramaPosgrado	Clase	SeñalEstudAsist
-----	--------------	-----------	----------	------------------	-------	-----------------

Figura 21.12 Transformación de la retícula de especialización EER de la figura 21.7, empleando múltiples opciones.

PERSONA

NSS	NúmLicConducir	Nombre	Dirección	IdDueño
-----	----------------	--------	-----------	---------

BANCO

NombreB	DireccB	IdDueño
---------	---------	---------

COMPAÑÍA

NombreC	DireccC	IdDueño
---------	---------	---------

DUEÑO

IdDueño

VEHÍCULO_REGISTERADO

IdVehículo	NúmeroDeMatrícula
------------	-------------------

COCHE

IdVehículo	EstiloA	MarcaA	ModeloA	AñoA
------------	---------	--------	---------	------

CAMIÓN

IdVehículo	MarcaC	ModeloC	Tonelaje	AñoT
------------	--------	---------	----------	------

POSEE

IdDueño	IdVehículo	FechaCompra	EmbargoONormal
---------	------------	-------------	----------------

Figura 21.13 Transformación a relaciones de las categorías de la figura 21.8.

Para establecer la transformación de una categoría cuyas superclases de definición tengan diferentes claves se acostumbra especificar un nuevo atributo clave, llamado **clave sustituta**, cuando se crea una relación que corresponda a la categoría. Esto se hace porque las claves de las clases de definición son diferentes, así que no podemos usar ninguna de ellas exclusivamente para identificar todas las entidades de la categoría. Ahora podemos crear un esquema de relación DUEÑO que corresponda a la categoría DUEÑO, tal como se ilustra en la figura 21.13, e incluir cualesquier atributos de la categoría en esta relación. La clave primaria de DUEÑO es la clave sustituta IdDueño. También agregamos el atributo de clave sustituta IdDueño como clave externa a cada relación que corresponda a una superclase de la categoría, a fin de especificar las correspondencias en valores entre la clave sustituta y la clave de cada superclase.

En una categoría cuyas superclases tengan la misma clave, como VEHÍCULO en la figura 21.8, no hay necesidad de una clave sustituta. La transformación de la categoría VEHÍCULO_REGISTERADO, que ilustra este caso, se muestra también en la figura 21.13.

21.3 Conceptos de abstracción de los datos y de representación de conocimientos*

En esta sección analizaremos en términos abstractos algunos de los conceptos de modelado que se describieron de manera muy específica en nuestra presentación de los modelos ER y EER en el capítulo 3 y en la sección 21.1. La terminología se usa tanto en el modelado de

datos conceptual como en la literatura de inteligencia artificial cuando se habla de la **representación de conocimientos** (abreviada RC), cuyo objetivo es desarrollar conceptos para modelar con exactitud algún "dominio de discurso" y así almacenar y manipular conocimientos para deducir inferencias, tomar decisiones o simplemente responder preguntas. Los objetivos de la RC son similares a los de los *modelos semánticos de los datos*, pero también hay algunas diferencias importantes entre las dos disciplinas. Aquí resumiremos las similitudes y las diferencias:

- Ambas disciplinas emplean un proceso de abstracción para identificar propiedades comunes y aspectos importantes de objetos del minimundo, al tiempo que suprimen diferencias insignificantes y detalles sin importancia.
- Ambas disciplinas cuentan con conceptos, restricciones, operaciones y lenguajes para definir datos y representar conocimientos.
- La RC tiene en general un alcance más amplio que los modelos semánticos de los datos. En los esquemas de RC se representan diferentes formas de conocimientos, como reglas (empleadas para inferir, deducir y buscar), conocimientos incompletos y por omisión, y conocimientos temporales y espaciales. Los modelos semánticos de los datos se están expandiendo para incluir algunos de estos conceptos.
- Los esquemas de RC cuentan con **mecanismos de razonamiento** que deducen hechos adicionales a partir de los hechos almacenados en una base de datos. Así pues, mientras que la mayoría de los sistemas de bases de datos actuales están limitados a contestar consultas directas, los sistemas basados en conocimientos que utilizan esquemas de RC pueden contestar consultas que impliquen **inferencias** sobre los datos almacenados. La tecnología de bases de datos se está extendiendo con mecanismos de inferencia (véase el Cap. 24).

- En tanto que la mayoría de los modelos de datos se concentran en la representación de esquemas de bases de datos, o metaconocimientos, los esquemas de RC a menudo combinan los esquemas con los ejemplares mismos a fin de lograr flexibilidad en la representación de excepciones. Con frecuencia, esto provoca ineficiencias cuando se implementan dichos esquemas de RC, en comparación con las bases de datos, sobre todo cuando es preciso almacenar una gran cantidad de datos (o hechos).

En esta sección examinaremos cinco **conceptos de abstracción** que se usan en los modelos semánticos de los datos, como el modelo EER, y también en los esquemas de RC. Se trata de los conceptos de clasificación/generación de ejemplares, identificación, generalización/especialización, agregación y asociación. Los conceptos apareados de clasificación y generación de ejemplares son inversos entre sí, como lo son la generalización y la especialización. Los conceptos de agregación y asociación también están relacionados. Trataremos estos conceptos abstractos y su relación con las representaciones concretas empleadas en el modelo EER para aclarar el proceso de abstracción de los datos y entender mejor el proceso relacionado de diseño de esquemas conceptuales.

21.3.1 Clasificación y generación de ejemplares

El proceso de **clasificación** implica asignar sistemáticamente objetos similares a clases de objetos. Ahora podemos describir (en BD) o razonar (en RC) sobre las clases más que sobre los

objetos individuales mismos. Los grupos de objetos comparten los mismos tipos de atributos y restricciones, y al clasificar los objetos simplificamos el proceso de descubrir sus propiedades. La **generación de ejemplares** es el inverso de la clasificación y se refiere a la producción y examen específico de los objetos distintos de una clase. Así pues, un ejemplar de objeto está relacionado con su clase de objetos mediante el vínculo *ES_UN_EJEMPLAR_DE*.

En general, los objetos de una clase deben tener una estructura de tipo similar. No obstante, algunos objetos pueden exhibir propiedades que difieran en algunos aspectos de los demás objetos de la clase; estos **objetos de excepción** también se tienen que modelar, y los esquemas de RC permiten excepciones más variadas que los modelos semánticos de los datos. Por añadidura, ciertas propiedades se aplican a la clase como un todo y no a los objetos individuales mismos; los esquemas de RC permiten semejantes **propiedades de clase**.

En el modelo EER, las entidades se clasifican en tipos de entidades según sus propiedades y estructura básicas. Las entidades se clasifican también en subclases y categorías dependiendo de las similitudes y diferencias (excepciones) adicionales entre ellas. Los ejemplares de vínculos se clasifican en tipos de vínculos. Así pues, tipos de entidades, subclases, categorías y tipos de vínculos son los diferentes tipos de clases del modelo EER. Este modelo no contempla explícitamente las propiedades de clase, pero puede ser extendido para que lo haga.

Los modelos de representación de conocimientos permiten múltiples esquemas de clasificación en los que una clase es un *ejemplar* de otra (llamada **metaclase**). Cabe señalar que esto *no se puede* representar directamente en el modelo EER, porque sólo tenemos dos niveles: clases y ejemplares. El único vínculo de clases en el modelo EER es un vínculo superclase/subclase, mientras que en algunos esquemas de RC se puede representar un vínculo adicional clase/ejemplar directamente en una jerarquía de clases. Un ejemplar puede ser en sí otra clase, lo que hace posible esquemas de clasificación de múltiples niveles.

21.3.2 Identificación

La **identificación** es el proceso de abstracción mediante el cual las clases y los objetos se hacen identificables de manera única por medio de algún **identificador**. Por ejemplo, un nombre de clase identifica de manera única toda una clase. Se requiere un mecanismo adicional para distinguir los diferentes ejemplares de objetos mediante identificadores de objetos. Además, es preciso identificar múltiples manifestaciones en la base de datos del mismo objeto del mundo real. Por ejemplo, podríamos tener una tupla *<Mateo Cadena, 610618, 3769821>* en una relación PERSONA y otra tupla *<301-54-0836, CS, 3,8>* en una relación ESTUDIANTE que casualmente representa la misma entidad del mundo real. No hay manera de identificar el hecho de que estos dos objetos (tuplas) de la base de datos representen la misma entidad del mundo real, si no tomamos medidas *durante el diseño* para efectuar referencias cruzadas apropiadas que provean dicha identificación. Así pues, la identificación es necesaria en dos niveles:

- Para distinguir entre los objetos y las clases de la base de datos.
- Para identificar los objetos de la base de datos y relacionarlos con sus contrapartes del mundo real.

En el modelo EER, la identificación de los elementos de un esquema se basa en un sistema de nombres únicos para los mismos. Por ejemplo, todas las clases de un esquema EER —sean tipos de entidades, subclases, categorías o tipos de vínculos— deben tener un nombre distinto. Los

nombres de los atributos de una clase dada también deben ser distintos. Además, se requieren reglas para identificar de manera no ambigua las referencias a los nombres de atributos en una retícula o jerarquía de especialización o generalización.

En el nivel de objetos, los valores de los atributos clave sirven para distinguir entre las entidades de un tipo dado. En el caso de los tipos de entidades débiles, las entidades se identifican con una combinación de los valores de su propia clave parcial y de las entidades con las que están relacionadas en el o los tipos de entidades propietarios. Los ejemplares de vínculos se identifican con la combinación de entidades que éstos relacionan; esto incluye una entidad de cada clase participante.

21.3.3 Especialización y generalización

La especialización es el proceso por el que se clasifica aún más una clase de objetos en subclases más especializadas. La generalización es el proceso inverso por el que se generalizan varias clases para obtener una clase abstracta de más alto nivel que incluya los objetos de todas estas clases. La especialización es refinación conceptual, y la generalización es síntesis conceptual. En el modelo EER la especialización y la generalización se representan con subclases. Al vínculo entre una subclase y su superclase lo llamamos aquí vínculo ES_UNA_SUBCLASE_DE.

21.3.4 Agregación y asociación

La **agregación** es un concepto de abstracción para construir objetos compuestos a partir de sus objetos componentes. Hay dos casos en los que es posible relacionar este concepto con el modelo EER. El primero es la situación en la que agregamos valores de atributos de un objeto para formar el objeto completo. El segundo caso, que no contempla explícitamente el modelo EER, implica la posibilidad de combinar objetos que se relacionen mediante un ejemplar de vínculo específico para formar un *objeto agregado de más alto nivel*. En ocasiones esto es útil cuando el propio objeto agregado de más alto nivel se debe relacionar con otro objeto. Al vínculo entre los objetos primitivos y su objeto agregado se le llama vínculo ES PARTE_DE, y al inverso se le denomina ES_UN_COMPONENTE_DE.

La abstracción de **asociación** sirve para asociar objetos de varias *clases independientes*; por tanto, es similar a la segunda aplicación de la agregación. Se representa en el modelo EER mediante tipos de vínculos. Este vínculo se denomina ESTÁ_ASOCIADO_A.

Para entender mejor la agregación, consideremos el esquema EER que se muestra en la figura 21.14(a), que almacena información sobre entrevistas de solicitantes de empleo en varias compañías. La clase COMPANÍA es una agregación de los atributos (u objetos componentes) NombreC (nombre de la compañía) y DireccC (dirección de la compañía), en tanto que SOLICITANTE_EMPLEO es un agregado de Nss, Nombre, Teléfono y Dirección. Los atributos de vínculo NombreContacto y TelContacto representan el nombre y el número telefónico de la persona de contacto en la compañía que se encarga de la entrevista. Supongamos que algunas entrevistas resultan en ofertas de empleo, pero otras no. Nos gustaría tratar a ENTREVISTA como una clase para asociarla a OFERTA_DE_EMPLEO. El esquema que se muestra en la figura 21.14(b) es *incorrecto* porque requiere que cada ejemplar de entrevista tenga una oferta de empleo. El esquema de la figura 21.14(c) está prohibido, porque el modelo EER no permite vínculos entre vínculos.

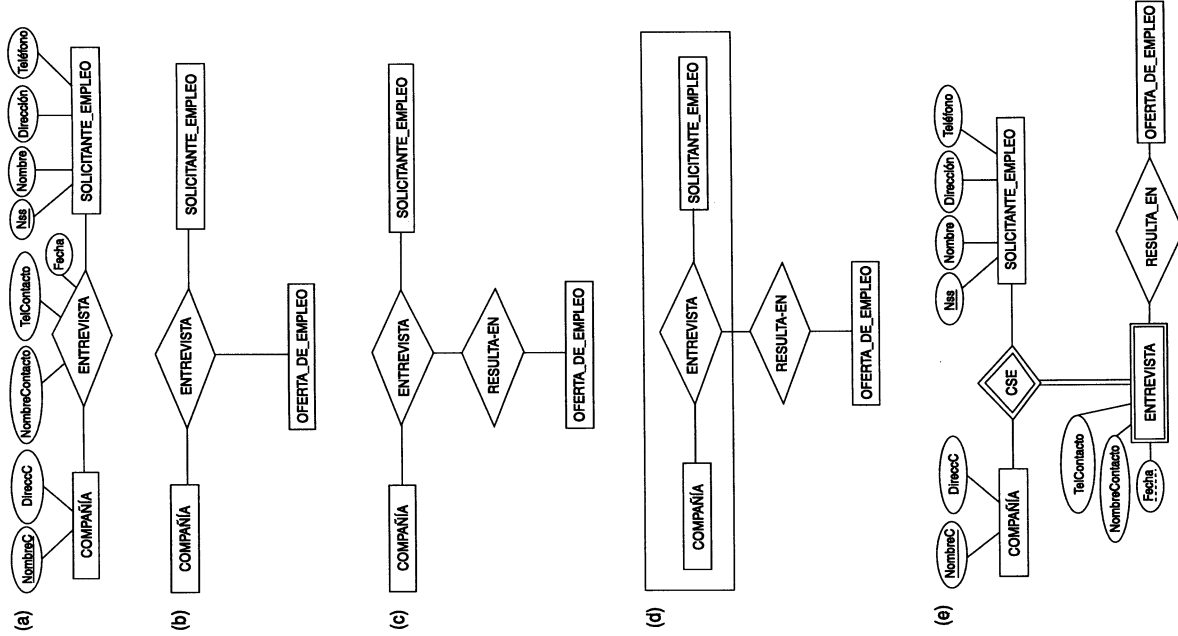


Figura 21.14 Agregación. (a) El tipo de vínculos ENTREVISTA. (b) Inclusión de OFERTA_DE_EMPLEO en un tipo de vínculos ternario (incorrecto). (c) Inclusión de OFERTA_DE_EMPLEO creando un vínculo en el que otro vínculo participa (no está permitido). (d) Empleo de agregación y de un objeto compuesto (molecular). (e) Representación correcta.

Una forma de representar esta situación consiste en crear una clase agregada de más alto nivel formada por `COMPANIA`, `SOLICITANTE_EMPLEO` y `ENTREVISTA` y en relacionar esta clase con `OFERTA_DE_EMPLEO` como se muestra en la figura 21.14(d). Aunque el modelo EER aquí descrito no cuenta con este recurso, algunos modelos semánticos de los datos sí lo permiten y llaman al objeto resultante un **objeto compuesto** o **molecular**. Otros modelos tratan a los tipos de entidades y a los tipos de vínculos de la misma manera y, por tanto, permiten vínculos entre vínculos.

Para representar correctamente esta situación en el modelo EER, necesitamos crear un nuevo tipo de entidades débil `ENTREVISTA`, como en la figura 21.14(e), y relacionarlo con `OFERTA_DE_EMPLEO`. Así, siempre podremos representar correctamente estas situaciones en el modelo EER si creamos tipos de entidades adicionales, aunque puede ser más deseable desde el punto de vista conceptual permitir una representación directa de la agregación, como en la figura 21.14(d), o permitir vínculos entre vínculos, como en la figura 21.14(c).

La distinción principal entre la agregación y la asociación es que, cuando se elimina un ejemplar de asociación, los objetos participantes siguen existiendo. Sin embargo, si manejamos la noción de objeto agregado —por ejemplo, un `COCHE` que se compone de los objetos `MOTOR`, `CHASIS` y `RUEDAS`—, entonces la eliminación del objeto agregado `COCHE` equivale a eliminar sus objetos componentes.

21.4 Restricciones de integridad en el modelado de datos*

Una base de datos almacena información sobre alguna *parte del mundo real*, a la que denominamos **minimundo** o **universo de discurso**. Ciertas reglas, las **restricciones de integridad**, gobiernan el minimundo, y suelen recibir el nombre de *reglas de negocios*. Cuando diseñamos un esquema para una aplicación de base de datos particular, una actividad importante consiste en identificar las restricciones de integridad que se deben cumplir en la base de datos. Queremos especificar el máximo posible de dichas restricciones al SGBD y hacer que éste se encargue de imponerlas. En esta sección resumiremos los principales tipos de restricciones de integridad que surgen con frecuencia en el modelado de bases de datos. Ya hablamos de las restricciones de integridad en cada uno de los modelos individuales que presentamos. En esta sección describiremos los tipos de restricciones de integridad, independientemente de un modelo de datos específico.

21.4.1 Restricciones de dominio

Las restricciones de dominio son restricciones sobre los tipos de valores que puede tener un atributo. Un **dominio** es un *conjunto de valores* que representan algún significado inherente. Se acostumbra asignar un **nombre de dominio** para indicar cómo pueden interpretarse los valores. Los valores que constituyen un dominio a menudo se especifican a través de un **tipo de datos**, por ejemplo enteros, números reales, cadenas, etc. En general, muchos dominios pueden tener el mismo tipo de datos, ya que en la mayor parte de los sistemas de bases de datos o lenguajes de programación hay un número limitado de tipos de datos básicos. El nombre del dominio se utiliza como concepto de alto nivel para distinguir entre dominios que pueden tener el mismo tipo de datos básico. Por ejemplo, dos dominios `Pesos_en_kilos` y `Alturas_en_metros` pueden tener, ambos, el tipo de datos de bajo nivel números reales positivos, pero los valores de los dominios representan diferentes conceptos de alto nivel, como

se refleja en los nombres de los dominios. En general, varios atributos pueden tener el mismo dominio.

Las definiciones de los tipos de datos en los sistemas de bases de datos son objeto de una constante expansión de modo que reflejen algo más del significado asociado a los valores de los datos. Ya es común en los sistemas orientados a objetos definir el conjunto apropiado de **operaciones** asociadas a cada tipo de datos. Los siguientes son algunos de los tipos de datos que suelen incluirse en los sistemas de bases de datos:

- **Tipos de datos numéricos:** Éstos incluyen los números enteros y reales en diversos formatos. A fin de reflejar los tipos de datos de bajo nivel disponibles en la mayor parte de los computadores, los enteros pueden diferenciarse en tipos enteros cortos (2 bytes), enteros (4 bytes) y enteros largos; de manera similar, los reales pueden diferenciarse en tipos de punto flotante y de doble precisión. Si se desea contemplar restricciones de dominio de más alto nivel, es posible introducir restricciones adicionales, como limitar un dominio a valores numéricos no negativos o positivos, o especificar explícitamente un subintervalo de valores numéricos que puede adoptar el atributo. Por ejemplo, se puede especificar que el dominio `Pesos_de_personas_en_kilos` tiene el tipo de datos: números positivos de punto flotante menores que 300. Las operaciones para los tipos de datos numéricos suelen incluir los operadores aritméticos (+, -, *, /, ...) y de comparación (=, <, >, ...) estándar.

- **Tipos de datos de caracteres y de cadenas:** Los tipos de datos de un solo carácter pueden incluir cualquier carácter del conjunto de caracteres ASCII estándar. Los tipos de datos para cadenas de caracteres a menudo se caracterizan por la longitud de las cadenas permitidas. Los tipos de datos de cadenas de longitud fija incluyen valores que tienen todos la misma longitud, en tanto que las cadenas de longitud variable no tienen esta restricción. Es común especificar una longitud máxima de las cadenas de longitud variable para fines de implementación. También es posible restringir un tipo de datos de cadena de modo que incluya sólo ciertos caracteres; la restricción más común consiste en limitar los caracteres a las letras del alfabeto, los dígitos decimales o los bits binarios. Los operadores para los tipos de datos de caracteres y cadenas por lo regular incluyen la comparación de igualdad y desigualdad y la concatenación de cadenas.

- **Tipos de datos booleanos:** Los valores de los tipos de datos booleanos están limitados a True y False (o 0 y 1), con las operaciones lógicas asociadas AND, OR y NOT.

- **Tipos de datos enumerados:** Éstos se identifican mediante un conjunto de valores explícito. Por ejemplo, se puede especificar el dominio *Calificaciones_de_letra_de_estudiantes* como el conjunto de caracteres {A, B, C, D, F, I, P}. También es posible, aunque no muy común, definir un tipo de datos enumerado *dinámico*; por ejemplo, el dominio de *Nombres_de_departamentos* de una universidad puede incluir el conjunto de nombres de todos los departamentos actuales. Si se crea un nuevo departamento, se deberá añadir su nombre al conjunto enumerado.

- **Tipos de datos de fechas y horas:** En vista de que las fechas de calendario y las horas desempeñan un papel cada vez más importante en los sistemas de bases de datos, ha aumentado también la importancia de definir tipos de datos básicos para manejar

estos conceptos.¹ Existen tipos de datos compuestos que por lo regular incluyen varios campos de **fecha** (año, mes, día) y **hora** (hora, minuto, segundo). Una **marca de tiempo** incluye campos tanto de fecha como de hora. Se debe contar con valores tanto absolutos (una hora específica en una fecha específica) como relativos (llamados **intervalos**), así como operaciones para comparar y manipular los valores de estos tipos de datos. Una operación de diferencia entre dos marcas de tiempo absolutas produce un intervalo, en tanto que una operación que incrementa o decrementa una marca de tiempo absoluta en un intervalo fijo produce otra marca de tiempo absoluta.

- **Tipos de datos definidos por el usuario:** La capacidad de permitir que un usuario defina dominios con un tipo de datos compuesto nuevo, así como sus operaciones, otorga más flexibilidad a un sistema de bases de datos; algunos sistemas orientados a objetos ya cuentan con esta capacidad.

21.4.2 Restricciones de clave y de vínculo

La **restricción de clave** es una de las restricciones estándar que con frecuencia aparecen en las aplicaciones de bases de datos. Estas restricciones se manejan de formas ligeramente distintas en los diversos modelos de datos. En el modelo ER, una clave es un atributo de un tipo de entidades que debe tener un valor único para cada entidad que pertenezca a dicho tipo en *cualquier momento específico*. Así, el valor del atributo clave puede servir para identificar de manera única cada entidad. Los atributos clave deben ser monovaluados, pero pueden ser simples o compuestos. Un tipo de entidades **normal** puede tener una o más claves; un tipo de entidades **débil** no tiene clave, pero casi siempre tiene una *clave parcial* cuyos valores identifican de manera única las entidades débiles que *están relacionadas a la misma entidad propietario* a través de un vínculo identificador (véase la Sec. 3.3.4). Las subclases heredan la clave de su superclase.

El concepto de clave en el modelo de red es similar, excepto que se utiliza la palabra **reservada UNIQUE** para identificar las claves. Además, algunos tipos de registros pueden no tener claves. En el modelo relacional, cada relación base tiene una **clave primaria** y puede tener claves secundarias adicionales (identificadas con la palabra reservada **UNIQUE** en SQL). SQL también permite relaciones sin claves. Una clase de entidades sin clave suele recibir el nombre **de bolsa**; una clase así permite múltiples entidades idénticas y por tanto no es un conjunto.

Las restricciones estructurales de los vínculos sirven para especificar la forma en que las entidades de un tipo pueden participar en ejemplares de un tipo de vínculos particular. Hay dos tipos principales de restricciones de vínculo en el modelo ER. La **razón de cardinalidad** especifica si una entidad puede participar sólo en un ejemplar de vínculo o en varios. Las razones de cardinalidad comunes para los vínculos binarios son 1:1, 1:N, N:1 y M:N. La **restricción de participación** especifica si una entidad *debe* participar en un ejemplar de vínculo o si puede existir de manera independiente ya sea que esté relacionada o no con otra entidad. Lo primero se denomina *participación total* o *dependencia de existencia*, y lo segundo se denomina *participación parcial*. Una técnica alternativa, más general, para especificar restricciones estructurales sobre vínculos especifica un par de enteros (*mín*, *máx*) ligados a cada participación de un tipo de entidades E en un tipo de vínculos V; esto restringe a cada entidad e de E a participar en *cuando menos mín* y *cuando más máx* ejemplares de V en todo

¹Por ejemplo, estos tipos de datos están incluidos en la norma SQL2 (véase la Sec. 7.1).

momento. Esta técnica cubre tanto la razón de cardinalidad como la restricción de participación, y es aplicable a vínculos de cualquier grado.

En el modelo relacional, los vínculos se representan mediante claves externas. En SQL podemos especificar el comportamiento de cada clave externa, cuando la clave primaria referida se actualiza o elimina, con una de las opciones **REJECT** (rechazar), **PROPAGATE** (propagar), **SET TO NULL** (poner nulo) o **SET TO DEFAULT** (poner valor por omisión). En el modelo de red, los vínculos se especifican a través de tipos de conjuntos, y su comportamiento se define con las opciones de inserción y de retención.

21.4.3 Restricciones generales de integridad semántica

Muchas restricciones de un minimundo para una aplicación de bases de datos se pueden especificar como restricciones de clave y de vínculo. Sin embargo, es frecuente que haya otras restricciones, más complejas, que no puedan especificarse a partir de estos conceptos. Estas se conocen como *restricciones de integridad semántica* generales. Los mecanismos para especificar tales **restricciones explícitas** todavía se están refinando en los sistemas de bases de datos comerciales. En general, se han propuesto dos estrategias para especificar estas restricciones: la estrategia *por procedimientos* y la *declarativa*.

Especificación de restricciones por procedimientos en las transacciones. Un método para especificar restricciones explícitas consiste en escribir instrucciones que verifiquen las restricciones en los programas de actualización (o transacciones) que se aplicarán a la base de datos. Esto se denomina **especificación de restricciones por procedimientos** (o *técnica de restricciones codificadas*). El programador codifica las restricciones en las transacciones apropiadas. Por ejemplo, consideremos la restricción general de que “el salario de un empleado no debe ser mayor que el del gerente del departamento al que pertenece el empleado”, especificada sobre el esquema ER de la figura 3.14. Para asegurar una imposición correcta, esta restricción debe verificarse en *cada transacción de actualización* que pudiera violar la restricción. Esto incluye cualquier transacción que modifique el salario de un empleado, que inserte un nuevo empleado o que relacione un empleado con un departamento, que asigne un nuevo gerente a un departamento, y así sucesivamente. En todas estas transacciones, parte del código de la transacción debe verificar explícitamente que la restricción no se esté violando. Si así fuera, la transacción deberá abortarse sin introducir cambios en la base de datos. Esto garantiza que la transacción se comportará como unidad atómica para el control de la integridad semántica.

La técnica precedente para manejar las restricciones explícitas la usan muchos SCDB existentes. También es común en las bases de datos orientadas a objetos, donde se pueden codificar restricciones explícitas como parte de los métodos u operaciones que se **encapsulan** con los objetos. Esta estrategia es completamente general, ya que las verificaciones se programan en un lenguaje de programación de propósito general, y el programador puede codificar las verificaciones de manera eficiente; sin embargo, no es muy flexible, y representa una carga adicional para el programador, quien debe conocer todas las restricciones que puede violar una transacción, y debe incluir verificaciones para asegurar que ninguna de ellas se viole. Una omisión, malentendido o error por parte del programador puede dejar la base de datos en un estado inconsistente.

Otra desventaja de especificar las restricciones por procedimientos es que las restricciones pueden modificarse con el tiempo, al cambiar las reglas de negocios en una situación

del mínimo. Si una restricción cambia, el DBA debe ordenar a los programadores apropiados que recodifiquen todas las transacciones y operaciones afectadas por la alteración. Esto también deja abierta la posibilidad de pasar por alto algunas transacciones y así producir errores en la representación e imposición de las restricciones.

Especificación declarativa de restricciones como aserciones. Una técnica más formal para representar restricciones explícitas es con un **lenguaje de especificación de restricciones**, que suele basarse en alguna variación del cálculo relacional. Este **enfoque declarativo** establece una separación clara entre la **base de restricciones** (en la que las restricciones se almacenan en una forma codificada apropiada) y el **subsistema de control de integridad** del SGBD (que tiene acceso a la base de restricciones para aplicar estas últimas correctamente a las transacciones afectadas).

Cuando se usa esta técnica, las restricciones suelen llamarse **aserciones**. Se ha sugerido el uso de esta estrategia con SGBD relacionales.[†] El subsistema de control de integridad compila las aserciones, que entonces se almacenan en el catálogo del SGBD, donde el subsistema de control de integridad puede consultarlas e imponerlas automáticamente. En consecuencia, las transacciones de actualización se pueden escribir *sin instrucciones de verificación de restricciones*. Esta estrategia es muy atractiva desde el punto de vista de los usuarios y programadores por su flexibilidad. Las transacciones de actualización se escriben sin ninguna verificación concomitante de violaciones de las restricciones. Una restricción se puede modificar independientemente de las transacciones sin que el diseñador tenga que recodificar alguna de ellas; todo lo que tiene que hacer el diseñador es cancelar la aserción anterior y especificar su reemplazo en el lenguaje de especificación de aserciones. La nueva aserción se compila y a partir de ese momento sustituye a la aserción cancelada en el catálogo del SGBD. Desafortunadamente, se ha demostrado que esta técnica es muy difícil de implementar con eficiencia, porque los subsistemas de control de integridad han resultado ser muy complejos. Las investigaciones para hacer más eficiente este enfoque continúan; incluyen el empleo de sistemas de reglas, bases de datos deductivas (véase el Cap. 24) y bases de datos activas (véase el Cap. 25).

21.4.4 Restricciones inherentes, implícitas y explícitas

En esta sección resumiremos las tres principales técnicas para especificar restricciones en un sistema de bases de datos. Cada modelo de datos tiene un conjunto de restricciones integradas asociadas a los elementos del modelo de datos. Llamamos a éstas las **restricciones inherentes** del modelo de datos, y *no es preciso especificarlas* en el DDL cuando se está definiendo un esquema de base de datos; son propiedades inherentes de los elementos mismos del modelo de datos. Un ejemplo de restricción inherente en el modelo ER es que todo ejemplo de vínculo de un tipo de vínculos *n-ario V* relaciona *exactamente una entidad de cada tipo de entidades que participa en V en un papel específico*, y que cada ejemplar de vínculo relaciona una asociación única de entidades en cualquier momento dado. Así pues, las restricciones inherentes son las **reglas** que definen a los elementos del modelo de datos.

Las **restricciones implícitas**, en cambio, se especifican al SGBD por medio del lenguaje de definición de datos durante el proceso de creación de esquemas de bases de datos. Las

[†]El lenguaje SQL tiene el enunciado ASSERT para especificar aserciones (véase el Cap. 7), aunque este enunciado no se ha implementado en muchos SGBD relacionales actuales.

diversas especificaciones que describen cada tipo de entidades, atributos y vínculos en un esquema particular *implican* dichas restricciones. Por ejemplo, especificar que un cierto atributo es una clave de un tipo de entidades implica la restricción de que el atributo debe tener un valor único para cada entidad. De manera similar, especificar una restricción estructural sobre un tipo de vínculos *V* define implícitamente las restricciones correspondientes sobre *V*. El compilador de DDL interpreta y almacena estas restricciones en el catálogo del SGBD para que el software del SGBD pueda imponer automáticamente las restricciones cada vez que se efectúe una actualización. Las restricciones implícitas contempladas por un SGBD se *especifican en el DDL* durante la definición de un esquema.

Cada modelo de datos incluye un conjunto distinto de restricciones inherentes e implícitas. Una implementación dada de un modelo de datos en un SGBD *particular* casi siempre provee apoyo automático sólo para algunas de las restricciones inherentes e implícitas del modelo de datos. En general, los modelos de datos de alto nivel representan más restricciones inherentes e implícitas que los de bajo nivel. Sin embargo, ningún modelo de datos es capaz de representar todos los tipos de restricciones que pueden ocurrir en un minimundo. Por tanto, es necesario especificar **restricciones explícitas** adicionales sobre cada esquema de base de datos en particular. Ejemplos de restricciones especificadas explícitamente son las restricciones generales de integridad semántica que vimos en la sección 21.4.3. Los programadores especifican estas restricciones ya sea por procedimientos cuando implementan las transacciones de la base de datos o declarativamente por medio de un lenguaje de especificación de aserciones.

21.4.5 Restricciones de estado y de transición

Por último, hay otra clasificación posible de las restricciones, que depende de si hacen referencia a un solo estado de la base de datos o a múltiples estados. Una **restricción de estado** es una restricción que vale para todos los **estados no transitorios** de la base de datos; esto es, todos los estados en los que la base de datos no está en proceso de actualizarse. Recuerde que un *estado de base de datos* se refiere a todos los datos que están en ella en un momento determinado. Un estado de base de datos no transitorio es **consistente** si satisface todas las restricciones de estado. Ejemplos de restricciones de estado son las de dominio, las de clave y las estructurales sobre vínculos, así como la mayor parte de las otras restricciones que hemos analizado hasta ahora.

Las restricciones de estado deben verificarse siempre que una transacción de actualización modifique el estado de la base de datos. Una transacción debe incluir verificaciones suficientes para garantizar que, si el estado de la base de datos es consistente antes de ejecutarse la transacción, su estado también será consistente después de la ejecución. Se considera que las transacciones son unidades de actualización atómicas, no sólo para las restricciones de integridad, sino también para el control de concurrencia y la recuperación, como vimos en los capítulos 17, 18 y 19.

Otro tipo de restricciones se aplica a múltiples estados de la base de datos y se denomina **restricción de transición**. Ésta es una restricción sobre la transición de un estado a otro, no sobre un estado individual. Un ejemplo es que el atributo Salario de un empleado sólo se puede aumentar; esto significa que cualquier actualización del atributo Salario se aceptará sólo si el nuevo valor de Salario es mayor que el anterior. Cabe señalar que la restricción no se aplica al estado de la base de datos antes de la actualización ni a su estado

después de la actualización; se especifica sobre la transición de estados y abarca los valores de los atributos tanto *antes* como *después* de la transacción. En general, las restricciones de transición son mucho menos frecuentes que las de estado, y se especifican principalmente como *restricciones explícitas*.

21.5 Operaciones de actualización en EER y especificación de transacciones*

Hay dos partes complementarias en el proceso de diseño de bases de datos, como se ilustra en la figura 3.1. Hasta aquí, nos hemos concentrado en el diseño conceptual de los requerimientos de información de una aplicación de base de datos. La especificación de los requerimientos funcionales y de procesamiento debe efectuarse en paralelo, lo que, por lo regular, se logra con una técnica de especificación de procesos de alto nivel, como los diagramas de flujo de datos, seguida de un diseño más detallado. En general, es preciso usar los elementos de los lenguajes de programación, como las instrucciones *if*, las construcciones cíclicas y la entrada-salida para especificar transacciones complejas de base de datos. Además hay que incluir un conjunto de operaciones de base de datos para especificar los accesos a esta última. En esta sección presentaremos las operaciones de actualización básicas del modelo EER, y después mostraremos cómo se pueden especificar transacciones sencillas durante el diseño de una base de datos combinando estas operaciones.

Podemos usar ocho operaciones básicas para especificar actualizaciones en un esquema EER. Las operaciones **insertar entidad** y **eliminar entidad** especifican la creación y eliminación de una entidad en un tipo de entidades. Las operaciones **añadir vínculo** y **quitar vínculo** cumplen funciones similares con los tipos de vínculos. Abreviaremos estas cuatro operaciones como **insertar E**, **eliminar E**, **añadir V** y **quitar V**, respectivamente. Dos operaciones más, **modificar E** y **modificar V**, son necesarias para especificar la modificación de los valores de los atributos de entidades y vínculos, respectivamente. Por último, las dos operaciones **añadir a clase** y **quitar de clase**, abreviadas **añadir C** y **quitar C** respectivamente, se usan para añadir una entidad a una subclase o categoría y para quitarla de ella. Ahora especificaremos informalmente la forma y argumentos de cada una de estas operaciones:

- **insertar E** nombre_de_tipo_de_entidades atributos nombre_de_atributo ← valor, ...
- **eliminar E** nombre_de_tipo_de_entidades donde condición
- **añadir V** nombre_de_tipo_de_vínculos donde nombre_de_tipo_de_entidades : condición, ...
atributos nombre_de_atributo ← valor, ...
- **quitar V** nombre_de_tipo_de_vínculos donde nombre_de_tipo_de_entidades : condición, ...
- **modificar E** nombre_de_tipo_de_entidades donde condición atributos nombre_de_atributo ← valor, ...
- **modificar V** nombre_de_tipo_de_vínculos donde nombre_de_tipo_de_entidades : condición, ...
atributos nombre_de_atributo ← valor, ...
- **añadir C** nombre_de_subclase_o_categoria de nombre_de_superclase donde condición atributos nombre_de_atributo ← valor, ...
- **quitar C** nombre_de_subclase_o_categoria donde condición

En la lista anterior, *nombre_de_tipo_de_entidades* es el nombre de uno de los tipos de entidades del esquema de base de datos; de manera similar, *nombre_de_tipo_de_vínculos*, *nombre_de_subclase_o_categoria* y *nombre_de_atributo* son los nombres de los tipos de vínculos, subclases o categorías y atributos apropiados del mismo esquema. Un *valor* puede ser un valor constante del dominio de los atributos o un parámetro (variable) que contiene un valor así. Por último, *condición* especifica un predicado que selecciona una o más entidades de un tipo de entidades o clase. He aquí seis ejemplos que ilustran algunas de estas operaciones sobre el esquema ER COMPANÍA de la figura 3.14:

```
insertar E PROYECTO
atributos Nombre ← "ProductoA", Número ← 9, Lugar ← "Minneapolis";
eliminar E PROYECTO donde (Nombre = "ProductoA");
añadir V TRABAJA_EN
donde PROYECTO:(Nombre = "ProductoA"), EMPLEADO:(NSS = "123456789")
atributos Horas ← 20;
quitar V TRABAJA_EN
donde PROYECTO:(Nombre = "ProductoA"), EMPLEADO:(NSS = "123456789");
modificar E PROYECTO donde (Nombre = "ProductoA")
atributos Lugar ← "Ancona";
modificar V TRABAJA_EN
donde PROYECTO:(Nombre = "ProductoA"), EMPLEADO:(NSS = "123456789")
atributos Horas ← 30;
```

Las dos primeras órdenes especifican la adición y la eliminación de una entidad PROYECTO; las siguientes dos especifican la adición y la eliminación de un ejemplar de vínculo en TRABAJA_EN, y las últimas dos especifican modificaciones de atributos. Observe que los ejemplares de vínculos se identifican especificando una condición sobre cada una de las entidades que participan en el vínculo. Si cualquiera de estas condiciones especifica más de una entidad, más de un ejemplar de vínculo resultará afectado por la orden. Por ejemplo, la operación

```
quitar V TRABAJA_EN
donde PROYECTO:(Lugar = "Higueras"), EMPLEADO:(Salario > 50000);
```

especificaría la eliminación de todos los ejemplares de vínculos que relacionen *cualquier* PROYECTO ubicado en "Higueras" con *cualquier* EMPLEADO cuyo Salario sea mayor que \$50 000. Si no se especifica uno de los tipos de entidades participantes, se seleccionarán todas sus entidades. Por ejemplo, la operación

```
quitar V TRABAJA_EN donde PROYECTO:(Nombre = "ProductoA");
```

especificaría la eliminación de todos los ejemplares de vínculos que relacionen *cualquier* EMPLEADO con la entidad PROYECTO especificada. También podemos anexar nombres de papeles a los tipos de entidades participantes, si es necesario, como se ilustra en los ejemplos que siguen.

Estas operaciones básicas de actualización se pueden combinar para especificar transacciones lógicas de actualización sobre un esquema determinado durante el diseño de la base de datos. En este caso, la transacción recibe un nombre, varios parámetros y un cuerpo de código de especificación que incluye las operaciones de base de datos apropiadas. Por ejemplo, consideremos la transacción CONTRATAR_EMP_EN_DEPTO, por la cual se contrata un empleado nuevo que trabajará en un cierto departamento. Los parámetros incluyen todos los atributos pertinentes de la nueva entidad EMPLEADO, junto con una forma de

identificar el departamento al que él se incorporará, y posiblemente el nombre de su supervisor directo. Esto puede especificarse como sigue:

```

DEFINIR TRANSACCIÓN CONTRATAR_EMP_EN_DEPTO
PARÁMETROS NP, IN, AP, NSS, FN, SEXO, SAL, DIR, NOMD, NSSSUPER
INICIAR_TRANS
  Insertar E EMPLEADO atributos
  Nombre.NombreP ← NP, Nombre.Inic ← IN, Nombre.Apellido ← AP, Nss
  ← NSS;
  FechaN ← FN, Sexo ← SEXO, Salario ← SAL, Dirección ← DIR;
  añadir V PERTENECE_A donde
    EMPLEADO : (Nss = NSS), DEPARTAMENTO : (Nombre = NOMD);
  añadir V SUPERVISIÓN donde
    EMPLEADO.supervisor : (Nss = NSSSUPER),
    EMPLEADO.supervisado : (Nss = NSS);
FIN_TRANS;

```

Aquí, los parámetros de la transacción se especifican como variables, cuyos ejemplares deben crearse proporcionando valores apropiados siempre que un empleado nuevo se inserte realmente en la base de datos. Usamos la notación de *punto* para especificar los atributos componentes de un atributo compuesto (Nombre.NombreP), o para especificar pares (EMPLEADO.supervisor); con una notación similar se pueden especificar entidades relacionadas mediante un vínculo. Esta especificación de transacciones de alto nivel es lo bastante precisa como para especificar las acciones de la transacción con toda claridad, y para establecer una base sobre la cual implementar la transacción posteriormente. Observa que no especificamos ninguna verificación de restricciones de integridad en esta transacción, porque el objetivo del proceso de especificación aquí es definir sin ambigüedad el significado de la transacción. Todas las restricciones de integridad especificadas en el esquema COMPANÍA, así como cualesquier otras restricciones explícitas adicionales, se deberán imponer cuando se implemente el sistema de base de datos.

Ahora especificaremos otras cuatro transacciones: CAMBIAR_SUPERVISOR_EMP, TERMINAR_PROYECTO, ASIGNAR_EMP Y DESASIGNAR_EMP, sobre el esquema ER COMPANÍA:

```

DEFINIR TRANSACCIÓN CAMBIAR_SUPERVISOR_EMP
PARÁMETROS NSSEMP, NUEVONSSSUPER
INICIAR_TRANS
  quitar V SUPERVISIÓN donde
    EMPLEADO.supervisado : (Nss = NSSEMP);
  añadir V SUPERVISIÓN donde
    EMPLEADO.supervisor : (Nss = NUEVONSSSUPER),
    EMPLEADO.supervisado : (Nss = NSSEMP);
FIN_TRANS;

```

```

DEFINIR TRANSACCIÓN TERMINAR_PROYECTO
PARÁMETROS NOMBREPR
INICIAR_TRANS
  eliminar E PROYECTO donde Nombre = NOMBREPR;
  quitar V TRABAJA_EN donde
    PROYECTO : (Nombre = NOMBREPR);
  quitar V CONTROLA donde
    PROYECTO : (Nombre = NOMBREPR);
FIN_TRANS;

```

```

DEFINIR TRANSACCIÓN ASIGNAR_EMP
PARÁMETROS NSSEMP, NOMBREPR, HORASPORSEMANA
INICIAR_TRANS
  añadir V TRABAJA_EN donde
    EMPLEADO : (Nss = NSSEMP), PROYECTO : (Nombre = NOMBREPR)
    atributos Horas ← HORASPORSEMANA;
FIN_TRANS;

DEFINIR TRANSACCIÓN DESASIGNAR_EMP
PARÁMETROS NSSEMP, NOMBREPR
INICIAR_TRANS
  quitar V TRABAJA_EN donde
    EMPLEADO : (Nss = NSSEMP),
    PROYECTO : (Nombre = NOMBREPR);
FIN_TRANS;

```

Por último, ilustramos las operaciones de subclases con una transacción sobre el esquema EER UNIVERSIDAD de la figura 21.10. La siguiente transacción especifica la contratación de una nueva entidad PROFESORADO; para ello insertamos la entrada nueva en el tipo de entidades PERSONA, añadiéndolo después a la subclase PROFESORADO y a la categoría PROFESOR_INVESTIGADOR, y finalmente relacionando el PROFESORADO a un DEPARTAMENTO a través del vínculo PERTENECE.

```

DEFINIR TRANSACCIÓN CONTRATAR_NUEVO_PROFESOR
PARÁMETROS NP, IN, AP, NSS, FN, SEXO, DIR, SAL, RANGO, OFIC, TEL,
NOMBRED
INICIAR_TRANS
  Insertar E PERSONA atributos
  Nombre.NombreP ← NP, Nombre.Inic ← IN, Nombre.Apellido ← AP,
  Nss ← NSS;
  FechaN ← FN, Sexo ← SEXO, Dirección ← DIR;
  añadir C PROFESORADO de PERSONA donde Nss = NSS atributos
  Rango ← RANGO, Salario ← SAL, OficinaP ← OFIC, TeléfonoP ← TEL;
  añadir C PROFESOR_INVESTIGADOR de PROFESORADO donde Nss
  = NSS;
  añadir V PERTENECE donde
    PROFESORADO : (Nss = NSS), DEPARTAMENTO : (NombreD = NOMBRED);
FIN_TRANS;

```

Observe que los atributos heredados, como Nss de PROFESORADO, pueden servir para especificar condiciones sobre una subclase. Es evidente que el proceso de especificar transacciones simples es bastante directo. Si son más complejas, es necesario usar elementos notacionales de lenguaje de programación, como instrucciones *if* y *case*, ciclos, entrada y salida, y (en muchos casos) presentación de los resultados de consultas de la base de datos. Por ejemplo, una transacción compleja para especificar el proceso de reservar asientos en una línea aérea puede requerir que primero se obtengan los vuelos apropiados de la base de datos y se presenten en la pantalla para que los vea el agente de viajes; en seguida, se actualizará la base de datos para especificar la reserva de asientos en un cierto vuelo. Como no existe una notación estándar para especificar esto, dejaremos que nuestros lectores utilicen las construcciones de su lenguaje de programación favorito o de pseudocódigo para especificar transacciones complejas durante el diseño de bases de datos.

21.6 Panorama de otros modelos de datos*

Como mencionamos al iniciarse este capítulo, se han propuesto muchos modelos de datos conceptuales. En la sección 21.1 presentamos los conceptos de modelado semántico más importantes como parte del modelo EER. En esta sección ofreceremos un bosquejo de otros cuatro modelos de datos: el funcional, el relacional anidado, el estructural y el semántico.

21.6.1 Modelos de datos funcionales

Los modelos de datos funcionales (MDF) emplean el concepto de función matemática como elemento de modelado fundamental. Cualquier solicitud de información se puede visualizar como una llamada a una función con ciertos argumentos, y la función devuelve la información requerida. Existen varias propuestas de modelos de datos y lenguajes de consulta funcionales (véase la bibliografía al final de este capítulo). El modelo DAPLEX y su lenguaje son tal vez el ejemplo mejor conocido.

Las principales primitivas de modelado de un MDF son **entidades** y **vínculos funcionales**. Hay varios tipos de entidades estándar en el nivel más básico, como STRING, INTEGER, CHARACTER y REAL (entre otras), y se denominan **tipos de entidades imprimibles**. Los tipos de entidades abstractos que corresponden a objetos del mundo real tienen el tipo ENTITY. Por ejemplo, consideremos el diagrama EER de la figura 21.10; mostraremos cómo se pueden especificar algunos de los tipos de entidades y de vínculos de ese esquema en una notación funcional similar a la de DAPLEX. Los tipos de entidades PERSONA, ESTUDIANTE, CURSO, SECCIÓN y DEPARTAMENTO se declaran como sigue:

```
PERSONA() → ENTITY
ESTUDIANTE() → ENTITY
CURSO() → ENTITY
SECCIÓN() → ENTITY
DEPARTAMENTO() → ENTITY
```

Estos enunciados especifican que las funciones PERSONA, ESTUDIANTE, CURSO, SECCIÓN y DEPARTAMENTO devuelven entidades abstractas; por tanto, estos enunciados sirven para definir los tipos de entidades correspondientes. Un atributo de un tipo de entidades también se especifica como una función cuyo argumento (dominio) es el tipo de entidades y cuyo resultado (intervalo) es una entidad imprimible. Las siguientes declaraciones de funciones especifican los atributos de PERSONA:

```
NSS(PERSONA) → STRING
FECHAN(PERSONA) → STRING
SEXO(PERSONA) → CHAR
```

Al aplicar la función NSS a una entidad de tipo PERSONA se devuelve el número de seguro social de la PERSONA, que es un valor imprimible de tipo STRING (cadena). Si queremos declarar atributos compuestos como NOMBRE en la figura 21.10, deberemos declararlos como entidades y luego declarar sus atributos componentes como funciones, como se muestra aquí para el atributo NOMBRE:

```
NOMBRE() → ENTITY
NOMBRE(PERSONA) → NOMBRE
NOMBREP(NOMBRE) → STRING
```

```
INIC(NOMBRE) → CHAR
APELLIDO(NOMBRE) → STRING
```

Para declarar un tipo de vínculos, le damos un **nombre de papel** en una dirección y lo definimos también como función. Por ejemplo, los tipos de vínculos CARRERA y ESPECIALIDAD de la figura 21.10 se pueden declarar así:

```
CARRERA(ESTUDIANTE) → DEPARTAMENTO
ESPECIALIDAD(ESTUDIANTE) → DEPARTAMENTO
```

Estas declaraciones especifican que la aplicación de una función CARRERA o ESPECIALIDAD a una entidad ESTUDIANTE deberá devolver como resultado una entidad de tipo DEPARTAMENTO. Para declarar un nombre de papel inverso para los mismos tipos de vínculos, escribimos:

```
CARRERA_EN(DEPARTAMENTO) → ESTUDIANTE INVERSE OF CARRERA
ESPECIALIDAD_EN(DEPARTAMENTO) → ESTUDIANTE INVERSE OF ESPECIALIDAD
```

La cláusula INVERSE OF declara que estas funciones son los inversos de las dos funciones previamente declaradas y que por ello especifican los mismos tipos de vínculos que las otras dos, pero en la *dirección opuesta*. Observe también las flechas dobles (→), que especifican que aplicar la función CARRERA_EN o ESPECIALIDAD_EN a una sola entidad DEPARTAMENTO puede devolver un *conjunto de entidades* de tipo ESTUDIANTE. Esto especifica que los tipos de vínculos son 1:N. Esta notación también puede servir para especificar atributos multivaluados. Si queremos especificar un tipo de vínculos M:N, usamos flechas dobles en *ambas direcciones*, como en el siguiente ejemplo:

```
CURSOS_COMPLETADOS(ESTUDIANTE) → SECCIÓN
ESTUDIANTES_QUE_ASISTIERON(SECCIÓN) → ESTUDIANTE INVERSE OF CURSOS_COMPLETADOS
```

Esto declara el vínculo BOLETA de la figura 21.10 en ambas direcciones. Algunas funciones pueden tener más de un argumento; por ejemplo, si queremos declarar el atributo NOTA del vínculo anterior, escribimos:

```
NOTAS(ESTUDIANTE, SECCIÓN) → CHAR
```

Observe que no es preciso declarar un inverso para cada vínculo. Por ejemplo, los tipos de vínculos DC y CS de la figura 21.10, que relacionan un curso con el departamento que lo ofrece y un curso con sus secciones, se pueden declarar en una sola dirección, como sigue:

```
DEPARTAMENTO_QUE_OFRECE(CURSO) → DEPARTAMENTO
SECCIONES_DE(CURSO) → SECCIÓN
```

La figura 21.15(a) muestra, en notación MDF, la declaración de una parte del esquema EER de la figura 21.10, y la figura 21.15(b) ilustra una notación diagramática para ese esquema. También existe una notación para especificar vínculos ES-UN, valores derivados y otros conceptos de modelado avanzados de DAPLEX. En la figura 21.15(a) la función ES_UNA_PERSONA especifica un vínculo ES-UN (un vínculo subclase/superclase, en la terminología de EER) entre ESTUDIANTE y PERSONA.

Un concepto fundamental en MDF es la **composición de funciones**. Al escribir NOMBREP(DEPARTAMENTO_QUE_OFRECE(CURSO)), componemos las dos funciones NOMBREP y DEPARTAMENTO_QUE_OFRECE. Esto se denomina **expresión de camino** y se puede escribir

con una *notación de punto* alternativa como `CURSO.DEPARTAMENTO_QUE_OFRECE.NOMBRE`. La composición de funciones puede servir para declarar **funciones derivadas**; por ejemplo, las funciones `NOMBRE(PERSONA)` o `NOMBREP(ESTUDIANTE)` de la figura 21.15(a) se declaran como composiciones de funciones previamente declaradas. Este enfoque también puede usarse para especificar los atributos heredados explícitamente, como se ve en la figura 21.15(a). Cabe señalar que no mostramos funciones derivadas ni vínculos inversos en el diagrama de esquema de la figura 21.15(b), aunque podríamos haberlo hecho.

La composición de funciones es también el principal concepto empleado en los **lenguajes de consulta funcionales**. Ilustramos esto con un ejemplo simple. Supongamos que deseamos obtener los apellidos de todos los estudiantes que estudian la carrera 'Matemáticas'; podríamos hacerlo escribiendo la siguiente consulta:

```
RETRIEVE APELLIDO (NOMBRE(ES_UNA_PERSONA(CARRERA_EN
(DEPARTAMENTO))))
WHERE NOMBREP(DEPARTAMENTO) = 'Matemáticas'
```

Aquí usamos la función `CARRERA_EN`, que es el inverso de `CARRERA`. Como alternativa podríamos usar la función derivada `APELLIDO(ESTUDIANTE)` para acortar la consulta anterior a:

```
RETRIEVE APELLIDO(CARRERA_EN(DEPARTAMENTO))
WHERE NOMBREP(DEPARTAMENTO) = 'Matemáticas'
```

La función inversa `CARRERA_EN(DEPARTAMENTO)` devuelve entidades `ESTUDIANTE`, así que podemos aplicar la función derivada `APELLIDO(ESTUDIANTE)` a esas entidades.

21.6.2 El modelo de datos relacional anidado

El **modelo relacional anidado** elimina la restricción de *primera forma normal* (1FN; véase las Seccs. 6.1 y 12.3.2) del modelo relacional básico. El modelo resultante se conoce también como modelo relacional **no-1FN** o **N1FN**. En el modelo relacional estándar —también llamado modelo *plano*— los atributos deben ser monovaluados y tener dominios atómicos. El modelo relacional anidado permite atributos compuestos y multivaluados, lo que conduce a tuplas complejas con una estructura jerárquica. Esto resulta útil para representar objetos que por su naturaleza están estructurados jerárquicamente. La figura 21.16(a) muestra un esquema de relación anidado (N1FN) `DEPTO` basado en parte de la base de datos `COMPañía`, y la figura 21.16(b) ofrece un ejemplo de tupla N1FN en `DEPTO`. Para definir el esquema `DEPTO` como estructura anidada, podemos escribir lo siguiente:

```
DEPTO = (NÚMD, NOMBRED, GERENTE, EMPLEADOS, PROYECTOS, LUGARES)
EMPLEADOS = (NOMBREE, DEPENDIENTES)
PROYECTOS = (NOMBREP, LUGARP)
LUGARES = (LUGARD)
DEPENDIENTES = (NOMBRED, EDAD)
```

Primero se definen todos los atributos de la relación `DEPTO`. Después se definen cualesquier atributos anidados que tenga `DEPTO`; a saber, `EMPLEADOS`, `PROYECTOS` y `LUGARES`. A continuación, se definen cualesquier atributos anidados de segundo nivel que haya, como `DEPENDIENTES` de `EMPLEADOS`, y así sucesivamente. Todos los nombres de atributos deben ser distintos en la definición de la relación anidada. Observe que los atributos anidados suelen ser atributos *multivaluados compuestos*, lo que da pie a una "relación anidada" dentro de cada tupla. Por ejemplo, el valor del atributo `PROYECTOS` dentro de cada tupla `DEPTO` es

(a) DECLARACIONES DE TIPOS DE ENTIDADES (INCLUIDO EL ATRIBUTO COMPUESTO `NOMBRE`):

```
PERSONA() → ENTITY
CURSO() → ENTITY
DEPARTAMENTO() → ENTITY
ESTUDIANTE() → ENTITY
SECCIÓN() → ENTITY
NOMBRE() → ENTITY
```

DECLARACIONES DE ATRIBUTOS (INCLUIDOS TRES ATRIBUTOS DERIVADOS):

```
NSS(PERSONA) → STRING
FECHAN(PERSONA) → STRING
SEXO(PERSONA) → CHAR
NOMBREP(PERSONA) → NOMBRE
NOMBREP(PERSONA) → NOMBRE
(PERSONA)
INIC(PERSONA) → INIC(NOMBRE(PERSONA))
APELLIDO(PERSONA) → APELLIDO(NOMBRE
(PERSONA))
CLASE(ESTUDIANTE) → STRING
AÑO(SECCIÓN) → STRING
NOMBRED(DEPARTAMENTO) → STRING
TELEFONOD(DEPARTAMENTO) → STRING
OFICINA(DEPARTAMENTO) → STRING
NÚM(CURSO) → STRING
NOMBREC(CURSO) → STRING
DESCC(CURSO) → STRING
NUMSEC(SECCIÓN) → INTEGER
TRIM(SECCIÓN) → STRING
TELONOD(DEPARTAMENTO) → STRING
NÚM(CURSO) → STRING
DESCC(CURSO) → STRING
```

DECLARACIONES DE TIPOS DE VÍNCULOS (INCLUIDOS INVERSOS Y ATRIBUTOS DE VÍNCULOS):

```
CARRERA(ESTUDIANTE) → DEPARTAMENTO
ESPECIALIDAD(ESTUDIANTE) → DEPARTAMENTO
CARRERA_EN(DEPARTAMENTO) → ESTUDIANTE
INVERSE OF CARRERA
ESPECIALIDAD_EN(DEPARTAMENTO) → ESTUDIANTE
INVERSE OF ESPECIALIDAD
CURSOS_COMPLETADOS(ESTUDIANTE) → SECCIÓN
ESTUDIANTES_QUE_ASISTIERON(SECCIÓN) → ESTUDIANTE
INVERSE OF
CURSOS_COMPLETADOS
NOTAS(ESTUDIANTE, SECCIÓN) → CHAR
DEPARTAMENTO_QUE_OFRECE(CURSO) → DEPARTAMENTO
SECCIONES_DE(CURSO) → SECCIÓN
```

DECLARACIONES DE VÍNCULOS ES-UN Y ATRIBUTOS HEREDADOS:

```
ES_UNA_PERSONA(ESTUDIANTE) → PERSONA
NSS(ESTUDIANTE) → NSS(ES_UNA_PERSONA(ESTUDIANTE))
FECHAN(ESTUDIANTE) → FECHAN(ES_UNA_PERSONA(ESTUDIANTE))
SEXO(ESTUDIANTE) → SEXO(ES_UNA_PERSONA(ESTUDIANTE))
NOMBRE(ESTUDIANTE) → NOMBRE(ES_UNA_PERSONA(ESTUDIANTE))
NOMBREP(ESTUDIANTE) → NOMBREP(ES_UNA_PERSONA(ESTUDIANTE))
INIC(ESTUDIANTE) → INIC(ES_UNA_PERSONA(ESTUDIANTE))
APELLIDO(ESTUDIANTE) → APELLIDO(ES_UNA_PERSONA(ESTUDIANTE))
```

Figura 21.15 El modelo de datos funcional. (a) Declaración de parte del esquema EER de la figura 21.10 como funciones. (continúa en la siguiente página)

una relación con dos atributos (`NOMBREP`, `LUGARP`). En la tupla `DEPTO` de la figura 21.16(b), el atributo `PROYECTOS` contiene tres tuplas como valor suyo. Otros atributos anidados pueden ser atributos *multivaluados simples*, como `LUGARES` de `DEPTO`. También es posible tener un atributo anidado que sea *monovaluado y compuesto*, aunque la mayoría de los modelos relacionales anidados tratan este tipo de atributos como si fueran multivaluados.

Cuando se define un esquema de base de datos relacional anidado, éste consiste en varios esquemas de relaciones **externas**; éstos definen el nivel superior de las relaciones anidadas individuales. Además, los atributos anidados se denominan esquemas de relaciones **internas**, ya que definen estructuras relacionales que están *anidadas dentro* de otra relación. En nuestro ejemplo, `DEPTO` es la única relación externa; todas las demás —`EMPLEADOS`, `PROYECTOS`,

(b)

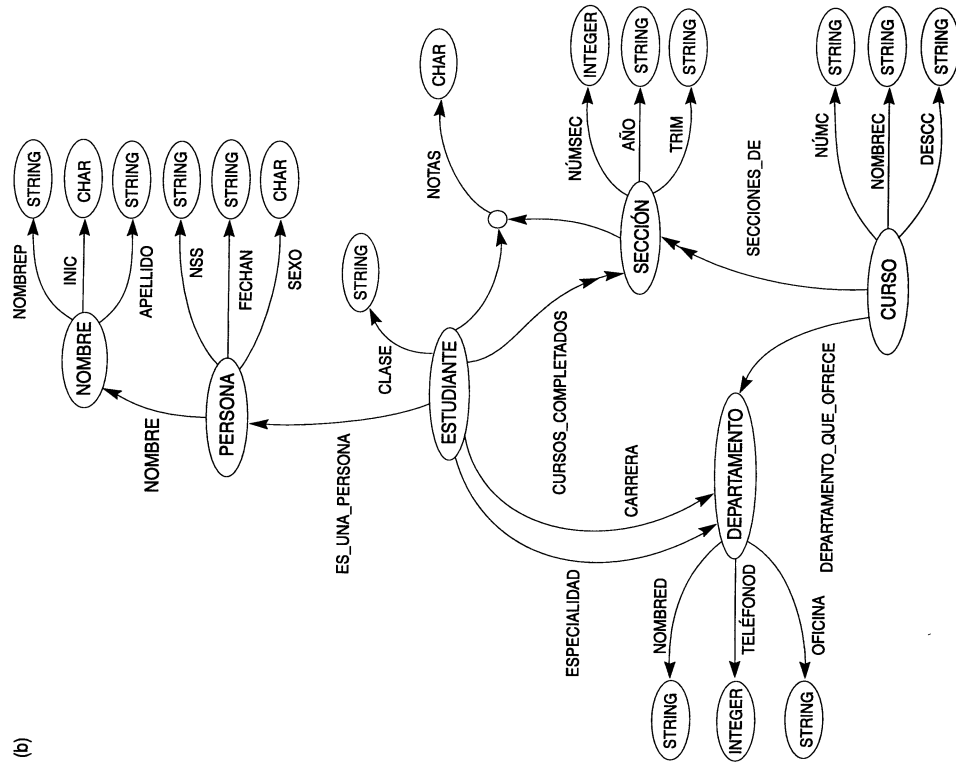


Figura 21.15 (continuación) (b) Representación diagramática del esquema MDF declarado en la figura 21.15(a).

LUGARES Y DEPENDIENTES—son relaciones internas. Por último, aparecen **atributos** en el nivel de hoja y no están anidados. Podemos representar cada esquema de relación por medio de una estructura de árbol, como en la figura 21.16(c), donde la raíz es un esquema de relación externa, las hojas son atributos y los nodos internos son esquemas de relaciones internas. Observe la similitud entre esta representación y un esquema jerárquico (véase el Cap. 11).

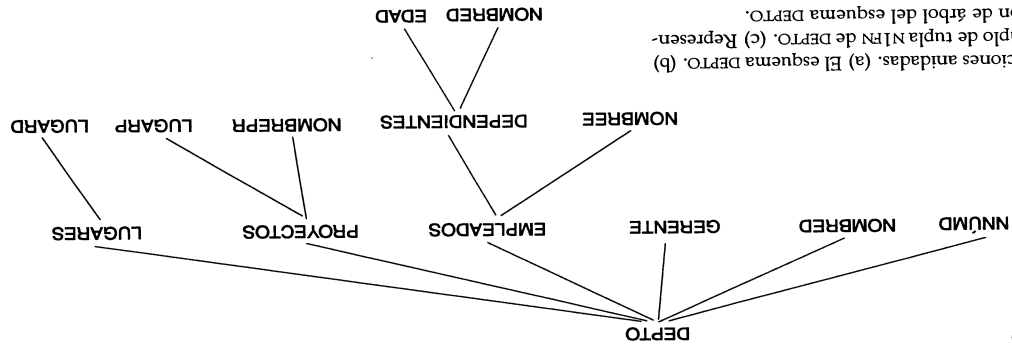
Es importante darse cuenta de que las tres relaciones añadidas de primer nivel de DEPTO representan información *independiente*. Así, EMPLEADOS representa a los empleados que pertenecen al departamento, PROYECTOS representa los proyectos *controlados por* el departamento, y LUGARES representa las diversas ubicaciones de los departamentos. El vínculo entre EMPLEADOS y PROYECTOS no se representa en el esquema; se trata de un vínculo M:N, que es difícil de representar en una estructura jerárquica.

NÚMD	NOMBRED	GERENTE	EMPLEADOS		PROYECTOS			LUGARES
			NOMBREE	DEPENDIENTES	NOMBREPR	LUGARP	LUGARD	
(a)	DEPTO (esquema)							

(a) DEPTO (esquema)

DEL 10 (ejemplo de copia anexo)																																																																									
4	Administración	Valdés	<table><tr><td rowspan="6">Morales</td><td colspan="3" rowspan="3">Valdés</td><td colspan="3" rowspan="3">Zapata</td><td colspan="2">Automatización</td><td rowspan="3">Santiago</td></tr><tr><td colspan="2">Nuevas Prestaciones</td></tr><tr><td colspan="2">Santiago</td></tr><tr><td colspan="3" rowspan="3">Telefonia</td><td colspan="2">Santiago</td><td rowspan="3">Gante</td></tr><tr><td colspan="2">Gante</td></tr><tr><td colspan="2">Santiago</td></tr><tr><td colspan="2"></td><td colspan="2">6</td><td colspan="2">Josefina</td><td colspan="2">8</td><td colspan="2"></td></tr><tr><td colspan="2"></td><td colspan="2">18</td><td colspan="2">Jaime</td><td colspan="2">15</td><td colspan="2">10</td></tr><tr><td colspan="2"></td><td colspan="2"></td><td colspan="2">Roberto</td><td colspan="2"></td><td colspan="2"></td></tr><tr><td colspan="2"></td><td colspan="2"></td><td colspan="2">Maria</td><td colspan="2"></td><td colspan="2"></td></tr></table>							Morales	Valdés			Zapata			Automatización		Santiago	Nuevas Prestaciones		Santiago		Telefonia			Santiago		Gante	Gante		Santiago				6		Josefina		8						18		Jaime		15		10						Roberto										Maria					
Morales	Valdés			Zapata			Automatización		Santiago																																																																
							Nuevas Prestaciones																																																																		
							Santiago																																																																		
	Telefonia			Santiago		Gante																																																																			
				Gante																																																																					
				Santiago																																																																					
		6		Josefina		8																																																																			
		18		Jaime		15		10																																																																	
				Roberto																																																																					
				Maria																																																																					

(b) DEPTO (ejemplo de tupla anidada)



(c)

Relaciones anidadas. (a) El esquema DEPTO. (b) Ejemplo de tupla NI FN de DEPTO. (c) Representación de árbol del esquema DEPTO.

Se han propuesto extensiones del álgebra relacional y del cálculo relacional, y también de SQL, para las relaciones anidadas. El lector interesado en los detalles puede consultar la bibliografía citada al final del capítulo. Aquí ilustraremos dos operaciones, **ANIDAR** y **DESANIDAR**, que pueden servir para aumentar las operaciones estándar del álgebra relacional y así realizar la conversión entre relaciones anidadas y planas. Consideremos la relación plana EMP_PROY de la figura 12.4, y supongamos que la proyectamos sobre los atributos NSS, NÚMEROP, HORAS, NOMBREE como sigue:

$EMP_PROY_PLANA \leftarrow \pi_{NSS, NOMBREE, NÚMEROP, HORAS}(EMP_PROY)$

Para crear una versión anidada de esta relación, donde hay una tupla por cada empleado y (NÚMEROP, HORAS) están anidados, usamos la operación **ANIDAR** como sigue:

$EMP_PROY_ANIDADA \leftarrow ANIDAR_{PROYS = (NÚMEROP, HORAS)}(EMP_PROY_PLANA)$

El efecto de esta operación es crear una relación anidada interna PROYS = (NÚMEROP, HORAS) dentro de la relación externa EMP_PROY_ANIDADA. Así pues, **ANIDAR** agrupa las tuplas que tienen el mismo valor para los atributos que no se especifican en la operación **ANIDAR**; en nuestro ejemplo, éstos son los atributos NSS y NOMBREE. Para cada uno de estos grupos, que representa un empleado en nuestro ejemplo, se crea una sola tupla anidada con una relación anidada interna PROYS = (NÚMEROP, HORAS). Así, la relación EMP_PROY_ANIDADA se ve como la relación EMP_PROY que se muestra en la figura 12.9(a) y (b). Observe la similitud entre la anidación y la agrupación para funciones agregadas (véase la Sec. 6.6.1). En la primera cada grupo de tuplas se convierte en una sola tupla anidada, pero en la segunda cada grupo se convierte en una sola tupla resumida después de aplicarse una función agregada al grupo.

La operación **DESANIDAR** es la inversa de **ANIDAR**. Podemos reconvertir EMP_PROY_ANIDADA a EMP_PROY_PLANA como sigue:

$EMP_PROY_PLANA \leftarrow DESANIDAR_{PROYS = (NÚMEROP, HORAS)}(EMP_PROY_ANIDADA)$

Aquí, el atributo anidado PROYS se aplanará para dar sus componentes NÚMEROP, HORAS.

21.6.3 El modelo de datos estructural

En diversos modelos de datos se ha propuesto extender el modelo relacional original con restricciones y semántica adicionales. En algún momento, algunas de las extensiones propuestas se incorporaron al modelo relacional y a SQL2. Presentamos un panorama sobre el **modelo de datos estructural** como ejemplo representativo de estas extensiones del modelo relacional primigenio. El modelo estructural usó dos tipos de estructuras: relaciones y conexiones. El concepto de **relación** es similar al del modelo relacional estándar (plano). Sin embargo, las relaciones se clasificaron en varios tipos para indicar su comportamiento de actualización y sus conexiones con otras relaciones.

Cada relación en el modelo estructural tiene una **parte gobernante**; ésta corresponde al concepto de clave primaria en el modelo relacional descrito en el capítulo 6. El modelo estructural original definía seis tipos de relaciones: primarias, referidas, dependientes, de asociación, de léxico y subrelaciones. Las conexiones se clasificaron en tres tipos: propiedad, referencia y conexión de identidad. Ilustraremos estos conceptos con el esquema que se muestra en la figura 21.17, que es la representación en el modelo estructural de un esquema de base de datos **COMPANÍA** simplificado.

Las relaciones **primarias** son aquellas a las que no se hace referencia; por tanto, se puede eliminar una tupla de una relación primaria sin que el usuario tenga que verificar tuplas de referencia en otras relaciones. En la figura 21.17, **PROYECTO** es la única relación primaria. Las relaciones **referidas** tienen un atributo en alguna otra relación que hace referencia a su parte gobernante; por tanto, corresponden a relaciones a las que hace referencia una clave externa. Sin embargo, en las relaciones referidas, el **comportamiento** de la clave externa es por definición **rechazar** en caso de eliminación. Esto significa que si se intenta eliminar una tupla referida, la eliminación se rechaza a menos que ninguna tupla de la relación de referencia se refiera realmente a la tupla por eliminar. En la figura 21.17, **DEPTO** y **EMPLEADO** son relaciones referidas, las cuales se conectan a sus relaciones de referencia a través de una **conexión de referencia**; ésta se indica con una arista dirigida (flecha) a la relación referida, como en la figura 21.17. Observe que ésta es la misma notación gráfica que usamos para todas las **claves externas** en la figura 6.7; en el modelo estructural, sólo las claves externas con semántica de **rechazar al eliminar** se definen con una conexión de referencia.

Las relaciones **dependientes** por lo regular corresponden a atributos multivaluados o tipos de entidades débiles en el modelo ER, y suelen representarse con una **relación interna anidada** en el modelo relacional anidado. En el modelo estructural, una relación dependiente es propiedad de otra relación. En la figura 21.17, **DEPENDIENTE** es una relación dependiente propiedad de **EMPLEADO**. Una relación dependiente incluye un atributo de clave externa *dentro de su parte gobernante* que hace referencia a la relación propietario; el **comportamiento** definido para la clave externa es **propagar** al eliminar, lo que significa que si se elimina la tupla propietario (referida), las tuplas dependientes (de referencia) se eliminan automáticamente. Las relaciones de asociación por lo regular representan vínculos binarios M:N o n-arios. Su

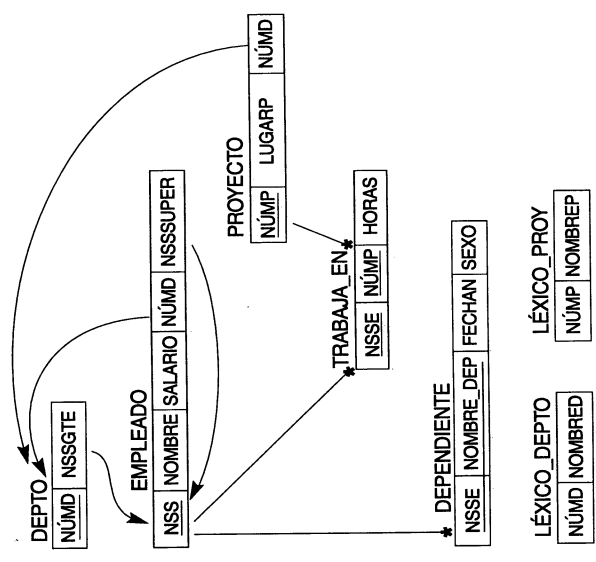


Figura 21.17 Esquema **COMPANÍA** simplificado en el modelo estructural.

parte gobernante está totalmente formada por *dos o más* claves externas, cada una de las cuales hace referencia a una relación propietario. Las claves externas se definen también con el comportamiento *propagar al eliminar*. En la figura 21.17, *TRABAJA_EN* es una relación de asociación.

Las **conexiones de propiedad** sirven para conectar una relación propietario con una relación dependiente o de asociación poseída, por lo que representa una clave externa desde la relación poseída al propietario, con semántica de *propagar al eliminar*. Se representa con una línea con un asterisco en el extremo de la relación (dependiente o de asociación) poseída, como se aprecia en la figura 21.17. La diferencia principal entre las relaciones dependiente y las de asociación es que las primeras sólo tienen un propietario (y por tanto representa un vínculo estrictamente jerárquico), en tanto que una asociación tiene dos o más propietarios.

Las relaciones de **léxico** sirven para almacenar correspondencias uno a uno entre pares de atributos. Por ello, las claves secundarias se pueden pasar a un léxico si así se desea. Esto lo ilustran las relaciones *LÉXICO_DEPTO* y *LÉXICO_PROY* de la figura 21.17. Por último, las subrelaciones y las conexiones de identidad se usan para representar vínculos clase/subclase, especialización y generalización. Una **conexión de identidad** representa una clave externa —que también es una clave primaria— de una **subrelación** a otra relación, con semántica de *propagar al eliminar*.

El modelo estructural se propuso antes de que se incorporaran las claves externas en el modelo relacional. Los conceptos de conexiones definieron claves secundarias con diferentes tipos de comportamientos. Varios de los conceptos del modelo estructural —y conceptos similares propuestos por otros investigadores— finalmente se integraron al modelo relacional estándar y a SQL2.

21.6.4 El modelo de datos semántico

El **modelo de datos semántico** (o *SDM: semantic data model*) introdujo los conceptos de clases y subclases en el modelado de datos; así pues, fue el origen de muchos de los conceptos que desde entonces se han incorporado en los modelos de datos conceptuales, como los modelos orientados a objetos (véase el Cap. 22). También clasificó las clases cuyos objetos representaban información con diversos tipos de semánticas. En esta sección, ofrecemos una introducción muy breve a algunos de los conceptos de SDM.

El principal concepto de modelado de SDM es la **clase**, que es una colección de objetos del mismo tipo. Las **propiedades** (atributos) de una clase especifican el tipo de objetos que contiene. Las propiedades se clasifican como *opcionales* (se permiten nulos) u *obligatorias* (no se permiten nulos); *simples* (atómicas) o *compuestas*; *monovaluadas* o *multivaluadas*; *derivables* o *almacenadas*; y *únicas* o *no únicas*. Los objetos existen independientemente de cualquier valor de sus atributos. Cada propiedad está asociada a un dominio (conjunto de valores) del cual se pueden escoger sus valores para objetos individuales. Si el dominio de una propiedad es otra clase, los valores de la propiedad se refieren a objetos de la otra clase.

Las clases de objetos también se clasifican en varios tipos. Una **clase de objetos concretos** representa objetos con una existencia concreta en el mundo. Una **clase de objetos abstractos** representa grupos de objetos de otras clases con propiedades idénticas. Por ejemplo, una clase *AVIONES* que contiene un objeto por cada avión individual propiedad de una aerolínea es una clase de objetos concretos, en tanto que una clase *TIPOS_DE_AVIONES* que

contiene un objeto por cada tipo de avión (*B737*, *B747*, *MD-11*, *DC-9*, ...) es una clase de objetos abstractos. Una **clase agregada** contiene objetos que son agregados de otros objetos; por ejemplo, cada objeto de una clase agregada *CONVOYES_DE_BUQUES* consiste en un agregado de objetos de la clase de objetos concretos *BUQUES*. Una **clase de sucesos** incluye objetos temporales, como *VIAJES* o *LLEGADAS*.

Una **subclase** es un subconjunto de objetos de una **clase base**. Por ejemplo, *BUQUES_TANQUE* y *BUQUES_CRUCERO* son subclases de una clase base *BUQUES*. Una **subclase de restricción** está definida por predicado, lo que no sucede con las **subclases sin restricción**. Además de las subclases definidas por los usuarios, las clases de objetos abstractos y de agregación por lo regular definen ciertas subclases. Por ejemplo, cada objeto de la clase de objetos abstractos *TIPOS_DE_AVIONES* define una subclase de objetos de *AVIONES* que pertenecen a un cierto tipo.

Con esto termina nuestro breve bosquejo de algunos conceptos de SDM. Históricamente, el SDM introdujo muchos conceptos que se han adoptado en modelos de datos avanzados. Algunos de los conceptos del modelo EER (véase la Sec. 21.1) tienen sus orígenes en conceptos de SDM.

21.7 Resumen

En este capítulo analizamos primero extensiones del modelo ER que mejoran sus capacidades de representación. Llamamos al modelo resultante modelo ER extendido o modelo EER. Se presentaron los conceptos de subclase y su superclase, y el mecanismo asociado de herencia de atributos. Vimos cómo en ocasiones es necesario crear clases de entidades adicionales, ya sea por atributos específicos adicionales o por tipos de vínculos específicos. Examinamos dos procesos principales para definir jerarquías y retículas superclase/subclase, a saber: especialización y generalización.

Después mostramos cómo representar estas nuevas construcciones en un diagrama EER. También vimos los diversos tipos de restricciones que se pueden aplicar a la especialización o a la generalización. Las dos restricciones principales son *total*/*parcial* y *disjunta*/*traslapada*. Además, puede especificarse un predicado de definición para una subclase o un atributo de definición para una especialización. Comentamos las diferencias entre las subclases definidas por el usuario y las definidas por predicado, y entre las especializaciones definidas por el usuario y las definidas por atributo. Finalmente, estudiamos el concepto de categoría, que es un subconjunto de la unión de dos o más clases, y dimos definiciones formales de todos los conceptos presentados.

Después, en la sección 21.2, mostramos cómo pueden transformarse los elementos del modelo EER a las estructuras relacionales. Estas transformaciones pueden usarse durante la fase lógica del diseño de bases de datos.

En la sección 21.3 examinamos varios conceptos de representación de datos abstractos. Los conceptos tratados fueron la clasificación y la generación de ejemplares, la generalización y la especialización, la identificación, la agregación y la asociación. Vimos la relación entre los conceptos del modelo EER y cada uno de ellos. También examinamos brevemente la disciplina de la representación de conocimientos y su relación con el modelado semántico de los datos.

La sección 21.4 ofreció una clasificación general de los tipos de restricciones de integridad que suelen utilizarse en el modelado de datos. La sección 21.5 presentó las operaciones

de actualización asociadas al modelo EER e ilustró su uso en la definición de transacciones en el nivel conceptual durante el diseño de bases de datos. Por último, la sección 21.6 nos mostró un somero panorama de cuatro modelos de datos adicionales: funcional, relacional anidado, estructural y semántico.

Preguntas de repaso

- 21.1. ¿Qué es una subclase? ¿Cuándo se necesita una subclase en el modelado de datos?
- 21.2. Defina los siguientes términos: *superclase de una subclase*, *vínculo superclase/subclase*, *vínculo ES-UN*, *especialización*, *generalización*, *categoría*, *atributos específicos*.
- 21.3. Explique el mecanismo de herencia de atributos. ¿Qué utilidad tiene?
- 21.4. Analice las subclases definidas por el usuario y las definidas por predicados, e identifique las diferencias entre las dos.
- 21.5. Analice las especializaciones definidas por el usuario y las definidas por atributo, e identifique las diferencias entre las dos.
- 21.6. Analice los dos tipos principales de restricciones sobre las especializaciones y las generalizaciones.
- 21.7. ¿Qué diferencia hay entre una jerarquía de subclases y una retícula de subclases?
- 21.8. ¿Qué diferencia hay entre la especialización y la generalización? ¿Por qué no mostramos esta diferencia en los diagramas de esquemas?
- 21.9. ¿Qué diferencia hay entre una categoría y una subclase compartida normal? ¿Para qué sirven las categorías? Ilustre su respuesta con ejemplos.
- 21.10. Mencione los diversos conceptos de abstracción de datos y los conceptos de modelado correspondientes en el modelo EER.
- 21.11. ¿Cuál característica de agregación falta en el modelo EER? ¿Cómo podría mejorarse aún más el modelo EER para contar con ella?
- 21.12. Analice los siguientes tipos de restricciones: *de dominio*, *de clave*, *de vínculo* (*estructurales*), *de integridad semántica*. ¿Qué diferencias hay entre los conceptos de cada uno de los siguientes apartados y cuáles son las ventajas y desventajas de cada uno?
 - a. Especificación de restricciones por procedimientos o en forma declarativa.
 - b. Restricciones inherentes, restricciones declaradas implícitamente y restricciones declaradas explícitamente.
 - c. Restricciones de estado y restricciones de transición.
- 21.13. Haga una lista con las operaciones de actualización del modelo EER, y explique cómo se usan al diseñar transacciones de base de datos en el nivel conceptual.
- 21.14. ¿Qué diferencia hay entre el modelo de datos relacional anidado y el modelo relacional estándar (plano)? Describa las operaciones con que se pueden transformar uno en el otro.

Ejercicios

- 21.15. Diseñe un esquema EER para una aplicación de base de datos que le interese. Especifique todas las restricciones que deban cumplirse en la base de datos. Asegúrese de que el esquema tenga por lo menos cinco tipos de entidades, cuatro tipos de vínculos, un tipo de entidades débil, un vínculo superclase/subclase, una categoría y un tipo de vínculos *n-ario* ($n > 2$).
- 21.16. La figura 21.18 muestra un ejemplo de diagrama EER para una pequeña base de datos de un aeropuerto, con la cual se mantiene información sobre aviones, sus dueños, los empleados del aeropuerto y los pilotos en un pequeño aeropuerto privado. A partir de los requerimientos de esta base de datos, se recabó la siguiente información. Cada avión tiene un número de registro [NúmReg], pertenece a un cierto tipo de aviones [DE_TIPO] y se guarda en un cierto hangar [GUARDADO_EN]. Cada tipo de avión tiene un número de modelo [Modelo], una capacidad [Capacidad] y un peso [Peso]. Cada hangar tiene un número [Número], una capacidad [Capacidad] y una ubicación [Lugar]. La base de datos también lleva el control de los dueños de cada avión [POSEE] y de los empleados que han dado mantenimiento al avión [MANTIENE]. Cada ejemplar de vínculo de POSEE relaciona un avión con un dueño e incluye la fecha de compra [FechaC]. Cada ejemplar de vínculo de MANTIENE relaciona un empleado con un registro de servicio [SERVICIO]. Cada avión recibe mantenimientos muchas veces; por tanto, está relacionado a través de [SERVICIO_AVIÓN] con varios registros de servicio. Un registro de servicio incluye como atributos la fecha de mantenimiento [Fecha], el número de horas invertidas en el trabajo [Horas] y el tipo de trabajo realizado [CódTrabajo]. Usamos un tipo de entidades débil [SERVICIO] para representar el servicio a los aviones porque se utiliza el número de registro del avión para identificar un registro de servicio. Un dueño es una persona o una corporación; por ello, usamos una categoría de generalización [DUEÑO] que es un subconjunto de la unión de los tipos de entidades corporación [CORPORACIÓN] y persona [PERSONA]. Tanto los pilotos [PILOTO] como los empleados [EMPLEADO] son subclases de PERSONA. Cada piloto tiene como atributos específicos su número de licencia [NúmLic] y sus restricciones [Restr]; cada empleado tiene como atributos específicos su salario [Salario] y el turno en que trabaja [Turno]. Todas las entidades persona de la base de datos tienen información correspondiente a su número de seguro social [Nss], nombre [Nombre], dirección [Dirección] y número telefónico [Teléfono]. En el caso de las entidades corporativas, los datos que se mantienen incluyen su nombre [Nombre], dirección [Dirección] y número telefónico [Teléfono]. La base de datos también contiene información sobre los tipos de aviones que cada piloto está autorizado a volar [VUELA] y los tipos de aviones a los cuales puede dar mantenimiento cada empleado en [TRABAJA_EN].
 - a. Transforme el esquema EER de la figura 21.18 a un esquema relacional.
 - b. Cree un esquema en el modelo de datos funcional que corresponda al esquema EER de la figura 21.18. Utilice la notación MDF de la figura 21.15(b).
- 21.17. Transforme el esquema EER UNIVERSIDAD de la figura 21.10 a un esquema relacional.
- 21.18. Cree un esquema en el modelo de datos funcional que corresponda al esquema EER UNIVERSIDAD de la figura 21.10. Utilice la notación MDF de la figura 21.15(b).

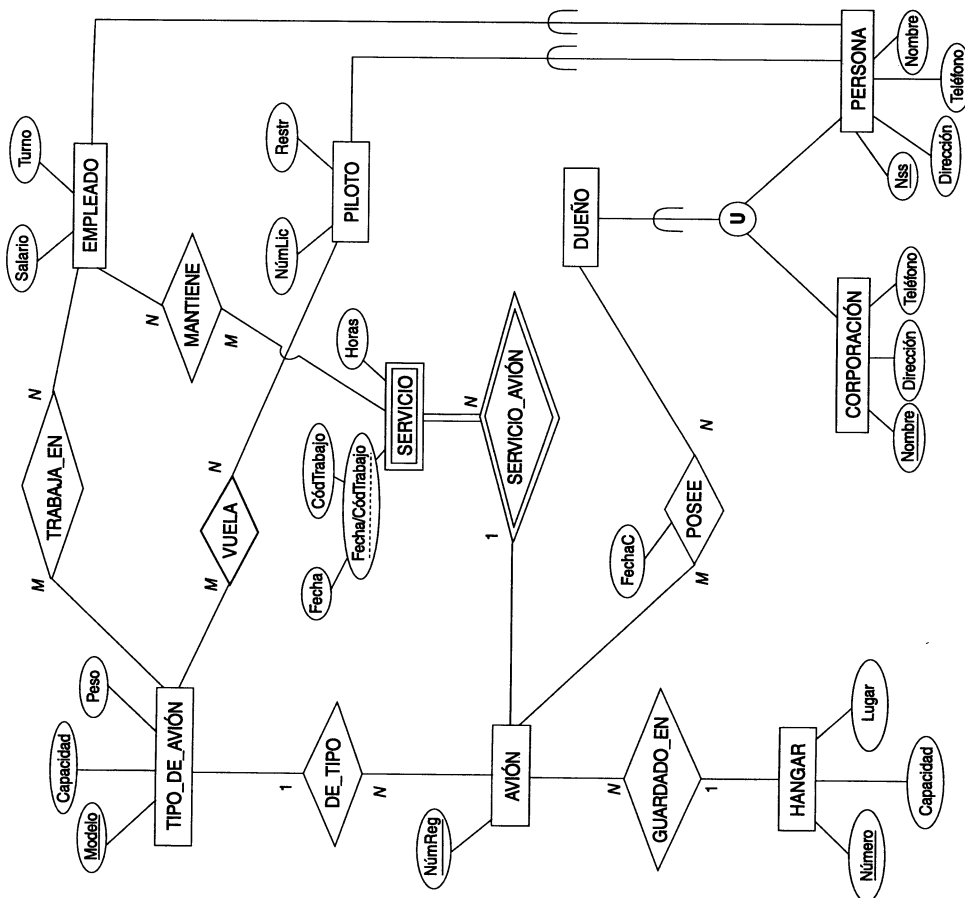


Figura 21.18 Diagrama de esquema ER-extendido para la base de datos de un aeropuerto pequeño.

- 21.19. Considere el esquema ER BANCO de la figura 3.23, y suponga que es necesario llevar el control de diferentes tipos de CUENTAS (CTAS_AHORRO, CTAS_CHEQUES, ...) y PRÉSTAMOS (PRÉST_AUTO, PRÉST_CASA, ...). Suponga que también se desea poder seguir la pista a cada TRANSACCIÓN de cada cuenta (depósitos, retirados, cheques, ...) y a cada PAGO de cada préstamo; los dos incluyen la cantidad, la fecha, la hora, ... Modifique el esquema BANCO utilizando conceptos de ER y EER de especialización y generalización. Exprese todas las suposiciones que haga sobre los requerimientos adicionales.
- 21.20. Compare las relaciones anidadas con las jerarquías del modelo de datos jerárquico (véase el Cap. 11). ¿Le gustaría cambiar el esquema relacional BIBLIOTECA de la figura 6.22 si contara con los elementos del modelo relacional anidado? Explique.

- 21.21. Dibuje un esquema en el modelo de datos funcional y otro en el modelo de datos estructural para el esquema BIBLIOTECA de la figura 6.22.

Bibliografía selecta

Muchos artículos han propuesto modelos de datos conceptuales o semánticos. Aquí daremos una lista representativa. En un grupo de artículos, como Abrial (1974), el modelo DIAM de Senko (1975), el método NIAM (Verheijen y VanBekkum 1982) y Bracchi *et al.* (1976), se presentan modelos semánticos basados en el concepto de vínculos binarios. Otro grupo de artículos pioneros analiza métodos para extender el modelo relacional a fin de mejorar sus capacidades de modelado; entre estos artículos están los de Schmid y Swenson (1975), Navathe y Schkolnick (1978), el modelo RM/T de Codd (1979), Furrado (1978) y el modelo estructural de Wiederhold y Elmasri (1979).

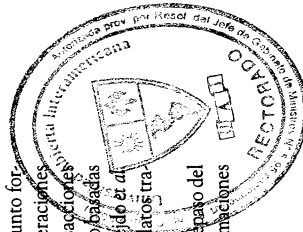
Chen (1976) propuso originalmente el modelo ER, y se formalizó en Ng (1981). Desde entonces, se han propuesto muchas extensiones de su capacidad de modelado, como en Scheuermann *et al.* (1979), Dos Santos *et al.* (1979), Teorey *et al.* (1986), Gogolla y Hohenstein (1991) y el modelo de entidades-categorías-vínculos (ECV) de Elmasri *et al.* (1985). Smith y Smith (1977) explica los conceptos de generalización y agregación. El modelo de datos semántico de Hammer y MacLeod (1981) presentó los conceptos de retículas de clases/subclases, además de otros conceptos avanzados de modelado. Weddell (1992) estudia las dependencias funcionales en el contexto de los modelos semánticos de los datos.

Otra clase de modelos, la de los llamados modelos de datos funcionales, fue propuesta originalmente por Sibley y Kerschberg (1977) y extendida por Shipman (1981); incluyó muchos conceptos de modelado avanzados así como el lenguaje funcional de consulta DAPLEX. Otro lenguaje de consulta funcional es FQL (Buneman y Frankel 1979). El libro de Tschritzky y Lochovsky (1982) examina y compara modelos de datos.

El modelo relacional anidado se trata en Schek y Scholl (1985), Jaeshke y Schek (1982), Chen y Kambayashi (1991) y Makinouchi (1977), entre otros. En Paredaens y VanGucht (1992), Pistor y Andersen (1986), Roth *et al.* (1988) y Ozsoyoglu *et al.* (1987), entre otros, se presentan álgebras y lenguajes de consulta para relaciones anidadas. En Dadam *et al.* (1986), Deshpande y VanGucht (1988) y Schek y Scholl (1989) se describe la implementación de prototipos de sistemas relacionales anidados.

Aunque no vimos lenguajes para el modelo de entidad-vínculo y sus extensiones, se han propuesto varios lenguajes de este tipo. Elmasri y Wiederhold (1981) propone el lenguaje de consulta GORDAS para el modelo ER, y se extiende al modelo ECR en Elmasri *et al.* (1985). Otro lenguaje de consulta ER es el propuesto por Markowitz y Raz (1983). Senko (1980) presenta un lenguaje de consulta para el modelo DIAM de Senko. Parent y Spaccapietra (1985) dio a conocer un conjunto formal de operaciones llamado álgebra ER. Campbell *et al.* (1985) presenta un conjunto de operaciones ER y demuestra que son relacionales completas. El lenguaje TAXIS para especificar transacciones conceptualmente se propuso en Mylopoulos *et al.* (1981). Interfaces amables con el usuario basadas en el modelo ER y otros modelos semánticos se presentan en Elmasri y Larson (1985) y Cayle *et al.* (1987), entre muchos otros. Kent (1978, 1979) estudia las deficiencias de los modelos de datos tradicionales. Bradley (1978) extiende el modelo de red con un lenguaje de alto nivel.

En Hull y King (1987) aparece un repaso del modelado semántico de los datos. Otro repaso del modelado conceptual es Pillalamari *et al.* (1988). Eick (1991) analiza el diseño y las transformaciones de los esquemas conceptuales.



CAPÍTULO 22

Bases de datos orientadas a objetos

En este capítulo trataremos los modelos de datos y los sistemas de bases de datos orientados a objetos. Como hemos visto en el libro, se han propuesto modelos de datos de muchos tipos distintos. En el capítulo 3 presentamos el modelo ER. En la parte 2 explicamos los conceptos y lenguajes del modelo relacional. Los modelos de red y jerárquico se analizaron en la parte 3. En el capítulo 21 incorporamos muchos conceptos de modelado avanzados en el modelo ER, dando lugar al modelo EER, y presentamos breves bosquejos de varios otros modelos de datos: funcional, relacional anidado, estructural y semántico. Muchos de los modelos de datos y sistemas que estudiamos tienen un éxito considerable para desarrollar la tecnología de bases de datos necesaria para las aplicaciones tradicionales en los negocios. Sin embargo, tienen ciertas deficiencias en relación con aplicaciones más complejas, como el diseño y la fabricación en ingeniería (CAD/CAM y CIM¹), las bases de datos gráficas y de imágenes, las bases de datos científicas, los sistemas de información geográfica, las bases de datos de multimedia, y los intentos por ofrecer acceso uniforme a sistemas de múltiples bases de datos. Estas aplicaciones más recientes tienen requerimientos y características que difieren de aquellos de las aplicaciones de negocios tradicionales, como estructuras más complejas para los objetos, transacciones de mayor duración, nuevos tipos de datos para almacenar imágenes o bloques de texto grandes, y la necesidad de definir operaciones no estándar, específicas para cada aplicación. Las bases de datos orientadas a objetos se propusieron con la idea de satisfacer las necesidades de estas aplicaciones más complejas. El enfoque orientado a objetos ofrece la flexibilidad para cumplir con algunos de estos requerimientos sin estar limitado por los tipos de datos y los lenguajes de consulta disponibles en los sistemas de bases de datos tradicionales. Una característica clave de las bases de datos orientadas a objetos es el poder que confieren al diseñador para especificar tanto la estructura de objetos complejos como las operaciones que se pueden aplicar a esos objetos.

¹Diseño asistido por computador/fabricación asistida por computador, y fabricación integrada por computador.

En años recientes, han aparecido muchos prototipos experimentales y sistemas de bases de datos comerciales orientados a objetos. Entre los primeros se cuentan los sistemas ORION, creado en MCC;¹ OpenOODB, de Texas Instruments, IRIS, creado en los laboratorios Hewlett-Packard, ODE, de ATT Bell Labs, y el proyecto ENCORE/ObServer de Brown University. Y entre los sistemas disponibles en el mercado están: GEMSTONE/OPAL de ServioLogic, ONTOS de Ontologic, Objectivity de Objectivity Inc., Versant de Versant Technologies, ObjectStore de Object Design y O2 de O2 Technology. Ésta es sólo una lista parcial de los prototipos experimentales y de los sistemas de bases de datos comerciales orientados a objetos. Desafortunadamente, es aún demasiado pronto para saber cuáles sistemas se instalarán como líderes en este campo. Hemos seleccionado dos sistemas —O2 y ObjectStore— para ilustrar, en la sección 22.7, los sistemas orientados a objetos.

Las bases de datos orientadas a objetos han adoptado muchos de los objetos creados para los lenguajes de programación orientados a objetos. En la sección 22.1 examinaremos los orígenes de este enfoque y analizaremos su aplicación a los sistemas de bases de datos. En seguida presentaremos los conceptos clave utilizados en muchos sistemas orientados a objetos. En la sección 22.2 estudiaremos la identidad y la estructura de los objetos, así como los constructores de tipos. La sección 22.3 presentará el concepto de encapsulamiento de las operaciones y la definición de métodos como parte de las declaraciones de clases. En la sección 22.4 se describirán las jerarquías de tipos y de clases y la herencia en las bases de datos orientadas a objetos, y la sección 22.5 ofrecerá un panorama sobre los problemas que surgen cuando hay que representar y almacenar objetos complejos. En la sección 22.6 se analizarán conceptos adicionales, como el polimorfismo, la sobrecarga de operadores, el enlace dinámico, la herencia múltiple y selectiva y la creación de versiones de objetos.

En la sección 22.7 veremos algunos ejemplos de sistemas orientados a objetos existentes, a fin de ilustrar la definición y manipulación de datos. La sección 22.7.1 ofrecerá un panorama sobre el sistema O2, y la sección 22.7.2, ejemplos del sistema ObjectStore. La sección 22.8 tratará el diseño de bases de datos orientadas a objetos mediante la transformación de un esquema conceptual EER a un esquema orientado a objetos.

El lector podrá pasar por alto parte de las secciones 22.5 a 22.8, o todas ellas, si no desea una introducción detallada al tema.

22.1 Panorama sobre los conceptos de orientación a objetos

Los orígenes del término **orientado a objetos** —abreviado OO u O-O— se remontan a los lenguajes de programación OO. Los conceptos de OO se aplican ahora en las áreas de bases de datos, ingeniería de software, bases de conocimientos, inteligencia artificial y sistemas de cómputo en general. Los lenguajes de programación OO tienen sus raíces en el lenguaje SIMULA, propuesto a finales de la década de 1960. En SIMULA, el concepto de *clase* agrupa la estructura de datos interna de un objeto en una declaración de clase. Más adelante, los investigadores propusieron el concepto de *tipo de datos abstracto*, que oculta las estructuras internas de los datos y especifica todas las posibles operaciones externas aplicables a un objeto, dando lugar al concepto de *encapsulamiento*. El lenguaje de programación SMALLTALK, desarrollado en Xerox PARC¹¹ en los años setenta, fue uno de los primeros lenguajes en incorporar explícitamente conceptos de OO adicionales, como la transferencia de mensajes y la

¹¹Microelectronics and Computer Technology Corporation, Austin, Texas.

¹²Palo Alto Research Center, Palo Alto, California.

herencia. Se le conoce como lenguaje de programación OO *puro*, lo que significa que se le diseñó explícitamente para que fuera orientado a objetos. Esto contrasta con los lenguajes de programación OO *híbridos*, que incorporan conceptos de OO en un lenguaje ya existente. Un ejemplo es C++, que incorpora conceptos de OO al popular lenguaje de programación C.

Los objetos en un lenguaje de programación OO existen sólo durante la ejecución de un programa. Una base de datos OO permite crear objetos que existan permanentemente, o *persistan*, y los puedan compartir muchos programas. Así, las bases de datos OO almacenan permanentemente *objetos persistentes* en almacenamiento secundario, y permiten el compartimiento de tales objetos entre múltiples programas y aplicaciones. Esto requiere la incorporación de otras características bien conocidas de los sistemas de gestión de bases de datos, como los mecanismos de indexación, el control de concurrencia y la recuperación. Un sistema de bases de datos OO se comunica con uno o más lenguajes de programación OO para ofrecer *objetos persistentes* y compartidos.

Un objetivo de las bases de datos OO es mantener una correspondencia directa entre los objetos del mundo real y de la base de datos, de modo que dichos objetos no pierdan su integridad ni su identidad y se puedan identificar y manipular fácilmente. Por ello, las bases de datos OO proveen un *identificador de objeto* (OID: *object identifier*) único, generado por el sistema, para cada objeto. Podemos comparar esto con el modelo relacional, por ejemplo, donde cada relación debe tener un atributo de clave primaria cuyo valor identifique de manera única cada tupla. En el modelo relacional, si se altera el valor de la clave primaria, la tupla tendrá una nueva identidad, aunque siga representando el mismo objeto del mundo real. Por otro lado, un objeto del mundo real puede poseer diferentes claves en diferentes relaciones, lo cual dificulta determinar si las claves representan o no el mismo objeto (por ejemplo, el identificador de objeto se puede representar como `Id_emp` en una relación y como `Nss` en otra).

Otra característica de las bases de datos OO es que los objetos pueden tener una *estructura de complejidad arbitraria* con el fin de contener toda la información significativa que describe al objeto. En contraste, en los sistemas tradicionales la información sobre un objeto complejo suele estar *dispersa* entre muchas relaciones o registros, haciendo que se pierda la correspondencia directa entre un objeto del mundo real y su representación en la base de datos.

La estructura interna de un objeto incluye la especificación de *variables de ejemplar*, que contienen los valores que definen el estado interno del objeto. Así pues, una variable de ejemplar es similar al concepto de *atributo*, con la excepción de que las variables de ejemplar pueden estar encapsuladas dentro del objeto y por ello no ser necesariamente visibles para los usuarios externos. Además, las variables de ejemplar pueden tener tipos de datos arbitrariamente complejos. Los sistemas orientados a objetos permiten definir las operaciones o funciones aplicables a objetos de un tipo en particular. De hecho, algunos modelos OO insisten en que todas las operaciones que un usuario pueda aplicar a un objeto deben estar predefinidas. Esto hace obligatorio un *encapsulamiento* completo de los objetos. A últimas fechas, este enfoque rígido se ha relajado en casi todos los modelos de datos OO, pues implica que cualquier obtención simple requerirá una operación predefinida.

A fin de fomentar el encapsulamiento, las operaciones se definen en dos partes. La primera, llamada *signatura o interfaz* de la operación, especifica el nombre de la operación y sus argumentos (o parámetros). La segunda, llamada *método o cuerpo*, especifica la *implementación* de la operación. Las operaciones pueden invocarse pasándole un *mensaje* a un objeto, que incluya el nombre de la operación y los parámetros. Después el objeto ejecutará el

método para esa operación. Este encapsulamiento permite modificar la estructura interna de un objeto y la implementación de sus operaciones sin necesidad de alterar los programas externos que invocan estas operaciones. Así, el encapsulamiento provee una forma de independencia con respecto a los datos y las operaciones (véase el Cap. 2).

Otros conceptos clave en los sistemas OO son los de jerarquías de tipos y de clases, y de *herencia*. Esto permite especificar nuevos tipos y clases que heredan gran parte de su estructura y operaciones de tipos o clases previamente definidos. Así, la especificación de los tipos de objetos puede efectuarse sistemáticamente, lo cual facilita la creación incremental de los tipos de datos de un sistema, y la *reutilización* de las definiciones de tipos existentes cuando se crean nuevos tipos de objetos.

Un problema de los sistemas de bases de datos OO atañe a la representación de *vínculos* entre los objetos. La insistencia en un encapsulamiento completo en los primeros modelos de datos OO dio pie al argumento de que los vínculos no se deberían representar explícitamente, sino describirse definiendo métodos apropiados que localizaran los objetos relacionados. Sin embargo, este enfoque no funciona muy bien en bases de datos complejas con múltiples vínculos, porque resulta útil identificar estos vínculos y hacerlos visibles para los usuarios. Muchos modelos de datos OO actuales permiten representar vínculos por medio de *referencias*; es decir, colocando los OID de objetos relacionados dentro de un objeto.

Algunos sistemas OO cuentan con capacidades para manejar *múltiples versiones* del mismo objeto, característica esencial en aplicaciones de diseño e ingeniería. Por ejemplo, conviene conservar una versión anterior de un objeto que representa un diseño probado y verificado hasta que la nueva versión se haya probado y verificado. Una nueva versión de un objeto complejo podría incluir sólo unas cuantas versiones nuevas de sus objetos componentes, mientras que otros componentes permanecerían sin alteración. Además de permitir múltiples versiones, las bases de datos OO idealmente deberían permitir la *evolución de esquemas*, que ocurre cuando se modifican las declaraciones de tipos o cuando se crean nuevos tipos o vínculos.

Otro concepto de OO es el *polimorfismo de operadores*, que se refiere a la posibilidad de aplicar una operación a diferentes tipos de objetos; en una situación así, un *nombre de operación* se puede referir a varias *implementaciones* distintas, dependiendo del tipo de objetos al que se aplique. Esta característica se denomina también *sobrecarga de operadores*. Por ejemplo, una operación para calcular el área de un objeto geométrico podría diferir en su método (implementación) dependiendo de si el objeto es de tipo triángulo, círculo o rectángulo. Esto puede requerir el empleo del *enlace tardío* del nombre de la operación al método apropiado en el momento de la ejecución, cuando ya se conoce el tipo de objeto al que se aplicará la operación.

En las secciones 22.2 a 22.6 estudiaremos con detalle los conceptos principales de las bases de datos OO.

22.2 Identidad de objetos, estructura de objetos y constructores de tipos

En esta sección trataremos el concepto de identidad de los objetos, y luego presentaremos las operaciones de estructuración comunes para definir la estructura del valor de un objeto. Estas operaciones de estructuración suelen denominarse **constructores de tipos**; definen las operaciones básicas de estructuración de datos que pueden combinarse para formar objetos de complejidad arbitraria.

22.2.1 Identidad de objetos

Un sistema de base de datos OO provee una **identidad única** a cada objeto independiente almacenado en la base de datos. Esta identidad única suele implementarse con un **identificador de objeto** único, generado por el sistema, u OID. El valor de un OID no es visible para el usuario externo, sino que el sistema lo utiliza a nivel interno para identificar cada objeto de manera única y para crear y manejar las referencias entre objetos.

La principal propiedad que debe tener un OID es la de ser **inmutable**; es decir, el valor del OID para un objeto en particular nunca debe cambiar. Esto preserva la identidad del objeto del mundo real que se está representando. También es preferible que cada OID se utilice sólo una vez; esto es, aunque un objeto se elimine de la base de datos, su OID no se deberá asignar a otro objeto. Estas dos propiedades implican que el OID no debe depender del valor de ningún atributo del objeto, pues estos valores pueden cambiar. También suele considerarse inapropiado basar el OID en la dirección física del objeto en el almacenamiento, ya que una reorganización de los objetos de la base de datos podría cambiar los OID. Sin embargo, algunos sistemas sí usan la dirección física como OID para aumentar la eficiencia de la obtención de los objetos. Si la dirección física cambia, puede colocarse un *apuntador indirecto* en la dirección anterior, dando la nueva ubicación física del objeto. Un sistema de bases de datos OO debe contar con algún mecanismo para generar los OID con la propiedad de inmutabilidad.

Algunos modelos de datos OO requieren que todo se represente como un objeto, ya sea un valor simple o un objeto complejo; así, todo valor básico, como un entero, una cadena o un valor booleano, tiene un OID. Con ello dos valores básicos pueden tener diferentes OID, lo cual es muy útil en algunos casos. Por ejemplo, en algunas ocasiones se podría usar el valor entero 50 para representar un peso en kilogramos, y en otras para referirse a la edad de una persona. Así, podrían crearse dos objetos básicos con diferentes OID, y ambos tendrían el mismo valor básico de 50. Aunque resulta útil como modelo teórico, esto no es muy práctico porque puede obligar a generar demasiados OID. Por ello, la mayor parte de los sistemas de bases de datos OO permiten representar tanto objetos como **valores**. Todo objeto debe tener un OID inmutable, pero los valores no tienen OID y se representan a sí mismos.

22.2.2 Estructura de objetos

En las bases de datos OO, los valores (o estados) de los objetos complejos se pueden construir a partir de otros objetos mediante ciertos **constructores de tipos**. Una forma de representar tales objetos es considerar a cada objeto como una tripleta (i, c, v) , donde i es un *identificador de objeto* único (el OID), c es un *constructor* (esto es, una indicación de cómo se construye el valor del objeto) y v es el *valor* (o estado) del objeto. Puede haber varios constructores, según el modelo de datos y el sistema OO. Los tres constructores más básicos son los constructores de **átomos**, **tuplas** y **conjuntos**. Otros constructores de uso común son los de **listas** y de **arreglos**. También existe un **dominio** D que contiene todos los valores atómicos básicos que están disponibles directamente en el sistema. Por lo regular, éstos incluyen los enteros, los números reales, las cadenas de caracteres, los tipos booleanos, las fechas y cualesquier otros tipos de datos que el sistema maneje directamente.

El valor v de un objeto se interpreta a partir del constructor c de la tripleta (i, c, v) que representa al objeto. Si c = átomo, el valor v es un valor atómico del dominio D de valores básicos que maneja el sistema. Si c = conjunto, el valor v es un conjunto de identificadores de

objetos $\{i_1, i_2, \dots, i_n\}$ que son los identificadores (OID) de un conjunto de objetos que normalmente son del mismo tipo. Si c = tupla, el valor v es una tupla de la forma $\langle a_1, i_1, a_2, i_2, \dots, a_n, i_n \rangle$, donde cada a_i es un nombre de atributo (a veces llamado *nombre de variable de ejemplo* en la terminología OO) y cada i_i es un identificador de objeto (OID). Si c = lista, el valor v es una *lista ordenada* de identificadores de objetos $\{i_1, i_2, \dots, i_n\}$ del mismo tipo. En el caso c = arreglo, el valor es un arreglo (matriz) de identificadores de objetos.

El modelo anterior permite una anidación arbitraria de constructores de conjuntos, listas, tuplas y de otro tipo. Todos los valores de los objetos no atómicos se refieren a otros objetos mediante sus identificadores de objeto; por tanto, el único caso en el que aparece un valor real es en el de los objetos atómicos. No resulta práctico generar un identificador único para cada valor, así que la mayor parte de los sistemas permiten tanto los OID como los *valores estructurados*. Los valores se pueden estructurar con los mismos constructores de tipos que los objetos, excepto que los valores *no tienen* OID.

Los constructores de tipos **conjunto**, **lista**, **arreglo** y **bolsa** se denominan **tipos de colección** o **tipos masivos**, para distinguirlos de los tipos básicos y de los tipos de tupla. Una lista es similar a un conjunto, con la excepción de que los OID de una lista están *ordenados*, de modo que podemos referirnos al primer objeto de una lista, al segundo o al i -ésimo. Una bolsa también es similar a un conjunto, excepto que permite la existencia de valores repetidos. La característica principal de un tipo de colección es que un valor es una *colección de objetos* que puede carecer de estructura (como un conjunto o una bolsa) o tenerla (como una lista o un arreglo).

Supongamos, para el ejemplo que sigue, que todo es un objeto, incluidos los valores básicos. Supongamos también que contamos con los constructores átomo, conjunto y tupla. A continuación representaremos algunos de los objetos de la base de datos relacional que se muestra en la figura 6.6, empleando el modelo anterior. Usaremos $i_1, i_2, i_3, \dots$ para representar identificadores de objeto únicos, generados por el sistema. Un objeto se define por una tripleta (OID, constructor del tipo, valor). Consideremos los siguientes objetos:

$$\begin{aligned} o_1 &= (i_1, \text{átomo, Higuera}) \\ o_2 &= (i_2, \text{átomo, Belén}) \\ o_3 &= (i_3, \text{átomo, Sacramento}) \\ o_4 &= (i_4, \text{átomo, 5}) \\ o_5 &= (i_5, \text{átomo, Investigación}) \\ o_6 &= (i_6, \text{átomo, 22-MAY-78}) \\ o_7 &= (i_7, \text{conjunto, } \{i_1, i_2, i_3\}) \\ o_8 &= (i_8, \text{tupla, } \langle \text{NOMBRED: } i_5, \text{NÚMEROD: } i_4, \text{GTE: } i_3, \text{LUGARES: } i_7, \\ &\quad \text{EMPLEADOS: } i_{10}, \text{PROYECTOS: } i_{11} \rangle) \\ o_9 &= (i_9, \text{tupla, } \langle \text{GERENTE: } i_{12}, \text{FECHAINCIOGERENTE: } i_6 \rangle) \\ o_{10} &= (i_{10}, \text{conjunto, } \{i_{12}, i_{13}, i_{14}\}) \\ o_{11} &= (i_{11}, \text{conjunto, } \{i_{15}, i_{16}, i_{17}\}) \end{aligned}$$

Los primeros cinco objetos de la lista ($o_1 - o_5$) son simplemente valores atómicos. Hay muchos objetos similares, uno para cada valor atómico constante distinto en la base de datos. Estos objetos atómicos son los que pueden provocar un problema, debido al empleo de demasiados identificadores de objetos, si es que este modelo se implementa directamente. El objeto o_7 es un objeto con valor de conjunto que representa el conjunto de ubicaciones del departamento 5; el conjunto $\{i_1, i_2, i_3\}$ se refiere a los objetos atómicos con los valores {Higuera, Belén, Sacramento}. El objeto o_8 es un objeto con valor de tupla que representa el departamento 5 mismo, y tiene los atributos NOMBRE, NÚMERO, GTE, LUGARES, etc. Los primeros dos atributos, NOMBRE y NÚMERO tienen los valores atómicos o_3 y o_4 como valores. El atributo GTE tiene un objeto de tupla o_9 como valor, el cual a su vez tiene dos atributos. El valor del atributo GERENTE es el objeto cuyo OID es i_{12} (no se muestra), y es el objeto empleado que dirige el departamento, en tanto que el valor de FECHAINICIO-GERENTE es otro valor atómico cuyo valor es una fecha. El valor del atributo EMPLEADOS de o_8 es un objeto de conjunto con $OID = i_{10}$, cuyo valor es el conjunto de identificados de objetos de los empleados que pertenecen a ese DEPARTAMENTO (los objetos i_{12}, i_{13}, i_{14} que no se muestran). De manera similar, el valor del atributo PROYECTOS de o_8 es un objeto de conjunto con $OID = i_{11}$, cuyo valor es el conjunto de identificadores de objeto de los proyectos controlados por el departamento número 5 (los objetos i_{15}, i_{16}, i_{17} que no se muestran).

En este modelo, un objeto puede representarse como una estructura de grafo, ya que se puede construir aplicando repetidamente los tres constructores básicos. El grafo que representa un objeto o_i se puede construir creando primero un nodo para el objeto o_i mismo. El nodo de o_i se rotula con el OID y el constructor c del objeto. También creamos un nodo en el grafo para cada valor básico en el dominio D de todos los valores atómicos. Si el objeto tiene un valor atómico, dibujamos una arista dirigida del nodo que representa o_i hasta el nodo que representa su valor básico. Si el valor del objeto está construido, dibujamos aristas dirigidas del nodo del objeto a un nodo que representa el valor construido. En general, el grafo de un *objeto individual* no debe tener ciclos, pues eso indicaría un objeto que se contiene a sí mismo como componente. Sin embargo, el grafo que representa un tipo de objetos puede tener ciclos que representen vínculos recursivos. Por ejemplo, en un grafo de tipos, un objeto EMPLEADO puede hacer referencia a un objeto SUPERVISOR, que también es de tipo EMPLEADO. Sin embargo, en el grafo de los objetos EMPLEADO individuales, el supervisor debe ser un objeto empleado distinto, y por tanto no hay ciclos. La figura 22.1 muestra el grafo del ejemplo de objeto DEPARTAMENTO que se dio antes.

El modelo anterior permite dos tipos de definiciones en una comparación de los valores de dos objetos para determinar igualdad. Se dice que dos objetos tienen valores idénticos si los grafos que representan sus valores son idénticos en todo sentido, incluidos los OID en todos los niveles. Otra definición de la igualdad, menos estricta, dice que dos objetos son iguales si tienen valores iguales. En este caso, las estructuras de grafo deben ser las mismas, y todos los valores atómicos correspondientes dentro de los grafos deben ser los mismos. Sin embargo, algunos nodos internos correspondientes en los dos grafos pueden tener objetos con distintos OID.

Un ejemplo simple puede ilustrar la diferencia entre las dos definiciones al comparar objetos para determinar igualdad. Consideremos los siguientes objetos o_1, o_2, o_3, o_4, o_5 y o_6 :

$$\begin{aligned} o_1 &= (i_1, \text{tupla}, \langle a_1, i_4, a_2, i_6 \rangle) \\ o_2 &= (i_2, \text{tupla}, \langle a_1, i_5, a_2, i_6 \rangle) \\ o_3 &= (i_3, \text{tupla}, \langle a_1, i_4, a_2, i_6 \rangle) \end{aligned}$$

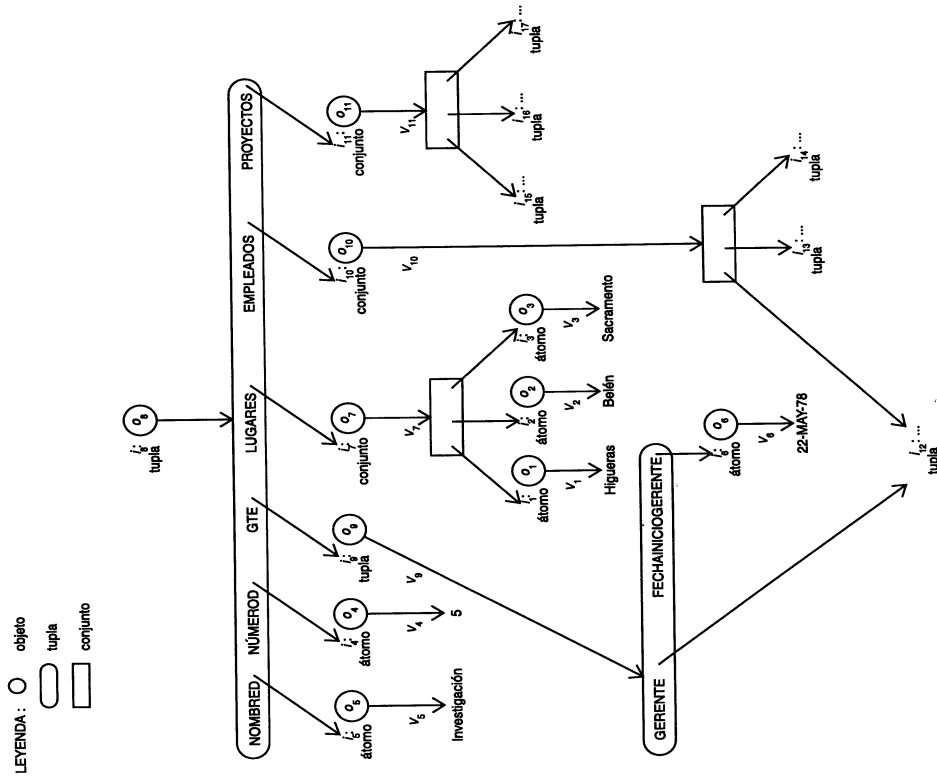


Figura 22.1 Representación gráfica de un objeto complejo.

$o_4 = (i_4, \text{átomo}, 10)$
 $o_5 = (i_5, \text{átomo}, 10)$
 $o_6 = (i_6, \text{átomo}, 20)$

Los objetos o_1 y o_2 tienen valores *iguales*, ya que sus valores en el nivel atómico son los mismos, aunque se llega a esos valores a través de objetos distintos, o_4 y o_5 . En cambio, los valores de los objetos o_1 y o_3 son *idénticos*. De manera similar, o_4 y o_5 son iguales pero no idénticos, porque tienen diferentes OID.

22.2.3 Constructores de tipos

Podemos usar un **lenguaje de definición de datos OO** (OODL: *object-oriented data definition language*) que incorpore los constructores de tipos anteriores para definir tipos de objetos para una aplicación de base de datos específica. Así, estos constructores de tipos pueden servir para definir las estructuras de datos de un *esquema de base de datos* OO. Veremos en la sección 22.3 cómo incorporar la definición de operaciones (o métodos) en el esquema OO. La figura 22.2 muestra cómo podemos declarar tipos Empleado y Departamento a partir de los objetos de la figura 22.1. También definimos un tipo Fecha como una tupla (en vez de un valor atómico, como en la figura 22.1). Usaremos las palabras reservadas *tuple*, *set* y *list* para los constructores de tipos de tupla, conjunto y lista, respectivamente, y usaremos los tipos de datos estándar disponibles (integer, string, real, etc.) para los tipos atómicos. Usaremos nuestra propia sintaxis —parecida a la de O2— debido a la carencia actual de una sintaxis de OODL estándar.

Los atributos que hacen referencia a otros objetos —como depto de Empleado o proyectos de Departamento— son básicamente **referencias**, y por tanto sirven para representar **vínculos** entre los tipos de objetos. Un vínculo binario se puede representar en una dirección, o puede tener una **referencia inversa**. Esta última representación hace más fácil recorrer el vínculo en ambas direcciones. Por ejemplo, el atributo empleados de Departamento tiene como valor un **conjunto de referencias** a objetos de tipo Empleado; éstos son los

```
define type tuple (
Empleado:
  nombre: string,
  nss: string,
  fechanac: Fecha,
  sexo: char,
  depto: Departamento);

define type tuple (
Fecha:
  año: integer,
  mes: integer,
  día: integer);

define type tuple (
Departamento:
  nombred: string,
  numerod: integer,
  gte: tuple (gerente: Empleado,
               fechalnic: Fecha),
  lugares: set (string),
  empleados: set (Empleado),
  proyectos: set (Proyecto));
```

Figura 22.2 Empleo de OODL para definir los tipos Empleado, Fecha y Departamento.

empleados que pertenecen al departamento. El atributo de referencia inversa depto de Empleado se refiere al departamento específico al que pertenece un empleado. Veremos más adelante que algunos SGBDOO permiten declarar explícitamente los inversos (véase la Sec. 22.7.2), para asegurar que las referencias inversas sean consistentes.

22.3 Encapsulamiento de operaciones, métodos y persistencia

El concepto de encapsulamiento es una de las características principales de los lenguajes y sistemas OO. También está relacionado con los conceptos de tipos de datos abstractos y de ocultación de información en los lenguajes de programación. En las bases de datos tradicionales no se aplicaba este concepto, pues era costumbre hacer que la estructura de los objetos de la base de datos fuera visible para los usuarios y programas externos. Por lo regular, varias operaciones de base de datos estándar son aplicables a objetos de cualquier tipo. Por ejemplo, en el modelo relacional, las operaciones para seleccionar, insertar, eliminar y modificar tuplas son genéricas y se pueden aplicar a cualquier tipo de relación de la base de datos. La relación y sus atributos son visibles para los usuarios y para los programas externos que tienen acceso a las tuplas y atributos de las relaciones mediante estas operaciones.

Los conceptos de ocultación de la información y encapsulamiento se pueden aplicar a los objetos de una base de datos. La idea principal es definir el comportamiento de un tipo de objeto con base en las operaciones que se pueden aplicar externamente a objetos de ese tipo. La estructura interna del objeto queda oculta, y sólo se puede tener acceso a él a través de una serie de operaciones predefinidas. Algunas operaciones pueden servir para crear o destruir objetos, otras pueden actualizar el valor (o el estado) del objeto, y otras pueden servir para obtener partes del valor del objeto o para aplicar cálculos a dicho valor. Otras operaciones más pueden llevar a cabo una combinación de obtención, cálculo y actualización. En general, la *implementación* de una operación puede especificarse en un *lenguaje de programación de propósito general* que ofrezca flexibilidad y potencia para definir las operaciones.

Los usuarios externos del objeto sólo perciben la **interfaz** del objeto, la cual define los nombres y argumentos (parámetros) de cada operación. La **implementación** del objeto queda oculta a los usuarios externos; incluye la definición de las estructuras de datos internas del objeto y la implementación de las operaciones que tienen acceso a dichas estructuras. En la terminología de OO, la parte de interfaz de cada operación se denomina **signatura**, y la implementación de la operación se llama **método**. Por lo regular, un método se invoca enviando un **mensaje** al objeto para que ejecute el método correspondiente. Cabe señalar que, como parte de la ejecución de un método, puede enviarse un mensaje subsiguiente a otro objeto, y este mecanismo puede servir para devolver valores de los objetos al entorno externo o a otros objetos.

En las aplicaciones de base de datos, el requisito de que todos los objetos queden totalmente encapsulados es demasiado estricto. Una forma de relajarlo consiste en dividir la estructura de un objeto en atributos (variables) **visibles** y **ocultos**. Los operadores externos, o un lenguaje de consulta de alto nivel, pueden tener acceso directo a los atributos visibles para leerlos. Los atributos ocultos de un objeto quedan completamente encapsulados, y sólo puede tenerse acceso a ellos a través de operaciones predefinidas. Casi todos

```
define class Empleado:
type
    tuple (
        nombre: string,
        rns: string,
        fechanac: Fecha,
        sexo: char,
        depto: Departamento);

operations
    edad (e: Empleado): integer,
    crear_nuevo_emp: Empleado,
    destruir_emp (e: Empleado): boolean;

define class Departamento
type
    tuple (
        nombred: string,
        numerod: integer,
        gte: tuple (gerente: Empleado,
                    fecha: Fecha));

        lugares: set (string),
        empleados: set (Empleado),
        proyectos: set (Proyecto));

operations
    numero_de_emps (d: Departamento): integer,
    crear_nuevo_depto: Departamento,
    destruir_depto (d: Departamento): boolean,
    añadir_emp (d: Departamento, e: Empleado): boolean,
    quitar_emp (d: Departamento, e: Empleado): boolean
    (* añade un nuevo empleado al departamento *)
    (* quita un empleado del departamento *);
```

Figura 22.3 Empleo de OODL para definir las clases Empleado y Departamento.

los SOBDOO emplean lenguajes de consulta de alto nivel para tener acceso a los atributos visibles, y varias propuestas han intentado extender el lenguaje SQL para usarlo con bases de datos OO.

En la mayoría de los casos, las operaciones que *actualizan* el estado de un objeto están encapsuladas. Ésta es una forma de definir la semántica de actualización de los objetos, en vista de que en muchos modelos de datos OO pocas restricciones de integridad están predefinidas en el esquema. Cada tipo de objetos tiene sus restricciones de integridad *programadas* en los *métodos* que crean, eliminan y actualizan los objetos. En tales casos, todas las operaciones de actualización se implementan con operaciones encapsuladas. El término *clase* se usa para referirse a una definición de tipo de objetos, junto con las definiciones de las operaciones para ese tipo.¹ La figura 22.3 muestra cómo se puede extender el OODL de la figura 22.2 para definir clases. Se declaran varias operaciones para cada clase de objetos, y la *signatura* (interfaz) de cada operación se incluye en la definición de la clase. Los métodos (implementaciones) de cada operación se deben definir en algún otro lugar, empleando un lenguaje de programación. Dos operaciones usuales son la operación de *constructor de objeto* —con que se crea un objeto nuevo— y la operación de *destructor* —con que se

¹En el modelo EER, el término *clase* se usó para referirse a un tipo de objetos, junto con el conjunto de todos los objetos de ese tipo.

destruye un objeto—. También es posible declarar varias operaciones de *modificador de objeto* para modificar diversos atributos (variables) de un objeto.

Es común que un SOBDOO esté acoplado íntimamente a un lenguaje de programación OO. Con este lenguaje se especifican las implementaciones de los métodos. Lo normal es que los objetos los creen programas en ejecución, al invocar la operación de construcción de los objetos. No todos los objetos están destinados a almacenarse permanentemente en la base de datos; hay **objetos transitorios** en el programa en ejecución que desaparecen una vez que éste termina. Los **objetos persistentes** se almacenan en la base de datos y persisten después de la terminación del programa. El mecanismo de persistencia usual implica dar al objeto un **nombre persistente** único a través del cual los programas puedan obtener el objeto, o bien hacer que el objeto sea *alcanzable* desde algún objeto persistente. Se dice que un objeto *B* es **alcanzable** desde un objeto *A* si una secuencia de referencias del grafo de objetos conduce del objeto *A* al objeto *B*. Por ejemplo, todos los objetos de la figura 22.1 son alcanzables desde el objeto *o_g*; por tanto, si se hace que *o_g* sea persistente, todos los demás objetos de la figura 22.1 se volverán persistentes también.

Si creamos primero un objeto persistente *N* con nombre, cuyo valor es un *conjunto* o lista de objetos de alguna clase *C*, podemos hacer que los objetos de *C* sean persistentes *añadiéndolos* al conjunto o lista, haciéndolos así alcanzables desde *N*. De este modo, *N* define una **colección persistente** de objetos de clase *C*. Por ejemplo, demos definir una clase ConjuntoDeptos (véase la Fig. 22.4) cuyos objetos sean de tipo set(Departamento). Supongamos que se crea un objeto de tipo ConjuntoDeptos, y que se le da el nombre TodosDeptos, haciéndolo así persistente, como se ilustra en la figura 22.4. Cualquier objeto Departamento que se añada al conjunto de TodosDeptos con la operación añadir_depto se vuelve persistente en virtud de ser alcanzable desde TodosDeptos.

Observe la diferencia entre los modelos de bases de datos estándar y las bases de datos OO en este aspecto. En un modelo típico, como el modelo EER o el relacional, se da por hecho que *todos* los objetos son persistentes. Así pues, cuando se define un tipo de entidades

```
define class ConjuntoDeptos
type
    set (Departamento);

operations
    crear_conj_deptos: ConjuntoDeptos,
    destruir_conj_deptos (cd: ConjuntoDeptos): boolean,
    añadir_depto (cd: ConjuntoDeptos, d: Departamento): boolean,
    quitar_depto (cd: ConjuntoDeptos, d: Departamento): boolean;

persistent name TodosDeptos: ConjuntoDeptos;
    (* TodosDeptos es un objeto persistente nombrado de tipo set(Departamento) *)
...
d := crear_nuevo_depto;
... (* crea un nuevo objeto departamento en la variable d *)
b := añadir_depto(TodosDeptos, d);
    (* hace a d persistente añadiéndolo al objeto persistente nombrado TodosDeptos *)
```

Figura 22.4 Creación de objetos persistentes dándoles nombres y haciéndolos alcanzables.

o una subclase como EMPLEADO en el modelo EER, representa tanto la *declaración de tipo* de EMPLEADO como un *conjunto persistente* de todos los objetos EMPLEADO. En el enfoque OO representativo, una declaración de clase de EMPLEADO sólo especifica el tipo y las operaciones de una clase de objetos. El usuario debe definir por separado un objeto persistente de tipo set(EMPLEADO) o list(EMPLEADO) cuyo valor sea la *colección de referencias* a todos los objetos EMPLEADO persistentes, si esto es lo que se desea, como se ilustra en la figura 22.4. De hecho, es posible definir varias colecciones persistentes para la misma definición de clase, si así se desea. La razón principal de esta diferencia es que permite a los objetos transitorios y persistentes seguir las mismas declaraciones de tipos y clases del OODL y del lenguaje de programación OO. Como aquí nos interesan sobre todo las aplicaciones de bases de datos, suponemos que, por cada declaración de clase, el nombre de la clase se refiere tanto al tipo y a las definiciones de operaciones como al *conjunto de todos los objetos persistentes* de esa clase. Esto hace que nuestro empleo de la palabra *clase* sea congruente con el que le dimos en el capítulo 21.

22.4 Jerarquías de tipos y de clases y herencia

Otra característica importante de los sistemas OO es que deben permitir jerarquías de tipos o de clases y herencia. Las jerarquías de tipos y las de clases son conceptualmente distintas, pero como en última instancia a menudo originan estructuras que se comportan de manera similar, es común que se utilicen indistintamente. En nuestra exposición hablaremos primero de las jerarquías de tipos (en la Sec. 22.4.1) y luego de las jerarquías de clases (en la Sec. 22.4.2). Usaremos un *modelo OO diferente* en esta sección —un modelo en el que los atributos y las operaciones se tratan de manera uniforme— porque tanto los atributos como las operaciones se pueden heredar. Por añadidura, en esta sección el término *clase* se referirá al *conjunto de objetos* de un cierto tipo.

22.4.1 Jerarquías de tipos y herencia

En la mayor parte de las aplicaciones de bases de datos hay un gran número de objetos del mismo tipo. Por ello, las bases de datos OO deben proporcionar una capacidad para clasificar los objetos con base en su tipo, como hacen otros sistemas de bases de datos. Sin embargo, en las bases de datos OO hay un requisito adicional: que el sistema permita la definición de nuevos tipos basados en otros tipos predefinidos, dando origen a una *jerarquía de tipos*.

Por lo regular, para definir un tipo se le asigna un nombre de tipo y se define después una serie de atributos (variables de ejemplar) y operaciones (métodos) para ese tipo. En algunos casos, los atributos y operaciones juntos se denominan *funciones*, a fin de simplificar la terminología. Subsecuentemente, puede usarse un nombre de función para hacer referencia al valor de un atributo o a la implementación de una operación (método). En esta sección usaremos el término **función** para referirnos tanto a los atributos como a las operaciones de un tipo de objetos.

Para definir un tipo se le da un **nombre de tipo** y se listan en seguida los nombres de sus **funciones**. En esta sección, al especificar un tipo, usaremos el siguiente formato simplificado:

NOMBRE_DE_TIPO: función, función, ..., función

Por ejemplo, un tipo que describe características de una PERSONA se puede definir como sigue:

PERSONA: Nombre, Dirección, FechaNac, Edad, NSS

En el tipo PERSONA, las funciones Nombre, Dirección, NSS y FechaNac se implementan como atributos almacenados, en tanto que la función Edad se implementa como un método que calcula la edad a partir del valor del atributo FechaNac y la fecha actual.

El concepto de **subtipo** resulta útil cuando el diseñador o usuario debe crear un nuevo tipo que es similar, pero no idéntico, a un tipo ya definido. El subtipo **hereda** entonces todas las funciones del tipo predefinido, al cual llamaremos **supertipo**. Por ejemplo, suponemos que deseamos definir dos nuevos tipos EMPLEADO y ESTUDIANTE como sigue:

EMPLEADO: Nombre, Dirección, FechaNac, Edad, NSS, Salario, FechaContrato, Antigüedad
ESTUDIANTE: Nombre, Dirección, FechaNac, Edad, NSS, Carrera, Promedio

Puesto que tanto ESTUDIANTE como EMPLEADO incluyen todas las funciones definidas para PERSONA más algunas funciones propias adicionales, podemos declararlos como **subtipos** de PERSONA. Los dos heredarán las funciones que definimos antes para PERSONA; a saber, Nombre, Dirección, FechaNac, Edad y NSS. En el caso de ESTUDIANTE sólo será necesario definir las nuevas funciones Carrera y Promedio, que no se heredan. Es de suponer que Carrera se podrá definir como atributo almacenado, y que Promedio se podrá implementar como un método que tenga acceso a los valores Notas almacenados internamente dentro de cada objeto ESTUDIANTE. En el caso de EMPLEADO, las funciones Salario y FechaContrato pueden ser atributos almacenados, en tanto que Antigüedad puede ser un método que calcule la antigüedad a partir del valor de FechaContrato.

La idea de definir un tipo implica definir todas sus funciones e implementirlas ya sea como atributos o como métodos. Cuando se define un subtipo, puede heredar todas estas funciones y su implementación. Sólo es necesario definir e implementar las funciones que son específicas para el subtipo, y que por ello no se implementan en el supertipo. Por tanto, podemos declarar EMPLEADO y ESTUDIANTE como sigue:

EMPLEADO subtype-of PERSONA: Salario, FechaContrato, Antigüedad
ESTUDIANTE subtype-of PERSONA: Carrera, Promedio

Al declarar que EMPLEADO y ESTUDIANTE son *subtipos de* (subtype-of) PERSONA, estamos especificando que los dos heredan todas las funciones de PERSONA. En general, un subtipo incluye *todas* las funciones que están definidas para su supertipo, más algunas funciones adicionales que son específicas para el subtipo. Así pues, es posible generar una **jerarquía de tipos** para indicar los vínculos supertipo/subtipo entre todos los tipos declarados en el sistema.

Como ejemplo adicional, consideremos un tipo que describe objetos de la geometría plana, el cual puede definirse así:

OBJETO_GEOMÉTRICO: Forma, Área, PuntoReferencia

En el caso del tipo OBJETO_GEOMÉTRICO, Forma se implementa como atributo (sus valores pueden ser triángulo, rectángulo, círculo, etc.) y Área es un método que se aplica para calcular el área. Supongamos ahora que deseamos definir varios subtipos del tipo OBJETO_GEOMÉTRICO, como sigue:

RECTÁNGULO subtype-of OBJETO_GEOMÉTRICO: Ancho, Altura
 TRIÁNGULO subtype-of OBJETO_GEOMÉTRICO: Lado1, Lado2, Ángulo
 CÍRCULO subtype-of OBJETO_GEOMÉTRICO: Radio

Observe que la operación Área se puede implementar con un método diferente para cada subtipo, puesto que el procedimiento de cálculo del área es diferente para los rectángulos, los triángulos y los círculos. De manera similar, el atributo PuntoReferencia puede tener un significado distinto para cada subtipo; podría ser el punto central de los objetos RECTÁNGULO y CÍRCULO, y el punto vértice entre los dos lados dados en el caso de un objeto TRIÁNGULO. Algunos sistemas OO permiten **cambiar el nombre** de las funciones heredadas en los diferentes subtipos para que reflejen mejor el significado (véase la Sec. 22.7.1).

Una forma alternativa de declarar estos tres subtipos sería especificar el valor del atributo Forma como una condición que debe satisfacerse para los objetos de cada subtipo:

RECTÁNGULO subtype-of OBJETO_GEOMÉTRICO (Forma='rectángulo'): Ancho, Altura
 TRIÁNGULO subtype-of OBJETO_GEOMÉTRICO (Forma='triángulo'): Lado1, Lado2, Ángulo
 CÍRCULO subtype-of OBJETO_GEOMÉTRICO (Forma='círculo'): Radio

Aquí, sólo los objetos OBJETO_GEOMÉTRICO para los que Forma = 'rectángulo' son del subtipo RECTÁNGULO, y de manera similar con los otros dos subtipos. En este caso, cada uno de los tres subtipos hereda todas las funciones del supertipo OBJETO_GEOMÉTRICO, pero el valor del atributo Forma está restringido a un valor específico para cada uno.

Adviértase que las definiciones de tipos no generan objetos por sí mismas. Son sólo declaraciones de ciertos tipos; y como parte de esa declaración, se especifica la implementación de las funciones de cada tipo. En una aplicación de bases de datos hay muchos objetos de cada tipo. Cuando se crea un objeto, por lo regular pertenece a uno o más de estos tipos que se han declarado. Por ejemplo, un objeto círculo es de tipo CÍRCULO y OBJETO_GEOMÉTRICO (por herencia). Además, cada objeto se hace miembro de una o más clases de objetos, que sirven para agrupar colecciones de objetos que son significativas para la aplicación de base de datos. Analizaremos las clases de objetos y las jerarquías de clases en la siguiente sección.

22.4.2 Jerarquías de clases

Una **clase** es una colección de objetos que es significativa para alguna aplicación.[†] En la mayor parte de las bases de datos OO, la colección de objetos de una clase tiene el mismo tipo; sin embargo, esto no es una condición necesaria. Por ejemplo, SMALLTALK, un lenguaje OO caracterizado como *sin tipos*, permite que una clase contenga una colección de objetos sea cual sea el tipo de éstos. Esto puede suceder también cuando otros lenguajes sin tipos no orientados a objetos, como LISP, se extienden con conceptos OO.

Por lo regular, una clase se define por su nombre y por la colección de objetos incluidos en ella. A menudo conviene poder definir una **subclase** de otra clase de objetos, donde esta última sea la **superclase**. En este caso, la restricción es que todos los objetos de la subclase deben ser también miembros de la superclase. Algunos sistemas OO tienen una clase preddefinida del sistema (llamada clase ROOT (raíz) o clase OBJECT (objeto)) que contiene todos los

[†]Usamos el término *clase* en la sección 22.3 para indicar una declaración de tipo y las operaciones aplicables. En esta sección, *clase* se refiere a una colección de objetos.

objetos del sistema. Entonces, la clasificación se lleva a cabo especializando los objetos para generar clases adicionales que sean significativas para la aplicación, creando así una **jerarquía de clases** del sistema. Todas las clases definidas por el sistema y por el usuario son subclases de la clase OBJECT, directa o indirectamente.

Cabe señalar que los constructores de tipos que vimos en la sección 22.2 permiten que el valor de un objeto sea una colección de objetos, la cual es, en esencia, una clase. Así, una clase de objetos cuyos valores se basan en el *constructor de conjunto* define varias colecciones, una correspondiente a cada objeto. Los objetos mismos con valor de conjunto son miembros de otra clase. Esto hace posible que haya esquemas de clasificación de múltiples niveles, donde un objeto de una clase define como su valor una clase de objetos.

En las bases de datos OO basadas en tipos (es decir en la mayoría de las bases de datos OO), una jerarquía de clases a menudo tiene una jerarquía de tipos correspondiente, por la restricción de que todos los objetos de una clase sean del mismo tipo. Así, es posible crear una *clase por omisión* para cada tipo (o subtipo), que contenga todos los objetos persistentes de ese tipo.[†] Tales sistemas siempre tendrían una jerarquía de clases que correspondería a la jerarquía de tipos. Debido a esto, es frecuente que no haya una distinción clara entre la jerarquía de tipos y la de clases; más bien, las dos se combinan en una sola jerarquía. Cada clase tiene entonces un cierto tipo y contiene la colección de todos los objetos persistentes de ese tipo.

En la mayor parte de los sistemas OO, se hace una distinción entre los objetos y **clases persistentes** y **transitorios**. Una **clase persistente** es una clase cuya colección de objetos se almacena permanentemente en la base de datos, de modo que múltiples programas pueden tener acceso a ella y compartirla. Una **clase transitoria** es una clase cuya colección de objetos existe temporalmente durante la ejecución de un programa, pero que no se conserva cuando el programa termina. Por ejemplo, puede crearse una clase transitoria en un programa para contener el resultado de una consulta que selecciona algunos objetos de una clase persistente y los copia en la clase transitoria. Esta última tiene el mismo tipo que la clase persistente. Así, el programa puede manipular los objetos de la clase transitoria; y una vez que el programa termina, la clase transitoria deja de existir. En general, varias clases —transitorias o persistentes— pueden contener objetos del mismo tipo.

22.5 Objetos complejos*

Una de las razones fundamentales para crear los sistemas OO fue el deseo de representar objetos complejos. Hay dos clases principales de objetos complejos: estructurados y no estructurados. Un objeto complejo estructurado está constituido por objetos componentes que se ensamblan aplicando recursivamente, en diversos niveles, los constructores de tipos disponibles. Los objetos complejos no estructurados casi siempre son tipos de datos que requieren una gran cantidad de almacenamiento, como un tipo de datos que representa una imagen o un objeto textual extenso. En las secciones 22.5.1 y 22.5.2 analizaremos los diversos tipos de objetos complejos.

22.5.1 Objetos complejos no estructurados y extensibilidad de tipos

Un recurso de **objeto complejo no estructurado** provisto por un SORBD permite almacenar y leer objetos extensos que necesitan la aplicación de base de datos. Ejemplos representativos de

[†]La mayoría de los sistemas OO no crean tales clases automáticamente.

tales objetos son las imágenes de mapa de bits y las cadenas de texto largas; también se conocen como **objetos binarios extensos**, que se abrevia **BLOB** (*binary large object*). Estos objetos carecen de estructura en el sentido de que el SGBD no sabe qué estructura tienen; sólo la aplicación que usa los objetos puede interpretar su significado. Por ejemplo, la aplicación puede tener funciones para exhibir una imagen o para buscar ciertas palabras clave en una cadena de texto larga. Los objetos se consideran complejos porque requieren un área de almacenamiento sustancial y no forman parte de los tipos de datos estándar que suelen ofrecer los SGBD. Puesto que el tamaño de los objetos es considerable, un SGBD podría obtener una porción del objeto y proporcionarla al programa de aplicación antes de obtener todo el objeto. El SGBD podría también usar técnicas de almacenamiento intermedio y *cache* para obtener por anticipado porciones del objeto, antes de que el programa de aplicación necesite tener acceso a ellas.

El software del SGBD no cuenta con la capacidad de procesar directamente las condiciones de selección ni otras operaciones basadas en valores de estos objetos, a menos que la aplicación provea el código necesario para realizar las operaciones de comparación que precisa la selección. En un SGBD, esto puede lograrse definiendo un nuevo tipo de datos abstracto para los objetos no interpretados y suministrando los métodos para seleccionar, comparar y exhibir tales objetos. Por ejemplo, consideremos objetos que sean imágenes bidimensionales de mapa de bits. Supongamos que la aplicación necesita seleccionar, a partir de una colección de objetos, sólo aquellos que incluyan un cierto patrón. En este caso, el usuario debe proveer el programa de reconocimiento de patrones como método para los objetos de tipo mapa de bits. El SGBD obtendrá entonces un objeto de la base de datos y ejecutará en él el método de reconocimiento de patrones para determinar si el objeto incluye o no el patrón requerido.

Como un SGBD permite a los usuarios crear nuevos tipos, y como un tipo incluye tanto estructura como operaciones, podemos considerar que un SGBD tiene un **sistema de tipos extensibles**. Podemos crear bibliotecas de nuevos tipos definiendo su estructura y operaciones, incluso con los tipos complejos. Las aplicaciones pueden entonces usar o modificar estos tipos, esto último creando subtipos de los tipos provistos en las bibliotecas. Sin embargo, el SGBD deberá contar con las capacidades de almacenamiento y obtención subyacentes para los objetos que requieran grandes espacios de almacenamiento, a fin de que las operaciones se puedan aplicar con eficiencia. Muchos SGBD pueden almacenar y obtener objetos no estructurados extensos en forma de cadenas de caracteres o de bits, que se pueden pasar "tal cual" al programa de aplicación para que las interprete.

22.5.2 Objetos complejos estructurados

La diferencia entre un **objeto complejo estructurado** y uno no estructurado consiste en que la estructura del objeto se define por aplicación repetida de los constructores de tipos provistos por el SGBD. Así, la estructura del objeto está definida y el SGBD la conoce. A guisa de ejemplo, consideremos el objeto **DEPARTAMENTO** de la figura 22.1. En el primer nivel, el objeto tiene una estructura de tupla con seis atributos: **NOMBRE**, **NÚMERO**, **GTE**, **LUGARES**, **EMPLEADOS** y **PROYECTOS**. Sin embargo, sólo dos de estos atributos —a saber, **NOMBRE** y **NÚMERO**— tienen valores básicos; los otros cuatro tienen valores complejos y por ende construyen el segundo nivel de la estructura del objeto complejo. Uno de estos cuatro (**GTE**) tiene una estructura de tupla, y los otros tres (**LUGARES**, **EMPLEADOS**, **PROYECTOS**) tienen una estructura de conjunto. En el tercer nivel, para una tupla **GTE**, tenemos un atributo

básico (**FECHAINICIOGERENTE**) y un atributo (**GERENTE**) que se refiere a un objeto empleado, el cual tiene una estructura de tupla. Para un conjunto **LUGARES** tenemos un conjunto de valores básicos, pero para los conjuntos **EMPLEADOS** y **PROYECTOS** tenemos conjuntos de objetos con estructura de tupla.

Existen dos tipos de semántica de referencia entre un objeto complejo y sus componentes en cada nivel. El primer tipo, que llamamos **semántica de propiedad**, se aplica cuando los subobjetos de un objeto complejo están encapsulados dentro de éste y, por ello, se consideran parte de él. El segundo tipo, denominado **semántica de referencia**, se aplica cuando los componentes del objeto complejo son ellos mismos objetos independientes, pero en ocasiones pueden considerarse parte del objeto complejo. Por ejemplo, podemos considerar que los atributos **NOMBRE**, **NÚMERO** y **LUGARES** son propiedad de un **DEPARTAMENTO**, en tanto que **GTE**, **EMPLEADOS** y **PROYECTOS** deben ser referidos porque representan subobjetos independientes. El primer tipo se conoce también como el vínculo *es-parte-de* o *es-componente-de*, y el segundo tipo se denomina vínculo *está-asociado-a*, pues describe una asociación equitativa entre dos objetos independientes. El vínculo *es-parte-de* (semántica de propiedad) para construir objetos complejos tiene la característica de que los objetos componentes están encapsulados dentro del objeto complejo y se consideran parte de la estructura interna del objeto. Sólo los métodos de ese objeto pueden tener acceso a ellos, y se eliminan si el objeto mismo se elimina. Por otro lado, un objeto complejo cuyos componentes sean referidos se considera formado por objetos independientes que pueden tener su propia identidad y métodos. Cuando un objeto complejo necesita tener acceso a sus componentes referidos, debe hacerlo invocando los métodos apropiados de los componentes, ya que no están encapsulados dentro del objeto complejo. Por añadidura, más de un objeto complejo puede hacer referencia a un objeto componente referido, y por ello este último no puede eliminarse automáticamente cuando el objeto complejo sea eliminado.

Los SGBD deben ofrecer opciones de almacenamiento para *formar grupos* con los objetos componentes de un objeto complejo en almacenamiento secundario, a fin de aumentar la eficiencia de las operaciones que tienen acceso al objeto complejo. En muchos casos la estructura del objeto se almacena, sin ser interpretada, en páginas de disco. Cuando se lee a la memoria una página de disco que incluye un objeto, el SGBD puede construir el objeto complejo estructurado a partir de la información contenida en la página, la cual puede hacer referencia a páginas de disco adicionales que será preciso obtener. Esto se conoce como **ensamble de objetos complejos**.

22.6 Otros conceptos de OO*

En esta sección presentaremos un panorama sobre algunos conceptos adicionales de OO, como los de **polimorfismo** (sobrecarga de operadores), herencia múltiple, herencia selectiva, creación de versiones y configuraciones.

22.6.1 Polimorfismo (sobrecarga de operadores)

Otra característica de los sistemas OO es que hacen posible el **polimorfismo** de las operaciones, también conocido como **sobrecarga de operadores**. Este concepto permite enlazar el mismo *nombre* o *símbolo de operador* a dos o más *implementaciones* diferentes del operador, dependiendo del tipo de objetos a los que éste se aplique. Un ejemplo simple de entre los

lenguajes de programación puede ilustrar este concepto. En algunos lenguajes, el símbolo de operador "+" puede significar diferentes cosas cuando se aplica a operandos (objetos) de distintos tipos. Si los operandos de "+" son de tipo *entero*, la operación invocada es la suma de enteros. Si los operandos de "+" son de tipo *conjunto*, la operación invocada es la unión de conjuntos. El compilador puede determinar cuál operación debe ejecutar con base en los tipos de los operandos que el programa le suministra.

En las bases de datos OO puede ocurrir una situación similar. Podemos usar el ejemplo de OBJETO_GEOMÉTRICO que vimos en la sección 22.4 para ilustrar el polimorfismo⁷ en las bases de datos OO. Suponga que declaramos OBJETO_GEOMÉTRICO y sus subtipos como sigue:

```
OBJETO_GEOMÉTRICO: Forma, Área, PuntoCentral
RECTÁNGULO subtype-of OBJETO_GEOMÉTRICO (Forma="rectángulo"): Ancho, Altura
TRIÁNGULO subtype-of OBJETO_GEOMÉTRICO (Forma="triángulo"): Lado1, Lado2, Ángulo
CÍRCULO subtype-of OBJETO_GEOMÉTRICO (Forma="círculo"): Radio
```

Aquí se declara la función Área para todos los objetos de tipo OBJETO_GEOMÉTRICO. Sin embargo, la implementación del método para calcular Área puede diferir en cada subtipo de OBJETO_GEOMÉTRICO. Una posibilidad es tener una implementación general para calcular el área de un OBJETO_GEOMÉTRICO generalizado (por ejemplo, escribiendo un algoritmo general que calcule el área de un polígono) y luego reescribir algoritmos más eficientes para calcular las áreas de diferentes tipos de objetos geométricos, como un círculo, un rectángulo, un triángulo, etc. En este caso, la función Área se *sobrecarga* con diferentes implementaciones.

El SGBD debe ahora seleccionar el método apropiado para la función Área con base en el tipo de objeto geométrico al que se aplique. En los sistemas de tipos estrictos, esto puede hacerse durante la compilación, pues deben conocerse los tipos de los objetos. Esto se denomina **enlace temprano**, porque la elección del método que se usará puede hacerse durante la compilación. Sin embargo, en los sistemas de tipos no estrictos o sin tipos (como SMALLTALK y LISP), es posible que el tipo del objeto al que se aplica una función no se conozca hasta el momento de la ejecución. En este caso, la función deberá verificar el tipo del objeto durante la ejecución y luego invocar el método apropiado. A esto se le llama **enlace tardío** de la función al método de implementación, puesto que el enlace se hace en el momento de la ejecución.

22.6.2 Herencia múltiple y herencia selectiva

La **herencia múltiple** en una jerarquía de tipos se da cuando un cierto subtipo *T* es un subtipo de dos (o más) tipos diferentes y por tanto hereda las funciones (atributos y métodos) de ambos supertipos. Por ejemplo, podemos crear un subtipo GERENTE_DE_INGENIERÍA que sea un subtipo tanto de GERENTE como de INGENIERO. Esto conduce a la creación de una **etiqueta de tipos** más que a una jerarquía de tipos. Un problema que puede presentarse con la herencia múltiple es que los dos supertipos de los cuales el tipo hereda pueden tener funciones distintas con el mismo nombre, lo que da lugar a una ambigüedad. Por ejemplo, es posible que tanto GERENTE como INGENIERO tengan una función llamada Salario. Si esta función se implementa con diferentes métodos en los supertipos GERENTE e INGENIERO, habrá

⁷En los lenguajes de programación hay varios tipos de polimorfismo. Se recomienda al lector interesado consultar las notas bibliográficas, donde se mencionan trabajos con análisis más completos.

una ambigüedad respecto a cuál de los dos heredará el subtipo GERENTE_DE_INGENIERÍA. Es posible, empero, que tanto GERENTE como INGENIERO hereden Salario del mismo supertipo (digamos, EMPLEADO) más arriba en la retícula. En un caso así no habrá ambigüedad; el problema sólo surge cuando las funciones son distintas en los dos supertipos.

Hay varias técnicas para resolver la ambigüedad en la herencia múltiple. Una solución es que el sistema compruebe si hay ambigüedad en el momento de crearse el subtipo, y dejar que el usuario elija explícitamente en este momento cuál función se heredará. Otra solución es usar algún criterio por omisión del sistema. Una tercera solución es prohibir la herencia múltiple si hay ambigüedad, obligando al usuario a cambiar el nombre de una de las funciones en uno de los supertipos. De hecho, algunos sistemas OO no permiten en lo absoluto la herencia múltiple.

La **herencia selectiva** ocurre cuando un subtipo hereda sólo algunas de las funciones de un supertipo; las demás funciones no se heredan. En este caso puede usarse una cláusula EXCEPT (excepto) para listar las funciones de un supertipo que el subtipo no debe heredar. No es usual que los sistemas de bases de datos OO cuenten con el mecanismo de herencia selectiva, pero se utiliza con mayor frecuencia en aplicaciones de inteligencia artificial.

22.6.3 Versiones y configuraciones

Muchas aplicaciones de bases de datos que usan sistemas OO requieren la existencia de varias **versiones** del mismo objeto. Por ejemplo, consideremos una aplicación de bases de datos para un entorno de ingeniería de software que almacena diversos artefactos de software, como serían módulos de diseño, módulos de código fuente, información de configuración que describa cuáles módulos deben enlazarse para formar un programa complejo y casos de prueba para el sistema. Por lo regular, se aplican *actividades de mantenimiento* a un sistema de software conforme sus requerimientos evolucionan. Por lo regular, el mantenimiento implica modificar algunos de los módulos de diseño y de implementación. Si el sistema ya está en operación, y si es preciso modificar uno o más de los módulos, el diseñador deberá crear una **nueva versión** de cada uno de ellos para efectuar los cambios. De manera similar, es posible que sea necesario generar nuevas versiones de los ejemplares de prueba para las nuevas versiones de los módulos. Sin embargo, las versiones existentes no se deberán desechar en tanto no se hayan probado exhaustivamente y aprobado las nuevas versiones; sólo entonces deberán las nuevas versiones reemplazar las anteriores.

Cabe señalar que puede haber más de dos versiones de un objeto. Por ejemplo, consideremos dos programadores que trabajan en forma concurrente para actualizar el mismo módulo de software. En este caso se requieren dos versiones, además del módulo original. Los programadores pueden actualizar concurrentemente sus propias versiones del mismo *módulo de software*. Es común llamar a esto **ingeniería concurrente**. Sin embargo, siempre llega el momento en que es preciso combinar (fusionar) estas dos versiones para que la versión híbrida incluya los cambios realizados por ambos programadores. Durante la combinación, también es necesario asegurarse de que sus cambios sean compatibles. Esto precisa la creación de una versión más del objeto: una que sea el resultado de fusionar las dos versiones que se actualizaron de manera independiente.

Como se deduce del análisis anterior, un SGBD debe ser capaz de almacenar y controlar múltiples versiones del mismo objeto. Varios sistemas ofrecen esta capacidad, con lo cual la aplicación puede mantener múltiples versiones de un objeto y hacer referencia explícitamente a versiones específicas según sea necesario. Sin embargo, el problema de combinar y conciliar los cambios realizados a dos versiones diferentes se deja por lo regular

a los creadores de la aplicación, que conocen la semántica de esta última. Algunos SGBD cuentan con ciertos recursos que pueden comparar las dos versiones con el objeto original y determinar si cualesquier cambios que se hayan realizado son incompatibles, ayudando así al proceso de combinación. Otros sistemas mantienen un **grafo de versiones** que muestra las relaciones entre las versiones. Siempre que se origina una versión v_i al copiarse otra versión v_j , se puede trazar una arista dirigida de v_j a v_i . De manera similar, si dos versiones v_2 y v_3 se combinan para crear una nueva versión v_4 , se trazan aristas dirigidas de v_2 y de v_3 a v_4 . Este grafo puede ayudar a los usuarios a entender las relaciones entre las diferentes versiones, y el sistema puede utilizarlo internamente para controlar la creación y eliminación de versiones.

Cuando la creación de versiones se aplica a objetos complejos, surgen problemas adicionales que es preciso resolver. Un objeto complejo, como un sistema de software, puede constar de muchos módulos. Cuando se permite la creación de múltiples versiones, es posible que cada uno de esos módulos tenga varias versiones distintas y un grafo de versiones. Una **configuración** del objeto complejo es una colección que consiste en una versión de cada módulo dispuesta de manera tal que las versiones del módulo en la configuración sean compatibles y juntas formen una versión válida del objeto complejo. Una nueva versión o configuración del objeto complejo no tiene que incluir nuevas versiones de cada módulo. Por ello, puede ser que ciertas versiones de módulos que no se han alterado pertenezcan a más de una configuración del objeto complejo. Cabe señalar que una configuración es una colección de versiones de *diferentes objetos* que, juntas, constituyen un objeto complejo, en tanto que el grafo de versiones describe la colección de versiones del *mismo objeto*. Una configuración debe seguir la estructura de tipos de un objeto complejo; múltiples configuraciones del mismo objeto complejo son análogas a múltiples versiones de un objeto componente.

22.7 Ejemplos de SGBDOO*

Ahora ilustraremos los conceptos estudiados en este capítulo, para lo cual examinaremos dos SGBDOO. La sección 22.7.1 presenta un panorama sobre el sistema O2 de O2 Technology, y la sección 22.7.2 hace lo propio con el sistema ObjectStore de Object Design, Inc. Como mencionamos al iniciar el capítulo, existen muchos otros SGBDOO comerciales y de prototipo; con estos dos ejemplos ilustraremos sistemas específicos.

22.7.1 Panorama sobre el sistema O2

En nuestro examen del sistema O2, ilustraremos primero la definición de datos y luego consideraremos ejemplos de su manipulación. Por último, analizaremos brevemente la arquitectura del sistema O2.

Definición de datos en O2. En O2, el esquema define los tipos y las clases de objetos del sistema. Un tipo de objetos se define usando los tipos atómicos provistos por O2 y aplicando los constructores de tipos de O2. Los *tipos atómicos* son booleano, de carácter, entero, real, de cadena y de bit. Los *constructores de tipos* son tupla, lista, conjunto y conjunto único. Cuando se especifica solo, el constructor de *conjunto* (*set*) permite elementos duplicados (en forma similar a lo que llamamos *bolsa* en la sección 22.2.2), no así el constructor *conjunto único* (*unique set*) (parecido a lo que llamamos *conjunto* en la sección 22.2.2). En O2, los métodos no se incluyen como parte de una definición de tipos. Más bien, una

definición de clase consta de dos partes: el *tipo* de los objetos que pueden ser miembros de la clase y los *métodos* que se pueden aplicar a dichos objetos.

En O2 se distingue entre valores y objetos. Un *valor* tiene sólo un tipo y se representa a sí mismo. Un *objeto* pertenece a una clase y por tanto tiene un tipo y un comportamiento especificado por los métodos de la clase. Por añadidura, un objeto tiene una *identidad única* (un OID) y un valor actual (o estado), mientras que un valor no tiene OID. Tanto los valores como los objetos pueden tener tipos complejos, y los diferentes niveles del tipo complejo pueden ser valores (lo que representa una semántica de propiedad) o bien pueden hacer referencia a otros objetos valiéndose de sus OID (lo que representa una semántica de referencia). El sistema O2 tiene un lenguaje, O2C, que sirve para definir clases, métodos y tipos, y para crear objetos y valores. Los objetos son *persistentes* (si se almacenan permanentemente en la base de datos) o *transitorios* (si sólo existen durante la ejecución de un programa). Los valores son transitorios a menos que se vuelvan parte de un objeto persistente.

La figura 22.5 muestra declaraciones posibles de tipos y de clases en O2C para una porción de la base de datos UNIVERSIDAD, cuyo esquema EER se dio en la figura 21.10. Hemos

```

type Teléfono: tuple (  cód_área: integer,
                        número: integer );

type Fecha: tuple (
    año: integer,
    mes: integer,
    día: integer );

class Persona
type tuple (
    nss: string,
    nombre: tuple (
        dirección: tuple (
            fechanac: Fecha,
            sexo: character ),
            nombrepila: string,
            paternmo: string,
            maternmo: string ),
            número: integer,
            calle: string,
            núm_apto: string,
            ciudad: string,
            estado: string,
            códpostal: string ),
            fechanac: Fecha,
            sexo: character ),
    edad: integer

method
end

class Estudiante inherit Persona
type tuple (
    clase: string,
    carrera_en: Departamento,
    especialidad_en: Departamento,
    inscrito_en: set (Sección),
    boleta: set (tuple (
        notas: character,
        nnotas: real,
        sección: Sección )))

method
    promedio: notas: real,
    cambiar_clase: boolean,
    cambiar_carrera: (nueva_carrera: Departamento): boolean
end

```

Figura 22.5 Declaraciones de clases en O2 para una parte de la base de datos UNIVERSIDAD de la figura 21.10. (continúa en la siguiente página)

```

class Estudiante_posgrad inherit Estudiante
  type tuple (
    grados: set (tuple (
      colegio: string,
      grado: string,
      año: integer ) ),
    asesor: Profesorado )
end

class Profesorado inherit Persona
  type tuple (
    salario: real,
    rango: string,
    oficina: string,
    teléfono: Teléfono,
    perteneca_a: set (Departamento),
    preside: Departamento,
    subvenciones: set (Subvención),
    asesora: set (Estudiante) )
  method
    promover_profesor,
    dar_aumento (porcentaje: real)
  end

class Departamento
  type tuple (
    nombred: string,
    oficina: string,
    teléfono: Teléfono,
    miembros: set (Profesorado),
    alumnos: set (Estudiante),
    director: Profesorado,
    cursos: set (Curso) )
  method
    añadir_profesor (p: Profesorado),
    añadir_a_carrera (e: Estudiante),
    quitar_de_carrera (e: Estudiante): boolean
  end

class Sección
  type tuple (
    núm_sec: integer,
    trim: Trimestre,
    año: Año,
    estudiantes: set (tuple (est: Estudiante,
      notas: character ) ),
    curso: Curso,
    prof: Profesor )
  method
    cambiar_notas (e: Estudiante, c: character)
  end

class Curso
  type tuple (
    nombrec: string,
    número: string,
    descripción: string,
    secciones: set (Sección),
    depto_que_ofrece: Departamento )
  method
    actualizar_descrip (nueva_d: string)
  end

```

Figura 22.5 (continuación)

incluido unos cuantos métodos para ilustrar la forma de declararlos. En O2 es posible definir una nueva clase *E'* que herede el tipo y los métodos de otra clase *E*. En nuestra terminología, *E'* es una subclase de *E*, y el tipo de *E'* debe ser un subtipo de *E*. Por ejemplo, en la figura 22.5, Estudiante y Profesorado se declaran como subclases de Persona usando el enunciado **inherit** Persona en la definición de Estudiante y Profesorado. Es posible **cambiar el nombre** de los atributos o métodos heredados y **redefinir** para una subclase la estructura de los atributos heredados o la implementación de los métodos heredados. Por ejemplo, si queremos cambiar el nombre del atributo *nss* de Persona a *id* estudiante en la subclase Estudiante de Persona, podemos incluir el siguiente enunciado en la definición de clase de Estudiante:

rename nss as id_estudiante

Para redefinir un método, primero cambiamos su nombre y luego incluimos una definición del método con el nuevo nombre.

O2 permite también la **herencia múltiple**, donde una clase hereda el tipo y los métodos de dos o más clases. Si las dos clases tienen un atributo o un método con el mismo nombre, lo mejor será cambiar el nombre de los dos para que se puedan distinguir en la subclase. En el enunciado de cambio de nombre podemos calificar el nombre de atributo o de método con el nombre de la clase anexándole *_nombre-de-clase*, obviando así el problema de ambigüedad de nombres que puede surgir con la herencia múltiple.

Los vínculos clase/subclase y la herencia se especifican en O2 usando la palabra reservada **inherit** (heredar) en la declaración de subclase. Por ejemplo, en la figura 22.5, las clases Profesorado y Estudiante heredan el tipo y los métodos de la clase Persona, y cada una incluye sus propios atributos y métodos adicionales. La clase Profesorado tiene los siguientes atributos adicionales: rango, salario, oficina y teléfono. Además, incluimos en la clase Profesorado tres atributos con valor de conjunto —perteneca_a, subvenciones y asesora— para representar los vínculos PERTENECE, IP y ASESOR, respectivamente.

Manipulación de datos en O2. Las aplicaciones de O2 pueden crearse de dos maneras con lenguajes de programación. La primera es usar los lenguajes de consulta y de programación propios de O2, O2SQL y O2C, respectivamente, para escribir programas de aplicación. El segundo es usar O2 como un sistema de almacenamiento de objetos persistentes para otro lenguaje autónomo, como C++ y crear la aplicación en ese lenguaje. O2 tiene además un generador de interfaces con el usuario llamado O2Look, que puede servir para acelerar la creación de aplicaciones facilitando la interacción con O2C a través de una interfaz amable con el usuario. Hay otra herramienta, llamada O2Tools, que es un entorno gráfico de creación de programas que cuenta con herramientas como un examinador, un depurador, un *shell* que permite editar y ejecutar órdenes de O2 e interactuar con archivos Unix, y un administrador de espacio de trabajo que puede almacenar temporalmente objetos de base de datos.

Primero ilustraremos el lenguaje O2C y la forma de usarlo para escribir métodos para las clases. Este lenguaje es un subconjunto del lenguaje C que se ha extendido para manejar tipos de O2, con operaciones para tipos de conjuntos y de listas. La figura 22.6(a) muestra la definición de unos cuantos de los métodos que se declararon en las clases O2 de la figura 22.5. El primer método, *edad*, calcula la edad de una persona a partir de su fecha de nacimiento y la fecha actual, utilizando exclusivamente elementos del lenguaje C. Cabe señalar que el empleo de la palabra *self* (mismo) dentro de un método se refiere al objeto para el cual

se invocó el método. Así, la palabra *self* en el método *edad* se refiere a un objeto de tipo *Persona* y tiene el tipo y métodos de uno de esos objetos. El segundo método, *promedio_notas*, usa un ciclo *for* que se repite sobre el conjunto de valores del atributo *boleta* de *Estudiante* para obtener la suma de las notas numéricas y el número de registros boleta; en seguida, calcula y devuelve el promedio de notas. El tercer método es un ejemplo de actualización, y cambia la carrera de un estudiante. Este método también invoca dos métodos de la clase *Departamento*—*quitar_de_carrera* y *añadir_a_carrera*— que también aparecen en la figura 22.6(a). Esto lo hace para asegurarse de que los valores del atributo inverso—alumnos de la clase *Departamento*—sigan siendo congruentes con el valor del atributo *carrera*, en de la clase *Estudiante*. Esto ofrece un ejemplo de la manera como el programador puede implementar vínculos bidireccionales definiendo métodos O2 apropiados.

Hay dos formas de crear un objeto persistente en O2. Una de ellas es hacer que el objeto mismo sea una raíz, dándole un nombre con el enunciado **name**. En este caso, el objeto se denomina *raíz persistente*. La otra forma es hacer que el objeto sea *alcanzable* desde una raíz persistente; por ejemplo, haciéndolo miembro de un objeto persistente con valor de conjunto o convirtiéndolo en un componente de un objeto complejo. En general, O2 no crea un conjunto de objetos persistente que corresponda a cada declaración de clase; en vez de ello, deja esta tarea para que el usuario la efectúe explícitamente. Así pues, si se necesita un conjunto de todos los objetos persistentes de tipo *Persona*, el programador deberá crear una raíz persistente de tipo *set(Persona)* y darle un nombre, como se muestra en la figura 22.6(b), donde al conjunto se le llama *Todas personas*. Cualquier objeto que se añada a ese conjunto se hará persistente automáticamente, pues se *conecta a una raíz persistente*. El objeto *Persona* “Federico Vizcarra” que se muestra en la figura 22.6(b) no es persistente cuando se crea, pero se vuelve persistente cuando es añadido al objeto *Todas personas*. También se puede crear un objeto persistente individual con sólo darle un nombre—sin que tenga que pertenecer a ningún conjunto persistente— como se ilustra con el objeto *Persona* llamado “José Silva” en la figura 22.6(b).

O2 también tiene un lenguaje de consulta, O2SQL, que puede servir para obtener un conjunto de valores y colocarlo en una clase transitoria en un programa escrito en O2C. O2SQL permite crear un nuevo tipo *sobre la marcha* para el resultado de una consulta. El lenguaje de consulta cuenta con una estructura **SELECT ... FROM ... WHERE** similar a la de SQL, pero en O2SQL es posible usar referencia funcional dentro de la consulta para referirse a los componentes de objetos complejos y a los valores. La referencia funcional usa la *notación de punto* para referirse a los componentes de objetos complejos; por ejemplo, al escribir *e.carrera_en.nombre*

en C1 de la figura 22-7, primero localizamos el objeto *DEPARTAMENTO* relacionado con el *ESTUDIANTE* *e* a través de la función *carrera_en*, y luego localizamos el atributo *nombre* de ese departamento.

En una consulta O2SQL, la cláusula **SELECT** define los datos que se desea obtener y la estructura (tipo) de los valores en el resultado de la consulta. La cláusula **FROM** especifica las colecciones de objetos a las que la consulta hace referencia y proporciona nombres de variables que abarcan los objetos de esas colecciones. Por lo regular, una *colección* es un conjunto de objetos. En la figura 22.7 supusimos que *Estudiante* es el nombre de una colección de todos los objetos *Estudiante* persistentes. Por último, la cláusula **WHERE** especifica las condiciones para seleccionar los objetos individuales que intervienen en el resultado de la consulta. La figura 22.7 muestra ejemplos de dos consultas. La consulta C1

```
(a)
method body edad: integer in class Persona
{
    int a;
    Fecha f;
    f = today();
    a = f->año - self->fechanac->año;
    if ( f->mes < self->fechanac->mes ) ||
        ( f->mes == self->fechanac->mes ) && ( f->día < self->fechanac->día )
        --a; /* decrementa a en 1 */
    return a;
}

method body promedio_notas: real in class Estudiante
{
    float suma = 0.0;
    int cuenta = 0;
    struct {
        char not;
        float mnot;
        o2_Sección sec;
    } t;
    for ( t in self->boleta ) {
        suma += t->mnot; ++cuenta; /* incrementa suma en mnot, cuenta en 1 */
    }
    return suma/cuenta;
}

method body cambiar_carrera (nueva_carrera: Departamento): boolean in class Estudiante
{
    if (self->carrera_en->quitar_de_carrera(self) ) {
        return 0;
    }
    else {
        nueva_carrera->añadir_a_carrera(self);
        self->carrera_en = nueva_carrera;
        return 1;
    }
}

method body quitar_de_carrera (e: Estudiante): boolean in class Departamento
{
    if ( e in self->alumnos ) {
        self->alumnos -= set(e); /* = aplica diferencia de conjuntos para quitar el objeto e del conjunto
        de alumnos */
        return 0;
    }
    else return 1;
}

method body añadir_a_carrera (e: Estudiante) in class Departamento
{
    self->alumnos += set(e); /* += aplica la unión de conjuntos para añadir el objeto e al conjunto de alumnos */
}
```

Figura 22.6 Programación en O2. (a) Declaraciones de cuerpos de métodos en O2 para algunos de los métodos incluidos en la figura 22.5. (b) Creación de objetos persistentes en O2. (continúa en la siguiente página)

```
(b)
name Todas_Personas: set (Persona) /* una raíz persistente para contener todos los objetos Persona
persistentes */

name José_Silva: Persona; /* una raíz persistente para contener un solo objeto Persona */

run body {
    o2 Persona p = new Persona; /* crea un nuevo objeto Persona p */
    'p' = tuple (nss: "333445555",
        nombre: tuple (nombrepila: "Federico", paternio: "Vizcarra", maternio: "Torres"),
        dirección: tuple (número: 638, calle: "Vale", ciudad: "Higueras",
            estado: "México", códigopostal: "77079"),
        fechanac: tuple (año: 1945, mes: 12, día: 8),
        sexo: M);
    Todas_Personas += set(p); /* p se vuelve persistente al anexarse a una raíz persistente */

    /* ahora pondremos valores en el objeto persistente con nombre José_Silva */
    José_Silva->nss = "123456789",
    José_Silva->nombre: tuple(nombrepila: "José", paternio: "Silva", maternio: "Barragán"),
    José_Silva->dirección: tuple(número: 731, calle: "Fresnos", ciudad: "Higueras",
        estado: "México", códigopostal: "77036"),
    José_Silva->fechanac: tuple(año: 1955, mes: 1, día: 9),
    José_Silva->sexo: M;
}
```

Figura 22.6 (continuación)

```
C1: select tuple ( nombre: e.nombre.nombrepila,
    apellido: e.nombre.paternio )
from e in Estudiante
where e.carrera_en.nombre = "Ciencias de la computación"

C2: select tuple ( nombre: e.nombre.nombrepila,
    apellido: e.nombre.paternio )
boleta: select tuple (
    nombre: sc.sección.curso.nombre,
    núm_sec: sc.sección.núm_sec,
    trimestre: sc.sección.trím,
    año: sc.sección.año,
    notas: sc.notas )
from sc in sc
from e in Estudiante, sec in e.boleta
where e.carrera_en.nombre = "Ciencias de la computación"
```

Figura 22.7 Dos consultas en O2SQL.

obtiene los nombres de todos los estudiantes cuya carrera es ciencias de la computación, y el tipo del resultado de la consulta es un conjunto de tuplas con dos atributos: nombre y apellido. La consulta C2 obtiene las boletas de los estudiantes cuya carrera es ciencias de la computación, y el resultado tiene un tipo complejo cuya estructura se especifica en la cláusula SELECT de la consulta. El recurso para definir una nueva estructura de tipo para los valores del resultado de una consulta no está disponible en todos los SGBDOO. Algunos sistemas sólo permiten obtener objetos completos de una colección tal cual, sin reestructurarlos para el resultado de la consulta. El programador puede entonces reestructurar explícitamente estos objetos como objetos transitorios de nuevos tipos que se declaran en el programa usuario.

```
class Persona: o2_root {
public:
    char* nss;
    struct {
        char* nombrepila;
        char* paternio;
        char* maternio } nombre;
    struct {
        int número;
        char* calle;
        char* núm_apto;
        char* ciudad;
        char* estado;
        char* códigopostal } dirección;
    struct {
        int año;
        int mes;
        int día } fechanac;
        char sexo;
        int edad();
    }
}
```

Figura 22.8 Declaración de clase en C++, correspondiente a la clase O2 Persona.

Una alternativa al lenguaje O2C es usar un lenguaje autónomo, como C++, junto con el sistema O2. Para facilitar esto, O2 provee un **recurso de exportación** que crea clases C++ correspondientes a las declaraciones de clase en O2. La figura 22.8 muestra una clase C++ que corresponde a la clase Persona declarada en la figura 22.5. Ahora podemos obtener los objetos de la clase O2 persistente Persona y colocarlos directamente en variables definidas del tipo de clase C++ correspondiente; estas variables pueden manipularse con código de programa en C++.

Panorama sobre la arquitectura del sistema O2. En esta sección presentaremos un breve bosquejo de la arquitectura del sistema O2. Una parte de este sistema, llamada O2Engine, se encarga de una buena parte de la funcionalidad del SGBD, como proporcionar recursos de almacenamiento, obtención y actualización de objetos almacenados persistentemente que pueden ser compartidos por múltiples programas. O2Engine pone en práctica los mecanismos de control de concurrencia, recuperación y seguridad comunes en los sistemas de bases de datos. Además, O2Engine implementa un modelo de gestión de transacciones y mecanismos de evolución de esquemas.

La implementación de O2Engine en el nivel de sistema se basó en una *arquitectura cliente/servidor*, contemplando la tendencia actual hacia los sistemas de cómputo en redes y distribuidos (véase el Cap. 23). El *componente servidor*, que puede ser una máquina servidora de archivos, es un *servidor de páginas*; sólo se ocupa del almacenamiento en el nivel de página (bloque de disco) y no conoce las estructuras de los objetos. Su obligación es obtener páginas de manera eficiente cuando un cliente le ordena que lo haga, y mantener el control de concurrencia y la información de recuperación apropiados en el nivel de página. En O2, el control de concurrencia se basa en bloqueo, y la recuperación en una técnica de escritura anticipada en la bitácora. El servidor también se ocupa de parte del almacenamiento de

páginas en *cache* para reducir la E/S de disco, y el acceso a él es a través de una interfaz de llamada a procedimiento remoto (RPC: *remote procedure call*) desde los clientes. Un cliente es por lo regular una estación de trabajo, y la mayor parte de la funcionalidad de O2 se provee en el nivel de cliente.

En el nivel funcional, O2Engine tiene tres componentes principales. El *componente de almacenamiento*, en el nivel más bajo, es una extensión de un sistema de almacenamiento llamado *wiss* (*Wisconsin Storage System*), que fue creado en la University of Wisconsin. La implementación de este nivel está dividida entre el cliente y el servidor. El proceso servidor se encarga de manejar el disco, almacenar y leer páginas, controlar la concurrencia y recuperarse. El proceso cliente almacena en *cache* páginas y candados provistos por el servidor y los pone a disposición de los módulos funcionales de mayor nivel del cliente O2.

El siguiente componente funcional, llamado *gestor de objetos*, se encarga de estructurar objetos y valores, formar grupos de objetos relacionados en páginas de disco, indizar objetos, mantener la identidad de los objetos, efectuar operaciones con objetos, etc. Los identificadores de objetos se implementaron en O2 como la dirección física en disco de los objetos, para evitar el gasto extra de la correspondencia lógica-a-física de identificadores, y se basan en los identificadores de registro *wiss*. El OID incluye un identificador de volumen de disco, un número de página dentro del volumen y un número de ranura dentro de la página. Los objetos complejos estructurados se descomponen en registros y se usan índices para tener acceso a los componentes con estructura de conjunto o de lista de un objeto.

El nivel funcional más alto de O2Engine se denomina *gestor de esquemas*. Lleva el control de las definiciones de clases, tipos y métodos; provee los mecanismos de herencia; verifica la consistencia de las declaraciones de clases, y hace posible la evolución de esquemas, lo que incluye la creación y la eliminación incrementales de declaraciones de clases. Para el lector interesado, al final de este capítulo se dan referencias bibliográficas que analizan diversos aspectos del sistema O2.

22.7.2 Panorama sobre el sistema ObjectStore

En esta sección presentaremos un bosquejo del *SOBDOO* ObjectStore. Primero ilustraremos la definición de datos en ObjectStore y luego daremos ejemplos de consultas y de manipulación de datos.

Definición de datos en ObjectStore. El sistema ObjectStore está íntimamente integrado al lenguaje C++ y proporciona capacidades de almacenamiento persistente de objetos C++. Se tomó esta decisión para evitar el problema de *diferencia de impedancia* entre un sistema de bases de datos y su lenguaje de programación, donde las estructuras provistas por el sistema son diferentes de las que provee el lenguaje de programación. Así pues, ObjectStore puede usar las declaraciones de clase de C++ como su lenguaje de definición de datos. Utiliza un C++ extendido que incluye elementos adicionales que son útiles específicamente en aplicaciones de bases de datos. Los objetos de una clase pueden ser transitorios en un programa o bien pueden almacenarse persistentemente con ObjectStore. Los objetos persistentes pueden ser compartidos por varios programas. Un apuntador a un objeto tiene la misma sintaxis ya sea que el objeto sea persistente o transitorio, así que la persistencia es más o menos transparente para los programadores y usuarios.

La figura 22.9 muestra declaraciones posibles de clase C++ de ObjectStore para una porción de la base de datos UNIVERSIDAD, cuyo esquema EER se dio en la figura 21.10. El compilador de C++ extendido de ObjectStore maneja declaraciones de vínculos inversos y

funciones adicionales.[†] En C++, un asterisco (*) especifica una referencia (apuntador), y el tipo de un campo (atributo) aparece antes de su nombre. Por ejemplo, la declaración

Profesorado *asesor

en la clase *Estudiante_posgrad* especifica que el atributo *asesor* tiene el tipo *apuntador a un objeto Profesorado*. Los tipos básicos de C++ son carácter (*char*), entero (*int*) y número real (*float*). Se puede declarar que una cadena de caracteres sea de tipo *char** (apuntador a un arreglo de caracteres).

En C++, una clase *derivada* E' hereda la descripción de una clase *base* E al incluir el nombre de E en la definición de E' después de un signo de dos puntos (;) y de la palabra reservada **public** o bien la palabra reservada **private**.[‡] Por ejemplo, en la figura 22.9 tanto la clase *Profesorado* como la clase *Estudiante* se derivan de la clase *Persona*, y ambas heredan los campos (atributos) y las funciones (métodos) declarados en la descripción de *Persona*. Las funciones se distinguen de los atributos al incluir parámetros entre paréntesis después del nombre de la función. Si una función no tiene parámetros, sólo incluimos los paréntesis (). Una función que no devuelve valor alguno es de tipo *void*. Existe una correspondencia directa entre las declaraciones de O2 de la figura 22.5 y las declaraciones de C++ de la figura 22.9, como puede verse si se les compara. ObjectStore añade su propio **constructor de conjuntos** a C++ valiéndose de la palabra reservada *os_Set* (que significa conjunto de ObjectStore). Por ejemplo, la declaración

os_Set<Boleta> boleta

dentro de la clase *Estudiante* especifica que el valor del atributo *boleta* en cada objeto *estudiante* es un *conjunto de apuntadores* a objetos de tipo *Boleta*. El constructor de tuplas está implícito en las declaraciones de C++ siempre que diversos atributos se declaren en una clase (o un *struct*, que es el concepto de C++ que corresponde a una tupla). ObjectStore ha añadido además constructores de bolsas y de listas a C++, llamados *os_Bag* y *os_List*, respectivamente.

Las declaraciones de clases de la figura 22.9 incluyen atributos de referencia en ambas direcciones para los vínculos de la figura 21.10. Por ejemplo, para el vínculo *CARRERAS* incluimos el atributo *carrera_en* —con tipo apuntador a Departamento— en la clase *Estudiante* y el atributo *alumnos* —con tipo conjunto de apuntadores a Estudiante— en la clase *Departamento*. ObjectStore contiene un **recurso de vínculos** que actúa como extensión de las declaraciones C++ y permite especificar atributos inversos que representan un vínculo binario. La figura 22.10 ilustra la sintaxis de este recurso, que se especifica añadiendo la palabra reservada *inverse_member* (miembro inverso) a un atributo, seguida de una lista de sus atributos inversos en la otra clase. Por ejemplo, en la figura 22.10 se declaran de este modo los vínculos *PRESENTE* y *PERTENECE* entre Profesorado y Departamento. El atributo *pertenece_a* de la clase *Profesorado* se declara como inverso del atributo *miembros* de la clase *Departamento*. Así, ObjectStore mantiene la consistencia en el vínculo de manera automática; si se añade una referencia a un objeto Profesorado al conjunto *miembros* de un determinado objeto Departamento, el sistema añadirá automáticamente una referencia al Departamento en el conjunto *pertenece_a* de ese objeto Profesorado. Las referencias en la

[†]ObjectStore puede usarse también con otros compiladores de C++; para ello se usa la interfaz de biblioteca de C++ de ObjectStore.

[‡]C++ tiene dos tipos de derivaciones: públicas y privadas. Aquí sólo consideraremos las derivaciones públicas.

```

/* éste es el archivo esquema_univ.H que incluye las declaraciones de clases de la base de datos */

struct Teléfono {
    int cód_área;
    int número;
}

struct Fecha {
    int año;
    int mes;
    int día;
}

class Persona {
public:
    char nss[9];
    struct {
        char* nombrepila;
        char* paterno;
        char* materno;
    } nombre;
    struct {
        int número;
        char* calle;
        char* núm_apto;
        char* ciudad;
        char* estado;
        char* códigopostal;
    } dirección;
    Fecha fechanac;
    char sexo;

    int edad();
}

struct Boleta {
    char notas;
    float nnotas;
    Sección *sección;
}

class Estudiante: public Persona {
/* Estudiante hereda (se deriva de) Persona */ public:
    char* clase;
    Departamento
        *carrera_en;
    Departamento
        *especialidad_en;
    os_Set<Sección*>
        inscrito_en;
    os_Set<Boleta*>
        boleta;

    float promedio_notas();
    void cambiar_clase();
    void cambiar_carrera (Departamento *nueva_carrera);
}

```

Figura 22.9 Algunas declaraciones de clases C++ para una base de datos
ObjectStore UNIVERSIDAD. (continúa en la siguiente página)

```

struct Grado {
    char* colegio;
    char* grado;
    int año;
}

class Estudiante_posgrad: public Estudiante {
/* Estudiante_posgrad hereda (se deriva de) Estudiante */ public:
    os_Set<Grado*> grados;
    Profesorado *asesor;
}

class Profesorado: public Persona { /* Profesorado hereda (se deriva de) Persona */
    float salario;
    char* rango;
    char* oficinap;
    Teléfono telefonop;
    os_Set<Departamento*> pertenece_a;
    Departamento *preside;
    os_Set<Subvención*> subvenciones;
    os_Set<Estudiante*> asesora;

    void promover_profesor();
    void dar_aumento (float porcentaje);
}

class Departamento {
    char* nombre;
    char* oficina;
    Teléfono telefonod;
    os_Set<Profesorado*> miembros;
    os_Set<Estudiante*> alumnos;
    Profesorado director;
    os_Set<Curso*> cursos;

    void añadir_profesor (Profesorado *p);
    void añadir_a_carrera (Estudiante *e);
    void quitar_de_carrera (Estudiante *e);
}

```

Figura 22.9 (continuación)

dirección 1:1 o N:1 (cuya participación *máx* es 1) se declaran como apuntadores individuales, en tanto que las referencias en las direcciones 1:N o M:N (cuya participación *máx* es mayor que 1) se declaran empleando la construcción `os_Set` (conjunto de apuntadores).

La figura 22.10 ilustra también otra característica de C++: la función **constructor** de una clase. Una clase puede tener una función con el mismo nombre que la clase y que sirve para crear *objetos nuevos* de esa clase. En la figura 22.10, el constructor de *Profesorado* suministra únicamente el valor *nss* de un objeto *Profesorado* (*nss* se *hereda* de *Persona*), y el constructor de *Departamento* sólo suministra el valor *nombre*. Los valores de los demás atributos se pueden agregar posteriormente a los objetos, aunque en un sistema real la función constructora incluiría más parámetros para construir un objeto más complejo. A continuación explicaremos cómo se pueden usar constructores para crear objetos persistentes.


```

extern database *bd_univ;

class Persona;

class Profesorado: public Persona { /* Profesorado hereda (se deriva de) Persona */
float salario;
char * rango;
char * oficina;
Teléfono teléfono;
os_Set<Departamento*> pertenece_a;
inverse_member Departamento:miembros;
Departamento *preside;
inverse_member Departamento:director;
os_Set<Subvención*> subvenciones;
os_Set<Estudiante*> asesora;

Profesorado(char s[9]) {nss = new(db_univ) char[9], strcpy(nss, s);
/* el atributo nss se hereda de Persona */
void promover_profesor();
void dar_aumento(float porcentaje);
}

class Departamento {
char * nombre;
char * oficina;
Teléfono teléfono;
os_Set<Profesorado*> miembros;
inverse_member Profesorado:pertenece_a;
os_Set<Estudiante*> alumnos;
Profesorado *director;
inverse_member Profesorado:preside;
os_Set<Curso*> cursos;

Departamento(char d) {nombre = new(bd_univ) char[strlen(d)+1]; strcpy(nombre, d);
void añadir_profesor(Profesorado *p);
void añadir_a_carrera(Estudiante *e);
int quitar_de_carrera(Estudiante *e);
}

```

Figura 22.10 Declaración de vínculos inversos en ObjectStore.

Manipulación de datos en ObjectStore. ObjectStore utiliza C++ para crear sus aplicaciones, y además cuenta con una herramienta de interfaz gráfica con el usuario para facilitar la creación de aplicaciones. Se puede aplicar funciones adicionales a los tipos de colección (`os_Set`, `os_Bag` y `os_List`); entre ellas están las funciones **insert(e)**, **remove(e)** y **create**, que sirven para insertar un elemento *e* en una colección, quitar un elemento *e* de una colección y crear una colección nueva, respectivamente. Además, una construcción de programación **foreach(c, colección)** permite al programa ejecutar ciclos sobre cada elemento *c* de una colección. Estas funciones se ilustran en la figura 22.11(a), que muestra cómo pueden especificarse en ObjectStore algunos de los métodos declarados en la figura 22.9. La función **añadir_a_carrera** añade un (apuntador a un) estudiante al atributo de conjunto **alumnos** de la clase **Departamento**, al invocar la función de inserción con el enunciado **alumnos->insert**. De manera similar, la función **quitar_de_carrera** elimina un apuntador a estudiante del mismo conjunto. Aquí suponemos que se hicieron las declaraciones **inverse_member**

```

(a)
int Persona::edad() {
Fecha f = today(); int a = f.año - fechanac.año;
if (f.mes < fechanac.mes) //
((f.mes == fechanac.mes) && (f.día < fechanac.día)) -- a;
return a;
};

float Estudiante::promedio_notas() {
float suma = 0.0; int cuenta = 0; Boleta *b;
foreach(b, boleta) {
suma += b->notas; ++cuenta; /* incrementa suma en notas, cuenta en 1 */
}
return suma/cuenta;
};

void Estudiante::cambiar_carrera(Departamento *nueva_carrera) {carrera_en = nueva_carrera; }
void Departamento::quitar_de_carrera(Estudiante *e) {alumnos->remove(e); }
void Departamento::añadir_profesor(Profesorado *p) {miembros->insert(p); }
void Departamento::añadir_a_carrera(Estudiante *e) {alumnos->insert(e); }

(b)
main() {
database *bd_univ = database::open ("basesdis/univ");
transaction::begin();
/* crear dos conjuntos persistentes para contener todos los objetos Profesorado y Departamento */
os_Set<Profesorado*> &todo_profesorado = os_Set<Profesorado*>::create(bd_univ);
os_Set<Departamento*> &todos_deptos = os_Set<Departamento*>::create(bd_univ);

/* crear un nuevo objeto Profesorado y un nuevo objeto Departamento empleando los constructores */
Profesorado *p = new (bd_univ) Profesorado ("123456789");
Departamento *d = new (bd_univ) Departamento ("Ciencias de la computación");
/* relacionar los objetos invocando el método apropiado */
d->añadir_profesor(p);
/* añadir los objetos a sus conjuntos persistentes */
todo_profesorado.insert(p); todos_deptos.insert(d);
transaction::commit();
}

```

Figura 22.11 Programación en ObjectStore. (a) Definiciones de funciones en C++ para algunos de los métodos incluidos en la figura 22.9. (b) Creación de objetos persistentes en ObjectStore.

apropiadas, de modo que el sistema da mantenimiento automático a cualesquier atributos inversos. En la función **promedio_notas**, se usa la construcción **foreach(b, boleta)** para recorrer el conjunto de registros **boleta** dentro de un objeto **Estudiante** y así calcular su promedio de notas de aprovechamiento.

En C++, la referencia funcional a los elementos de un objeto *o* usa la notación **de flecha** cuando se provee un *apuntador* a *o*, y usa la notación **de punto** cuando se provee una variable cuyo valor es el objeto *o* mismo. Con cualquiera de ellas podemos referirnos tanto a los atributos como a las funciones de un objeto. Por ejemplo, las referencias **faño** y **fb->notas** en

```

os_Set<Profesorado*> &profs_asist =
    todo_profesorado [: rango == 'Profesor asistente' : ]

os_Set<Profesorado*> &profs_asist_ricos =
    todo_profesorado [: rango == 'Profesor asistente' && salario > 5000.00 : ]

os_Set<Profesorado*> &presiden_deptos =
    todo_profesorado [: preside : ]

os_Set<Profesorado*> &profs_co =
    todo_profesorado [: pertenece_a [: nombred == 'Ciencias de la computación' : ] : ]

```

Figura 22.12 Consultas en ObjectStore.

las funciones *edad* y *promedio_notas* se refieren a atributos componentes, en tanto que la referencia a *alumnos->remove* en *quitar_de_carrera* invoca la función *remove* de *ObjectStore* sobre el conjunto *alumnos*.

Si el programador o usuario desea crear objetos y colecciones persistentes en *ObjectStore* deberá asignarles un **nombred**, al que también se le denomina *variable persistente*. La variable persistente puede considerarse como una referencia abreviada al objeto, y *ObjectStore* la “recuerda” permanentemente. Por ejemplo, en la figura 22.11 (b) creamos dos objetos persistentes con valor de conjuntos—*todo_profesorado* y *todos_deptos*—y los hicimos persistentes en la base de datos llamada *bd_univ*. La aplicación utiliza estos objetos para contener apuntadores a todos los objetos persistentes de tipo *profesorado* y departamento, respectivamente. Podemos crear un objeto miembro de una clase definida invocando la función constructora de objetos de esa clase, con la palabra reservada **new**. Por ejemplo, en la figura 22.11 (b) creamos un objeto *Profesorado* y un objeto *Departamento*, y luego los relacionamos invocando el método *añadir_profesor*. Por último, los agregamos a los conjuntos *todo_profesorado* y *todos_deptos* para hacerlos persistentes.

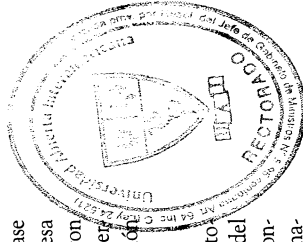
ObjectStore cuenta también con un recurso de consultas, que sirve para seleccionar un conjunto de objetos de una colección especificando una condición de selección. El resultado de una consulta es un conjunto de apuntadores a los objetos que satisfacen la consulta. Los enunciados del lenguaje de consulta pueden incorporarse en un programa en C++, y considerarse como un medio de acceso asociativo de alto nivel a los objetos seleccionados que hace innecesario crear una construcción cíclica explícita. La figura 22.12 ilustra unas cuantas consultas, cada una de las cuales devuelve un subconjunto de objetos de la colección *todo_profesorado* que satisfacen una condición determinada. La primera consulta de la figura 22.12 selecciona todos los objetos *Profesorado* de la colección *todo_profesorado* cuyo rango es *Profesor asistente*; la segunda consulta obtiene los profesores asistentes cuyo salario es mayor que \$5000.00; la tercera obtiene los profesores que presiden departamentos, y la cuarta los profesores de ciencias de la computación.

ObjectStore también permite al usuario especificar la creación de índices, con los cuales el sistema puede optimizar la obtención de objetos que satisfacen un criterio de selección. Otra característica útil es que maneja múltiples versiones de un objeto que pueden existir de manera concurrente.

El análisis anterior ofreció un somero bosquejo de algunas características de *ObjectStore*. En la bibliografía encontrará el lector referencias a documentos que describen *ObjectStore* con mayor detalle.

22.8 Diseño de bases de datos OO por transformación EER-OO*

Resulta sencillo diseñar las declaraciones de tipos de las clases de objetos para un SOBDOO a partir de un esquema EER que no contiene categorías ni vínculos *n*-arios con *n* mayor que 2. Por otro lado, como los métodos no se especifican en el diagrama EER, se deben añadir a las declaraciones de clases después de completar la transformación estructural. Toda clase debe incluir un método constructor y uno destructor para crear y destruir objetos de esa clase, además de cualesquier métodos adicionales que requiera la aplicación. Si se diseñaron conceptualmente las transacciones como se explicó en la sección 21.5, éstas podrán servir como base para definir e implementar los métodos. A grandes rasgos, la transformación de EER a OO es como sigue:



PASO 1: Crear una clase OO para cada clase EER. El tipo de la clase OO deberá incluir todos los atributos de la clase EER mediante un constructor de tupla en el nivel superior del tipo. Los *atributos multivaluados* se declaran a través de los constructores de bolsa, de conjunto o de lista.[†] Si los valores del atributo multivaluado de un objeto deben estar ordenados, se elegirá el constructor de lista; si se permiten duplicados, deberá escogerse el constructor de bolsa. Los *atributos compuestos* se declaran con un constructor de tupla. ■

PASO 2: Añadir atributos de referencia para cada vínculo binario a las clases OO que participen en el vínculo. Los atributos pueden crearse en una dirección 1:1 o N:1 y con valor de conjunto o con valuados para los vínculos binarios en la dirección 1:1 o N:N. Si un vínculo binario se representa valor de lista para los vínculos en la dirección 1:N o M:N. Si un vínculo binario se representa con referencias en ambas direcciones, se debe declarar que cada referencia es el inverso de la otra, si existe el recurso para hacerlo. Si hay atributos de vínculo, puede usarse un constructor de tupla para crear una estructura de la forma <referencia, atributos de vínculo>, la cual se incluye en lugar del atributo de referencia. ■

PASO 3: Incluir métodos apropiados para cada clase. Éstos no están disponibles en el esquema EER y se deberán agregar al diseño de la base de datos según se necesiten. Todo método constructor deberá incluir código que verifique cualesquier restricciones que se deben cumplir al crearse un nuevo objeto. Todo método destructor deberá verificar cualesquier restricciones que pudieran violarse al eliminarse el objeto. Los otros métodos deberán incluir cualesquier verificaciones adicionales de restricciones que sean pertinentes. ■

PASO 4: Una clase OO que corresponda a una subclase en el esquema EER heredará el tipo y los métodos de su o sus superclases en el esquema OO. Sus atributos y referencias específicas se declaran como se explicó en los pasos 1 y 2. ■

PASO 5: Los tipos de entidades débiles que no participan en ningún otro vínculo además de su vínculo de identificación se pueden transformar como si fueran atributos multivaluados compuestos del tipo de entidades propietario, empleando el constructor *set(tuple(...))*. ■

PASO 6: Los vínculos *n*-arios con *n* > 2 pueden transformarse a un tipo de objetos aparte, con referencias apropiadas a cada uno de los tipos de objetos participantes. Estas referencias se

[†]Debemos decidir si se usa un constructor de conjuntos o un constructor de listas, porque esta información no está disponible en el esquema EER.

basan en hacer corresponder un vínculo 1:N de cada uno de los tipos de entidades participantes con el vínculo n-ario. También puede utilizarse esta opción con los vínculos binarios M:N, si así se desea. ■

Estos pasos generales de transformación se pueden aplicar tanto a O2 como a ObjectStore y a otros modelos de datos y sistemas OO. Consideremos las declaraciones O2 de la figura 22.5, que muestran algunas de las declaraciones de tipos y clases O2 para la base de datos UNIVERSIDAD de la figura 21.10. Incluimos atributos de referencia en ambas direcciones para la mayoría de los vínculos. Por ejemplo, para el vínculo CARRERAS incluimos el atributo carrera, en —con tipo Departamento— en la clase ESTUDIANTE y el atributo alumnos —con tipo set(Estudiente)— en la clase DEPARTAMENTO. Sin embargo, en el vínculo ESPECIALIDADES sólo incluimos el atributo de referencia especialidad, en la clase ESTUDIANTE. La decisión de representar un vínculo binario sólo en una dirección o en ambas se deja al diseñador de la base de datos OO, según el uso que se piense dar a los objetos. Por ejemplo, si se espera usar principalmente el vínculo ESPECIALIDADES para referirse al departamento de especialidad de un objeto estudiante, y casi nunca a referirse al conjunto de todos los estudiantes que cursan su especialidad en un departamento dado, la elección que se hizo aquí sería la adecuada.

Cabe señalar que O2 se puede caracterizar como un tanto débil en su manejo de los vínculos en general, ya que las declaraciones de clase no indican si dos atributos de dos clases están representando el mismo vínculo. ObjectStore permite declarar dos atributos como inversos entre sí, de modo que el sistema mismo pueda mantener la consistencia del vínculo, como se ilustra en la figura 22.10. En O2 y otros sistemas similares, los programadores deben mantener todos los vínculos explícitamente, para lo cual han de codificar los métodos que actualizan los objetos en la forma apropiada. Así pues, en cierto sentido, estos sistemas no reconocen el concepto de vínculo bidireccional como tal; en vez de ello, dejan que los programadores implementen dichos vínculos.

22.9 Resumen

En este capítulo estudiamos el enfoque orientado a objetos de los sistemas de bases de datos, propuesto para poder satisfacer las necesidades de aplicaciones complejas de bases de datos y para añadir funcionalidad de bases de datos a los lenguajes de programación orientados a objetos como C++. En primer término analizamos los principales conceptos que se utilizan en las bases de datos OO, entre ellos los siguientes:

- **Identidad de objetos:** Los objetos tienen identidades únicas, independientes de los valores de sus atributos.
- **Constructores de tipos:** Las estructuras de objetos complejos pueden construirse aplicando recursivamente un conjunto de constructores básicos, como los de tupla, conjunto, lista y bolsa.
- **Encapsulamiento:** Tanto la estructura de los objetos como las operaciones que se pueden aplicar a ellos se incluyen en las definiciones de clase de los objetos.
- **Compatibilidad con los lenguajes de programación:** Tanto los objetos persistentes como los transitorios se manejan de manera uniforme. Para que los objetos sean persistentes se les anexa a una colección persistente.

- **Jerarquías de tipos y herencia:** Es posible especificar los tipos de los objetos mediante una jerarquía de tipos, que permite heredar tanto los atributos como los métodos de tipos previamente definidos.
- **Manejo de objetos complejos:** Es posible almacenar y manipular objetos complejos tanto estructurados como no estructurados.
- **Polimorfismo y sobrecarga de operadores:** Los operadores y los nombres de métodos se pueden “sobrecargar” de modo que se apliquen a diferentes tipos de objetos con distintas implementaciones.
- **Creación de versiones:** En algunos sistemas OO es posible mantener varias versiones del mismo objeto.

Después presentamos un panorama de dos SCBDOO —a saber, O2 y ObjectStore— y bosquejamos un procedimiento de transformación que sirve para diseñar un esquema de base de datos a partir de un diseño conceptual EER. Los investigadores han debatido sobre la conveniencia de ensayar un enfoque totalmente nuevo de la gestión de bases de datos —el enfoque orientado a objetos—, en vez de extender las capacidades de los sistemas de gestión relacionales que hay en el mercado, de modo que satisfagan las necesidades de aplicaciones de bases de datos más complejas. Actualmente se trabaja en una norma SQL3 que incorporará conceptos de OO en la norma de lenguaje relacional SQL2. Entre los prototipos relacionales extendidos están los proyectos POSTGRES y STARBURST (véase la bibliografía selecta).

Preguntas de repaso

- 22.1. ¿Qué orígenes tiene el enfoque orientado a objetos?
- 22.2. ¿Qué características primarias debe poseer un OID?
- 22.3. Analice los diversos constructores de tipos. ¿Cómo se les usa para crear estructuras de objetos complejos?
- 22.4. ¿Qué diferencia hay entre clase y tipo en la terminología OO? Analice el concepto de encapsulamiento, y explique cómo se aplica para crear tipos de datos abstractos.
- 22.5. Explique el significado de los siguientes términos en la terminología orientada a objetos: *método*, *signatura*, *mensaje*, *colección*.
- 22.6. ¿Qué relación hay entre un tipo y su subtipo en una jerarquía de tipos? ¿Qué diferencia hay entre una jerarquía de tipos y una jerarquía de clases?
- 22.7. ¿Qué diferencia hay entre los objetos persistentes y los transitorios? ¿Cómo se maneja la persistencia en la generalidad de los sistemas de bases de datos OO?
- 22.8. ¿Qué diferencias hay entre la herencia normal, la herencia múltiple y la herencia selectiva?
- 22.9. Analice el concepto de polimorfismo/sobrecarga de operadores.
- 22.10. ¿Qué diferencia hay entre los objetos complejos estructurados y los no estructurados?
- 22.11. ¿Qué diferencia hay entre la semántica de propiedad y la semántica de referencia en los objetos complejos estructurados?

- 22.12. ¿Qué es la creación de versiones? ¿Por qué es importante? ¿Qué diferencia hay entre las versiones y las configuraciones?
- 22.13. ¿Qué diferencia hay entre valores y objetos en O2? ¿Cómo se hacen persistentes los objetos en O2?
- 22.14. ¿Cómo se representan los vínculos en un modelo de datos OO? ¿Qué son las referencias inversas? ¿Cómo se declaran las referencias inversas en ObjectStore?

Ejercicios

- 22.15. Diseñe un esquema OO para una aplicación de base de datos que le interese. Construya primero un esquema EER para la aplicación; luego cree las clases correspondientes en O2. Especifique varios métodos para cada clase. Repita el ejercicio especificando las clases para ObjectStore.
- 22.16. Considere la base de datos AEROPUERTO descrita en el ejercicio 21.18. Especifique varias operaciones/métodos que, a su parecer, serían pertinentes para esa aplicación. Especifique las clases y métodos O2 para esa base de datos. Repita el ejercicio para ObjectStore.
- 22.17. Transforme el esquema completo EER UNIVERSIDAD de la figura 21.10 a clases O2 y ObjectStore. Incluya métodos apropiados para cada clase.
- 22.18. Transforme el esquema ER COMPANÍA de la figura 3.2 a clases O2 y ObjectStore. Incluya métodos apropiados para cada clase.

Bibliografía selecta

Los conceptos de bases de datos orientadas a objetos son una amalgama de conceptos provenientes de los lenguajes de programación OO y de los sistemas de bases de datos y los modelos de datos conceptuales. Varios libros de texto describen lenguajes de programación OO; por ejemplo, Stroustrup (1986) y Pohl (1991), para C++, y Goldberg (1989) para SMALLTALK. Un libro reciente de Cattell (1991) describe conceptos de bases de datos OO.

Existe una enorme bibliografía sobre las bases de datos OO, por lo que sólo podemos ofrecer aquí una muestra representativa. Los números de octubre de 1991 de CACM y de diciembre de 1990 de IEEE *Computer* describen conceptos y sistemas de bases de datos orientadas a objetos. Dittrich (1986) y Zaniolo *et al.* (1986) hacen un estudio de los conceptos básicos de los modelos de datos orientados a objetos. Uno de los primeros artículos sobre bases de datos orientadas a objetos es Baroody y DeWitt (1981). En Banerjee *et al.* (1987), Borgida *et al.* (1989), Kappel y Schrefl (1991) y Bouzeghoub y Metais (1991) se analizan cuestiones de modelado de datos en bases de datos OO. Su *et al.* (1988) presenta un modelo de datos orientado a objetos que se está usando en aplicaciones CAD/CAM. Mitschang (1989) extiende el álgebra relacional para cubrir objetos complejos. Los lenguajes de consulta e interfaces gráficas con el usuario para OO se describen en Gysens *et al.* (1990), Kim (1989), Alashkur *et al.* (1989), Bertino *et al.* (1992), Agrawal *et al.* (1990) y Cruz (1992).

Osborn (1989) y Zicari (1991) examinan la evolución de los esquemas en bases de datos OO. Heiler y Zdonick (1990) y Rudensteiner (1992) tratan las vistas de objetos, y Barsalou *et al.* (1991) analiza vistas de objetos que se construyen sobre una base de datos relacional. Bertino y Kim (1989) estudia la indización en bases de datos OO. Las cuestiones de almacenamiento, arquitectura y rendimiento en bases de datos OO se tratan en Biliris (1992), Cattell y Skeen (1992), Cheng (1991),

DeWitt *et al.* (1990), Shekita y Carey (1989), Willshire (1991), Chang y Katz (1989) y Tsangaris y Naughton (1992). Lehman y Lindsay (1989) tratan la implementación de objetos extensos. Isotras y Copinath (1992) analiza la creación de versiones de los objetos.

El sistema O2 se describe en Deux *et al.* (1991) y en un libro reciente (Bancilhon *et al.* 1992) que incluye una lista de referencias a otras publicaciones que se ocupan de diversos aspectos de O2. El modelo O2 se formalizó en Vélez *et al.* (1989). El sistema ObjectStore se describe en Lamb *et al.* (1991). Fishman *et al.* (1987) y Wilkinson *et al.* (1990) analizan IRIS, un SGBD orientado a objetos creado en los laboratorios Hewlett-Packard. Maier *et al.* (1986) y Butterworth *et al.* (1991) describen el diseño de GEMSTONE. Un sistema OO con arquitectura abierta, creado por Texas Instruments, se describe en Thompson *et al.* (1993). El sistema ODE creado en ATT Bell Labs se describe en Agrawal y Gehani (1989). El sistema ORION creado en MCC se describe en Kim *et al.* (1990). Morsi *et al.* (1992) describe una base de prueba OO.

El polimorfismo en las bases de datos y en los lenguajes de programación orientados a objetos se estudia en Osborn (1989), Atkinson y Buneman (1987) y Danforth y Tomlinson (1988). La identidad de objetos se trata en Abiteboul y Kanellakis (1989). Los lenguajes de programación OO para bases de datos se analizan en Kent (1991). Las restricciones de objetos se estudian en Delcambre *et al.* (1991) y en Elmastri *et al.* (1993). La autorización y la seguridad en bases de datos OO se examinan en Rabitti *et al.* (1991) y Bertino (1992).

CAPÍTULO 23

Bases de datos distribuidas y arquitectura cliente-servidor

En un **sistema de base de datos centralizado**, todos los componentes del sistema residen en un solo computador o sitio. Los componentes consisten en los datos, el software del SGBD y los dispositivos de almacenamiento secundario asociados, como discos para el almacenamiento en línea de la base de datos y cintas para las copias de seguridad. Podemos tener acceso remoto a una base de datos centralizada a través de terminales conectadas al sitio; sin embargo, los datos y el software del SGBD residen principalmente en un solo sitio. En años recientes se ha observado una marcada tendencia hacia la **distribución** de los sistemas de cómputo en múltiples sitios que se interconectan a través de una **red de comunicaciones**. En este capítulo estudiaremos el desarrollo de los **sistemas de bases de datos distribuidas (SBDD)** y las técnicas empleadas en su implementación. El software con que se implementa un sistema así se denomina **sistema de gestión de bases de datos distribuidas (SGBDD)**.

En la sección 23.1 analizaremos las razones para optar por la distribución, y presentaremos conceptos de SBDD. La sección 23.2 es un panorama sobre la arquitectura cliente-servidor, que se está popularizando para los sistemas distribuidos en general, y para las bases de datos distribuidas en particular. La sección 23.3 presenta técnicas para fragmentar una base de datos y distribuir sus porciones en múltiples sitios; esto se conoce también como diseño de bases de datos distribuidas. En la sección 23.4 clasificaremos los diversos tipos de sistemas de bases de datos distribuidas. Las secciones 23.5 y 23.6 explicarán el procesamiento de consultas distribuidas y las técnicas de control de concurrencia y recuperación, respectivamente.

Si se desea una introducción más breve al tema de las bases de datos distribuidas, se puede pasar por alto partes de las secciones 23.3.3, 23.5 y 23.6, o todas ellas.

23.1 Introducción a los conceptos de SGBD distribuidos

Una **base de datos distribuida** es una colección de datos que pertenece lógicamente al mismo sistema pero que está dispersa físicamente entre los sitios de una red de computadores. Varios factores han dado pie a la creación de los SBDD. Entre las ventajas potenciales de los SBDD están las siguientes:

- *La naturaleza distribuida de algunas aplicaciones de bases de datos:* Muchas de estas aplicaciones están distribuidas naturalmente en diferentes lugares. Por ejemplo, una compañía puede tener oficinas en varias ciudades, o un banco puede tener múltiples sucursales. Es natural que las bases de datos empleadas en tales aplicaciones estén distribuidas en esos lugares. Muchos usuarios locales tienen acceso exclusivamente a los datos que están en el lugar, pero otros usuarios globales —como la oficina central de la compañía— pueden requerir acceso ocasional a los datos almacenados en varios de estos lugares. Cabe señalar que, por lo regular, los datos en cada sitio local describen un “mínimo mundo” en ese sitio. Las fuentes de los datos y la mayoría de los usuarios y aplicaciones de la base de datos local residen físicamente en ese lugar.
- *Mayor fiabilidad y disponibilidad:* Éstas son dos de las ventajas potenciales de las bases de datos distribuidas que se citan más comúnmente. La **fiabilidad** se define a grandes rasgos como la probabilidad de que un sistema esté en funciones en un momento determinado, y la **disponibilidad** es la probabilidad de que el sistema esté disponible continuamente durante un intervalo de tiempo. Cuando los datos y el software del SGBD están distribuidos en varios sitios, un sitio puede fallar mientras que los demás siguen operando. Sólo los datos y el software que residen en el sitio que falló están inaccesibles. Esto mejora tanto la fiabilidad como la disponibilidad. Se logran mejoras adicionales si se *replican*, con un criterio adecuado, los datos y el software en más de un sitio. En un sistema centralizado, el fallo de un solo sitio hace que el sistema completo deje de estar disponible para todos los usuarios.
- *Posibilidad de compartir los datos al tiempo que se mantiene un cierto grado de control local:* En algunos tipos de SBDD (véase la Sec. 23.4), es posible controlar los datos y el software localmente en cada sitio. Sin embargo, los usuarios de otros sitios remotos pueden tener acceso a ciertos datos a través del software del SGBDD. Esto hace posible el compartimiento controlado de los datos en todo el sistema distribuido.
- *Mejor rendimiento:* Cuando una base de datos grande está distribuida en múltiples sitios, hay bases de datos más pequeñas en cada uno de éstos. En consecuencia, las consultas locales y las transacciones que tienen acceso a datos de un solo sitio tienen un mejor rendimiento porque las bases de datos locales son más pequeñas. Además, cada sitio tiene un menor número de transacciones en ejecución que si todas las transacciones se enviaran a una sola base de datos centralizada. En el caso de transacciones que impliquen acceso a más de un sitio, el procesamiento en los diferentes sitios puede efectuarse en paralelo, reduciéndose así el tiempo de respuesta.

La distribución produce un aumento en la complejidad del diseño y en la implementación del sistema. Para obtener las ventajas potenciales antes citadas, el software de SGBDD debe contar con las funciones de un SGBD centralizado y además con las siguientes:

- La capacidad de tener acceso a sitios remotos y transmitir consultas y datos entre los diversos sitios a través de una red de comunicaciones.
- La capacidad de seguir la pista a la distribución y la replicación de los datos en el catálogo del SGBDD.
- La capacidad de elaborar estrategias de ejecución para consultas y transacciones que tienen acceso a datos de más de un sitio.
- La capacidad de decidir a cuál copia de un elemento de información replicado se tendrá acceso.
- La capacidad de mantener la consistencia de las copias de un elemento de información replicado.
- La capacidad de recuperarse de caídas de sitios individuales y de nuevos tipos de fallos, como el fallo de un enlace de comunicación.

Por sí solas, estas funciones elevan la complejidad de un SGBDD en comparación con un SGBD centralizado. Para aprovechar todas las ventajas potenciales de la distribución debemos hallar soluciones satisfactorias a estas cuestiones. Es difícil incluir toda esta funcionalidad adicional, y más difícil aún encontrar las soluciones óptimas. Cuando consideramos el diseño de una base de datos distribuida surgen complejidades adicionales. Específicamente, debemos decidir cómo distribuir los datos entre los sitios y cuáles datos, si acaso, replicar.

En el nivel físico, de **hardware**, los principales factores que distinguen un SGBD de un sistema centralizado son los siguientes:

- Hay múltiples computadores, llamados **sitios** o **nodos**.

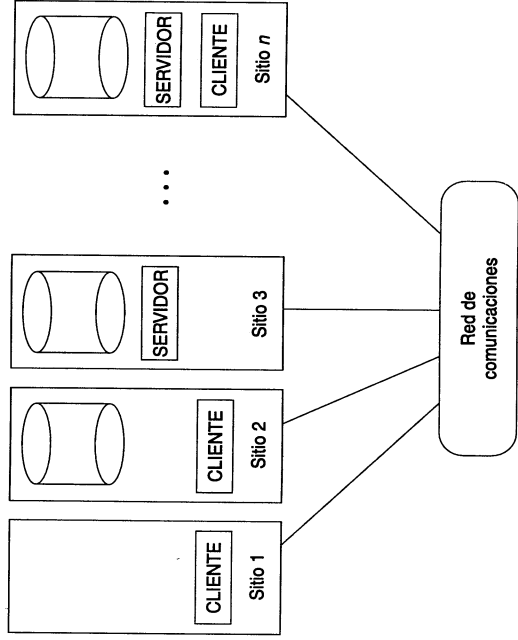


Figura 23.1 Arquitectura física cliente-servidor simplificada para un SGBD.

- Estos sitios deben estar comunicados por medio de algún tipo de **red de comunicaciones** para transmitir datos y órdenes entre los sitios, como se muestra en la figura 23.1.

Los sitios pueden estar cercanos físicamente —digamos, dentro del mismo edificio o grupo de edificios adyacentes— y conectados a través de una **red de área local**, o pueden estar distribuidos geográficamente a grandes distancias y conectados a través de una **red de larga distancia**. Las redes de área local por lo regular usan cables, en tanto que las redes de larga distancia emplean líneas telefónicas o satélites. También es posible usar una combinación de ambos tipos de redes.

Las redes pueden tener diferentes **topologías** que definen los caminos de comunicación directa entre los sitios. Por ejemplo, puede haber enlaces directos entre los sitios 1 y 2 y entre los sitios 2 y 3 pero no entre los sitios 1 y 3; en tal caso, la comunicación entre los sitios 1 y 3 debe pasar por el sitio 2. El tipo y la topología de la red empleada puede tener un efecto significativo sobre el rendimiento y, por ende, sobre las estrategias para el procesamiento de consultas distribuidas y el diseño de bases de datos distribuidas. Sin embargo, en lo tocante a los aspectos arquitectónicos de alto nivel, no importa qué tipo de red se use; sólo importa que cada uno de los sitios pueda comunicarse, directa o indirectamente, con todos los demás. En el resto de este capítulo, supondremos que existe algún tipo de red de comunicaciones entre los sitios, sea cual sea la topología específica.

23.2 Panorama sobre la arquitectura cliente-servidor

La **arquitectura cliente-servidor** se creó para manejar los nuevos entornos de cómputo en los que un gran número de computadores personales, estaciones de trabajo, servidores de archivos, impresoras y otros equipos están interconectados a través de una red. La idea es definir **servidores especializados** con funciones específicas. Por ejemplo, es posible conectar como clientes varias estaciones de trabajo o computadores personales sin disco a una máquina *servidora de archivos* que mantenga los archivos de los usuarios de los clientes. Se podría designar otra máquina como *servidora de impresoras* conectándola a diversas impresoras; en adelante, todas las solicitudes de impresión se enviarían a esa máquina. De esta manera, los recursos que proveen los servidores especializados están a disposición de muchos clientes. Esta idea puede aplicarse al software, al almacenar programas especializados— como un SGBD o un paquete CAD (diseño asistido por computador)— en máquinas servidoras específicas, a fin de ponerlo a disposición de múltiples clientes.

La arquitectura cliente-servidor se está incorporando cada vez más en los paquetes de SGBD comerciales conforme se van orientando hacia la distribución. La idea consiste en dividir el software de SGBD en dos niveles —cliente y servidor— para reducir su complejidad. Esto se ilustra en la figura 23.1; algunos sitios pueden ejecutar exclusivamente el software de cliente (éstos podrían ser máquinas sin disco, como el sitio 1, o máquinas con disco, como el sitio 2), mientras que otros sitios podrían ser máquinas servidoras dedicadas que ejecutan sólo el software de servidor, como el sitio 3. Algunos sitios más podrían manejar módulos de cliente y servidor, como el sitio *n* de la figura 23.1.

Todavía no se ha establecido una forma precisa de dividir la funcionalidad del SGBD entre cliente y servidor, y se han propuesto diferentes enfoques. Una posibilidad consiste en incluir la funcionalidad de un SGBD centralizado en el nivel de servidor. Varios productos de

SCBD relacionales han adoptado este enfoque, en el que se proporciona un **servidor SQL** a los clientes. Cada cliente debe entonces formular las consultas SQL apropiadas y proveer la interfaz con el usuario y las funciones de interfaz con los lenguajes de programación. Puesto que SQL es una norma relacional, diversos servidores SQL diferentes, posiblemente provenientes de distintos proveedores, pueden aceptar órdenes SQL. El cliente también puede hacer referencia a un diccionario de datos que tenga información sobre la distribución de los datos entre los diferentes servidores SQL, así como módulos para descomponer una consulta global en varias consultas locales que se pueden ejecutar en los diferentes sitios. Analizaremos un módulo de este tipo más a fondo en la sección 23.5. La interacción entre el cliente y el servidor podría efectuarse como sigue durante el procesamiento de una consulta SQL:

1. El cliente analiza sintácticamente la consulta del usuario y la descompone en varias consultas de sitio independientes. Cada consulta de sitio se envía al sitio servidor apropiado.
2. Cada servidor procesa la consulta local y envía la relación resultante al sitio cliente.
3. El sitio cliente combina los resultados de las subconsultas para producir el resultado de la consulta original.

En este enfoque, el servidor SQL recibe también el nombre de **procesador de bases de datos** (DP: *database processor*) o **máquina dorsal**, en tanto que el cliente se denomina **procesador de aplicaciones** (AP: *application processor*) o **máquina frontal**. El usuario puede especificar la interacción entre cliente y servidor en el nivel de cliente o a través de un módulo cliente especializado. Por ejemplo, el usuario podría saber qué datos están almacenados en cada servidor, descomponer manualmente una consulta en subconsultas de sitios y presentar subconsultas individuales a los diferentes sitios. Las tablas resultantes podrían combinarse explícitamente mediante otra consulta del usuario en el nivel de cliente. La alternativa es hacer que el módulo cliente lleve a cabo estas acciones automáticamente.

Otro enfoque, adoptado por algunos SCBD orientados a objetos (véase el Cap. 22), divide los módulos de software del SCBD entre cliente y servidor de una manera más integrada. Por ejemplo, el **nivel servidor** podría incluir la parte del software del SCBD encargada de manejar el almacenamiento de los datos en páginas de disco, el control de concurrencia y la recuperación locales, el almacenamiento intermedio y en *cache* de las páginas de disco y otras funciones similares. Por su lado, el **nivel cliente** podría manejar la interfaz con el usuario, las funciones de diccionario de datos, la interacción del SCBD con los compiladores de lenguajes de programación, la optimización/control de concurrencia/recuperación de consultas globales, la estructuración de objetos complejos a partir de los datos en almacenamiento intermedio y otras funciones similares. En este enfoque, la interacción cliente-servidor está más firmemente acoplada y la realizan internamente los módulos del SCBD, no los usuarios.

En un SCBDD representativo, se acostumbra dividir los módulos de software en tres niveles:

- El software **servidor** se encarga de la gestión local de los datos en un sitio, en forma muy parecida al software de un SCBD centralizado.
- El software **cliente** se encarga de casi todas las funciones de distribución; tiene acceso a información almacenada en el catálogo del SCBDD sobre la distribución de los datos y procesa todas las solicitudes que requieren acceso a más de un sitio.

- El **software de comunicaciones** (a veces en colaboración con un **sistema operativo distribuido**) proporciona las primitivas de comunicación con que el cliente transmite órdenes y datos entre los diferentes sitios según las necesidades. Éste no es estrictamente parte del SCBDD, pero provee primitivas y servicios de comunicación esenciales.

El **cliente** se encarga de generar un plan de ejecución distribuido para una consulta o transacción en múltiples sitios y de supervisar la ejecución distribuida enviando órdenes a los servidores. Estas órdenes son las consultas y transacciones locales que han de ejecutarse, así como órdenes para transmitir datos a otros clientes o servidores. Por ello, todos los sitios en los que se introduzcan consultas a múltiples sitios deberán contar con el software de cliente. Hablaremos más sobre el procesamiento de consultas distribuidas en la sección 23.5. Otra función que está bajo el control del cliente es la de asegurar que las réplicas de un elemento de información sean consistentes, para lo cual se vale de técnicas de control de concurrencia distribuidas (o globales). El cliente también debe asegurar que las transacciones globales sean atómicas realizando acciones de recuperación y el control de concurrencia globales. En la sección 23.6 estudiaremos la recuperación y el control de concurrencia globales. La figura 23.2 muestra una vista lógica de un SBDD; aquí se muestran los clientes y servidores sin especificar el sitio en que reside cada uno.

Una posible función del cliente es *ocultar* al usuario los detalles de la distribución de los datos; esto es, permite al usuario escribir consultas y transacciones globales como si la base de datos fuera centralizada, sin tener que especificar los sitios en donde residen los datos a los que se hace referencia en la consulta o transacción. Esta propiedad se llama **transparencia de distribución**. Como algunos SCBDD no cuentan con transparencia de distribución, exigen que los usuarios conozcan los detalles de la distribución de los datos. En este caso, los usuarios deben especificar los sitios en donde residen los datos a los que las consultas y transacciones globales hacen referencia; el usuario por lo regular *anexa el nombre del sitio* a todas las referencias a una relación, archivo o tipo de registros. Cuando hay transparencia de distribución, se presenta al usuario un esquema que no incluye información de distribución, y el SCBDD mismo registra los sitios donde están ubicados los datos en el **catálogo de SCBDD**. Con la ayuda de esta información, el software cliente puede descomponer una consulta en varias subconsultas que se pueden ejecutar en los diversos sitios, y puede planear la forma de transmitir los resultados de las subconsultas a otros sitios para un procesamiento adicional

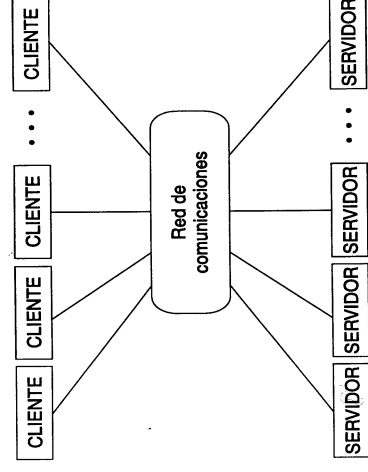


Figura 23.2 Arquitectura lógica cliente-servidor simplificada para un SBDD.

o para producir el resultado. Los SGBDD que ofrecen transparencia de distribución hacen más simple para el usuario la especificación de consultas y transacciones, pero requieren software más complejo. Los SGBDD que no ofrecen transparencia de distribución dejan al usuario la responsabilidad de especificar el sitio de cada relación o archivo, con lo que el software puede ser más simple.

23.3 Técnicas de fragmentación, replicación y reparto de los datos para el diseño de bases de datos distribuidas

Esta sección se ocupa de unas técnicas para dividir la base de datos en unidades lógicas, llamadas **fragmentos**, cuyo almacenamiento puede asignarse a los diversos sitios. En ella también se explican la **replicación de los datos**, con la que es posible almacenar ciertos datos en más de un sitio, y el proceso de **repartir fragmentos** —o réplicas de fragmentos— para almacenarse en los diferentes sitios. Estas técnicas se utilizan durante el proceso de **diseño de bases de datos distribuidas**. La información concerniente a la fragmentación de los datos, el reparto y la replicación se almacena en un **catálogo global del sistema** al que tiene acceso el software cliente cuando es necesario.

23.3.1 Fragmentación de los datos

En un SGBDD, es preciso tomar decisiones respecto a los sitios en los que se almacenarán las porciones de la base de datos. Por ahora supondremos que *no hay replicación*, esto es, que cada archivo —o porción de archivo— se almacenará en un solo sitio. Hablaremos de la replicación y de sus efectos más adelante en esta sección. También usaremos la terminología de las *bases de datos relacionadas*; conceptos similares se aplican a los demás modelos de datos. Supondremos que estamos partiendo de un esquema de base de datos relacional y debemos decidir cómo distribuir las relaciones entre los diferentes sitios. Para ilustrar nuestro análisis, nos referiremos al esquema de base de datos relacional de la figura 6.5.

Antes de decidir cómo distribuir los datos, debemos determinar las *unidades lógicas* de la base de datos que se van a distribuir. Las unidades lógicas más simples son las relaciones mismas; es decir, cada relación *completa* se almacenará en un sitio específico. En nuestro ejemplo, deberemos decidir en qué sitio almacenar cada una de las relaciones EMPLEADO, DEPARTAMENTO, PROYECTO, TRABAJA EN Y DEPENDIENTE de la figura 6.5. En muchos casos, empero, es posible dividir una relación en unidades lógicas más pequeñas para su distribución. Por ejemplo, consideremos la base de datos de una compañía que se muestra en la figura 6.6, y supongamos que hay tres sitios de computador, uno para cada departamento de la compañía. Desde luego, en una situación real, habrá muchas más tuplas en las relaciones. Tal vez queramos almacenar la información referente a cada departamento en el sitio de computador correspondiente a ese departamento. Para ello, necesitaremos dividir cada relación mediante una técnica llamada fragmentación horizontal.

Fragmentación horizontal. Un **fragmento horizontal** de una relación es un subconjunto de las tuplas de esa relación. Las tuplas que pertenecen al fragmento horizontal se especifican mediante una condición sobre uno o más de los atributos de la relación. Con frecuencia, sólo interviene un atributo. Por ejemplo, podríamos definir tres fragmentos horizontales

en la relación EMPLEADO de la figura 6.6 con las siguientes condiciones: (ND = 5), (ND = 4) y (ND = 1); cada fragmento contiene las tuplas EMPLEADO que pertenecen a un departamento en particular. De manera similar, podemos definir tres fragmentos horizontales para la relación PROYECTO de la figura 6.6 con las condiciones (NÚMD = 5), (NÚMD = 4) y (NÚMD = 1); cada fragmento contiene las tuplas PROYECTO controladas por un departamento en particular. La fragmentación horizontal divide una relación “horizontalmente” agrupando filas para crear subconjuntos de tuplas, donde cada subconjunto tiene un cierto significado lógico. Estos fragmentos pueden entonces asignarse a diferentes sitios en el sistema distribuido.

Fragmentación vertical. Un **fragmento vertical** de una relación mantiene sólo ciertos atributos de la relación. Por ejemplo, si quisiéramos dividir la relación EMPLEADO en dos fragmentos verticales, el primero incluiría información personal —NOMBRE, FECHAN, DIRECCIÓN y SEXO— y el segundo información relacionada con el trabajo: NSS, SALARIO, NSSUPER, ND. Esta fragmentación vertical no es del todo apropiada porque, si ambos fragmentos se almacenan por separado, no podremos juntar otra vez las tuplas de empleados originales, ya que *no existe un atributo común* entre los dos fragmentos. Es necesario incluir el *atributo de clave primaria* en todo fragmento vertical para que sea posible reconstruir la relación completa a partir de los fragmentos. Por tanto, habrá que añadir el atributo NSS al fragmento de información personal. La fragmentación vertical divide una relación “verticalmente” por columnas.

Observe que cada fragmento horizontal de una relación R se puede especificar con una operación $\sigma_{C_i}(R)$ del álgebra relacional. Un conjunto de fragmentos horizontales cuyas condiciones $C_1, C_2, \dots, C_n$ incluyen todas las tuplas de R —esto es, todas las tuplas de R satisfacen (C_1 OR C_2 OR ... OR C_n)— se denomina **fragmentación horizontal completa** de R. En muchos casos una fragmentación horizontal completa es además **disjunta**; es decir, ninguna tupla de R satisface (C_i AND C_j), para cualquier $i \neq j$. En nuestros dos ejemplos anteriores de fragmentación horizontal, las relaciones EMPLEADO y PROYECTO fueron tanto completas como disjuntas. Para reconstruir la relación R a partir de una fragmentación horizontal *completa*, necesitamos aplicar la operación UNIÓN a los fragmentos.

Un fragmento vertical de una relación R puede especificarse con una operación $\pi_{L_j}(R)$ del álgebra relacional. Un conjunto de fragmentos verticales cuyas listas de proyección $L_1, L_2, \dots, L_n$ incluyen todos los atributos de R, pero sólo comparten el atributo de clave primaria de R, se denomina **fragmentación vertical completa** de R. En este caso, las listas de proyección satisfacen estas dos condiciones:

- $L_1 \cup L_2 \cup \dots \cup L_n = \text{ATR}(R)$.
- $L_i \cap L_j = \text{CLP}(R)$ para cualquier $i \neq j$, donde $\text{ATR}(R)$ es el conjunto de atributos de R y $\text{CLP}(R)$ es la clave primaria de R.

Para reconstruir la relación R a partir de una fragmentación vertical *completa*, aplicamos la operación de UNIÓN EXTERNA a los fragmentos. Cabe señalar que también podríamos aplicar la operación de REUNIÓN EXTERNA COMPLETA y obtener el mismo resultado que con una fragmentación vertical completa. Los dos fragmentos verticales de la relación EMPLEADO con listas de proyección $L_1 = \{\text{NSS, NOMBRE, FECHAN, DIRECCIÓN, SEXO}\}$ y $L_2 = \{\text{NSS, SALARIO, NSSUPER, ND}\}$ constituyen una fragmentación vertical completa de EMPLEADO.

Dos fragmentos horizontales que no son ni completos ni disjuntos son los que se definen sobre la relación EMPLEADO de la figura 6.5 con las condiciones (SALARIO > 50000) y (ND = 4); puede ser que no incluyan todas las tuplas empleado, y que tengan tuplas en

común. Dos fragmentos verticales que no son completos son los que se definen con las listas de atributos $L1 = \{\text{NOMBRE, DIRECCIÓN}\}$ y $L2 = \{\text{NSS, NOMBRE, SALARIO}\}$; estas listas violan las dos condiciones de una fragmentación vertical completa.

Fragmentación mixta. Podemos entremezclar los dos tipos de fragmentación, para obtener una **fragmentación mixta**. Por ejemplo, podemos combinar las fragmentaciones horizontal y vertical de la relación EMPLEADO que vimos antes para tener una fragmentación mixta que incluya seis fragmentos. En este caso la relación original se puede reconstruir aplicando operaciones de UNIÓN y UNIÓN EXTERNA (o REUNIÓN EXTERNA) en el orden apropiado. En general, un **fragmento** de una relación R se puede especificar con una combinación de operaciones SELECCIONAR-PROYECTAR $\pi_1(\sigma_C(R))$. Si $C = \text{TRUE}$ y $L \neq \text{ATR}(R)$, obtendremos un fragmento vertical, y si $C \neq \text{TRUE}$ y $L = \text{ATR}(R)$, obtendremos un fragmento horizontal. Por último, si $C \neq \text{TRUE}$ y $L \neq \text{ATR}(R)$ obtendremos un fragmento mixto. Cabe señalar que una relación completa puede considerarse un fragmento con $C = \text{TRUE}$ y $L = \text{ATR}(R)$. En el análisis que sigue, utilizamos el término *fragmento* para referirnos a una relación o a cualquier uno de los tipos de fragmentos mencionados.

Un **esquema de fragmentación** de una base de datos es una definición de un conjunto de fragmentos que incluye todos los atributos y tuplas de la base de datos y satisface la condición de que la base de datos completa se puede reconstruir a partir de los fragmentos mediante alguna secuencia de operaciones UNIÓN EXTERNA (o REUNIÓN EXTERNA) y UNIÓN. En ocasiones también es útil —aunque no necesario— que todos los fragmentos sean disjuntos, excepto la repetición de las claves primarias entre los fragmentos verticales (o mixtos). En este último caso, toda la replicación y distribución de los fragmentos se especifica claramente en una etapa subsecuente, aparte de la fragmentación.

Un **esquema de reparto** describe el reparto de fragmentos entre los sitios del SBDP; por tanto, es una correspondencia que especifica el sitio o sitios donde se almacena cada fragmento. Si un fragmento se almacena en más de un sitio, se dice que está **replicado**. A continuación trataremos la replicación y el reparto de los datos.

23.3.2 Replicación y reparto de los datos

La replicación resulta útil para mejorar la disponibilidad de los datos. El caso más extremo es la replicación de *toda la base de datos* en todos los sitios del sistema distribuido, creando así una base de datos distribuida **totalmente replicada**. Esto puede mejorar la disponibilidad notablemente porque el sistema puede seguir operando mientras, por lo menos, uno de los sitios esté activo. También mejora el rendimiento de la obtención de datos en consultas globales, porque el resultado de semejante consulta se puede obtener localmente en cualquier sitio; así, una consulta de obtención de datos se puede procesar en el sitio local donde se introduce, si dicho sitio cuenta con un módulo servidor. La desventaja de la replicación completa es que puede reducir drásticamente la rapidez de las operaciones de actualización, pues una sola actualización lógica se deberá ejecutar con todas y cada una de las copias de la base de datos a fin de mantener la consistencia. Esto es especialmente cierto si existen muchas copias de la base de datos. Con la replicación completa, las técnicas de control de concurrencia y recuperación se tornan más costosas de lo que serían sin replicación, como veremos en la sección 23.6.

El extremo opuesto a la replicación completa es **no tener ninguna replicación**; esto es, cada fragmento se almacena exactamente en un sitio. En este caso todos los fragmentos

deben ser disjuntos, con excepción de la repetición de claves primarias entre los fragmentos verticales (o mixtos). Esto se denomina también **reparto no redundante**.

Entre estos dos extremos, tenemos una amplia gama de **replicación parcial** de los datos; es decir, algunos fragmentos de la base de datos pueden estar replicados y otros no. El número de copias de cada fragmento puede ir desde una hasta el número total de sitios en el sistema distribuido. En ocasiones se llama **esquema de replicación** a una descripción de la replicación de los fragmentos.

Cada fragmento —o cada copia de un fragmento— se debe asignar a un sitio determinado en el sistema distribuido. Este proceso se denomina **distribución de los datos** (o **reparto de los datos**). La elección de sitios y el grado de replicación dependen de los objetivos de rendimiento y disponibilidad para el sistema y de los tipos y frecuencias de transacciones introducidas en cada sitio. Por ejemplo, si se requiere una alta disponibilidad, y si las transacciones se pueden introducir en cualquier sitio y si la mayoría de ellas son de obtención de datos, entonces una base de datos completamente replicada será una buena opción. Sin embargo, si por lo regular ciertas transacciones que tienen acceso a partes específicas de la base de datos se introducen en un solo sitio, se podría asignar el conjunto de fragmentos correspondiente exclusivamente a ese sitio. Los datos que se utilizan en múltiples sitios se pueden replicar en esos sitios. Si se efectúan muchas actualizaciones, puede ser conveniente limitar la replicación. Encontrar una solución óptima, o siquiera buena, para el reparto de datos distribuidos es un problema de optimización muy complejo.

23.3.3 Ejemplo de fragmentación, reparto y replicación*

Ahora consideraremos un ejemplo de cómo fragmentar y distribuir la base de datos de la compañía que presentamos en las figuras 6.5 y 6.6. Supongamos que la compañía tiene tres sitios de computador, uno para cada uno de los departamentos actuales. Los sitios 2 y 3 son para los departamentos 5 y 4, respectivamente. En cada uno de estos sitios, esperamos que haya un acceso frecuente a la información de EMPLEADO y PROYECTO de los empleados que *trabajan en ese departamento* y los proyectos controlados por ese departamento. Además, suponemos que en estos sitios se tiene acceso principalmente a los atributos NOMBRE, NSS, SALARIO y NSSUPER de EMPLEADO. El sitio 1 lo usa la oficina central de la compañía y tiene acceso con regularidad a toda la información de empleados y proyectos, además de utilizar la información de DEPENDIENTE para cuestiones de seguros.

De acuerdo con estos requerimientos, la base de datos completa de la figura 6.6 se puede almacenar en el sitio 1. Para determinar los fragmentos que deben replicarse en los sitios 2 y 3, primero podemos fragmentar horizontalmente las relaciones EMPLEADO, PROYECTO, DEPARTAMENTO y LUGARES_DEPTOS por número de departamento —llamado ND, NÚMERO DE DEPARTAMENTO, respectivamente, en la figura 6.5—. Después podemos fragmentar verticalmente los fragmentos resultantes de EMPLEADO a modo de incluir sólo los atributos {NOMBRE, NSS, SALARIO, NSSUPER, ND}. La figura 23.3 muestra los fragmentos mixtos EMPD5 y EMPD4, que contienen las tuplas EMPLEADO que satisfacen las condiciones $ND = 5$ y $ND = 4$, respectivamente. Los fragmentos horizontales de PROYECTO, DEPARTAMENTO y LUGARES_DEPTOS se fragmentan de manera similar por número de departamento. Todos estos fragmentos —almacenados en los sitios 2 y 3— están replicados porque también se almacenan en el sitio 1 de la oficina central.

Ahora debemos fragmentar la relación TRABAJO_EN y decidir cuáles fragmentos almacenar en los sitios 2 y 3. Nos enfrentamos al problema de que ningún atributo de TRABAJO_EN

(a)

EMPDP5	NOMBREP	INIC	APELLIDO	NSS	SALARIO	NSSSUPER	ND
	José	B	Silva	123456789	30000	333445555	5
	Federico	T	Vicarraga	333445555	40000	888665555	5
	Ramón	K	Nieto	666884444	38000	333445555	5
	Joséfa	A	Esparza	453453453	25000	333445555	5

DEP5	NOMBRE	NÚMEROD	NSSGTE	FECHAINICGTE	LUGS_DEP5	NÚMEROD	LUGAR
Investigación		5	333445555	22-MAY-78		5	Belén
						5	Sacramento
						5	Higueras

TRABAJA_EN5	NSSE	NÚMP	HORAS
123456789	1	32.5	
123456789	2	7.5	
666884444	3	40.0	
453453453	1	20.0	
453453453	2	20.0	
333445555	2	10.0	
333445555	3	10.0	
333445555	10	10.0	
333445555	20	10.0	

PROYS5	NOMBREPR	NÚMEROP	LUGARP	NÚMD
	Producto X	1	Belén	5
	Producto Y	2	Sacramento	5
	Producto Z	3	Higueras	5

(b)

EMPDP4	NOMBREP	INIC	APELLIDO	NSS	SALARIO	NSSSUPER	ND
	Alicia	J	Zapata	999887777	25000	987654321	4
	Jazmín	S	Valdés	987654321	43000	888665555	4
	Ahmed	V	Jabbar	987987987	25000	987654321	4

DEP4	NOMBRE	NÚMEROD	NSSGTE	FECHAINICGTE	LUGS_DEP4	NÚMEROD	LUGAR
Administración		4	987654321	01-ENE-85		4	Santiago

TRABAJA_EN4	NSSE	NÚMP	HORAS
333445555	10	10.0	
999887777	30	30.0	
999887777	10	10.0	
987987987	10	35.0	
987987987	30	5.0	
987654321	30	20.0	
987654321	20	15.0	

PROYS4	NOMBREP	NÚMEROP	LUGARP	NÚMD
	Automatización	10	Santiago	4
	Nuevasprestaciones	20	Santiago	4

Figura 23.3 Reparto de fragmentos entre los sitios. (a) Fragmentos de relaciones en el sitio 2. (b) Fragmentos de relaciones en el sitio 3.

indica directamente el departamento al que pertenece cada tupla. De hecho, cada tupla de *TRABAJA_EN* relaciona un empleado *e* con un proyecto *p*. Podríamos fragmentar *TRABAJA_EN* según el departamento *d* al que *e* pertenece o bien según el departamento *d'* que controla *p*. La fragmentación se facilita si tenemos una restricción que especifique que *d = d'* para todas las tuplas *TRABAJA_EN*; esto es, si los empleados sólo pueden trabajar en los proyectos controlados por el departamento al que pertenecen. Sin embargo, no hay tal restricción en

(a)

G1	NSSE	NÚMP	HORAS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0

C1=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=5))

G2	NSSE	NÚMP	HORAS
	333445555	10	10.0

C2=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=4))

G3	NSSE	NÚMP	HORAS
	333445555	20	10.0

C3=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=1))

(b)

G4	NSSE	NÚMP	HORAS

C4=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=6))

G5	NSSE	NÚMP	HORAS

C5=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=4))

G6	NSSE	NÚMP	HORAS

C6=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=1))

(c)

G7	NSSE	NÚMP	HORAS

C7=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=5))

G8	NSSE	NÚMP	HORAS

C8=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=4))

G9	NSSE	NÚMP	HORAS

C9=C AND (NÚMP IN (SELECT NÚMEROP FROM PROYECTO WHERE NÚMD=1))

Figura 23.4 Fragmentos completos y disjuntos de la relación *TRABAJA_EN*. (a) Fragmentos de *TRABAJA_EN* para los empleados que pertenecen al departamento 5 (C = NSS IN (SELECT NSS FROM EMPLEADO WHERE ND = 5)). (b) Fragmentos de *TRABAJA_EN* para los empleados que pertenecen al departamento 4 (C = NSS IN (SELECT NSS FROM EMPLEADO WHERE ND = 4)). (c) Fragmentos de *TRABAJA_EN* para los empleados que pertenecen al departamento 1 (C = NSS IN (SELECT NSS FROM EMPLEADO WHERE ND = 1)).

nuestra base de datos de la figura 6.6. Por ejemplo, la tupla *TRABAJA_EN* <333445555, 10, 10.0> relaciona un empleado que pertenece al departamento 5 con un proyecto que está bajo el control del departamento 4. En este caso podríamos fragmentar *TRABAJA_EN* con base tanto en el departamento al que pertenece el empleado como en el departamento que controla el proyecto, como se muestra en la figura 23.4.

En la figura 23.4, la unión de los fragmentos G1, G2 y G3 produce todas las tuplas *TRABAJA_EN* de los empleados que pertenecen al departamento 5. De manera similar, la unión de los fragmentos G4, G5 y G6 produce todas las tuplas *TRABAJA_EN* de los empleados que pertenecen al departamento 4. Por otro lado, la unión de los fragmentos G1, G4 y G7 produce todas las tuplas *TRABAJA_EN* de los proyectos controlados por el departamento 5. En la figura 23.4 se muestra la condición para cada uno de los fragmentos G1 a G9. Las relaciones que representan vínculos M:N, como *TRABAJA_EN*, a menudo tienen varias fragmentaciones

lógicas posibles. En nuestra distribución de la figura 23.3 decidimos incluir todos los fragmentos que se pueden reunir para dar una tupla EMPLEADO o bien una tupla PROYECTO en los sitios 2 y 3. Por tanto, colocamos la unión de los fragmentos G1, G2, G3, G4 y G7 en el sitio 2 y la unión de los fragmentos G4, G5, G6, G2 y G8 en el sitio 3. Observe que los fragmentos G2 y G4 se replican en ambos sitios. Esta estrategia de reparto permite que la reunión entre los fragmentos locales de EMPLEADO o PROYECTO, en el sitio 2 o en el 3, y el fragmento local de TRABAJO_EN pueda efectuarse totalmente en forma local. Esto demuestra claramente lo complejo que es el problema de fragmentar y repartir una base de datos grande. Las notas bibliográficas contienen referencias a algunos de los trabajos realizados en esta área.

23.4 Tipos de sistemas de bases de datos distribuidas

El término *sistema de gestión de bases de datos distribuidas* puede describir diversos sistemas que presentan muchas diferencias entre sí. El punto principal que todos estos sistemas tienen en común es el hecho de que los datos y el software están distribuidos entre múltiples sitios conectados por alguna especie de red de comunicaciones. En esta sección analizaremos varios tipos de SGBDD y los criterios y factores que distinguen a algunos de estos sistemas.

El primer factor que consideraremos es el **grado de homogeneidad** del software de SGBDD. Si todos los servidores (o SGBD locales individuales) utilizan software idéntico y todos los clientes emplean software idéntico, se dice que el SGBDD es **homogéneo**; en caso contrario se le caracteriza como **heterogéneo**. Otro factor relacionado con el grado de homogeneidad es el **grado de autonomía local**. Si todo acceso al SGBDD debe hacerse a través de un cliente, el sistema **no tiene autonomía local**. Por otro lado, si se permite a las transacciones locales **acceso directo** a un servidor, el sistema tendrá cierto grado de autonomía local.

En un extremo de la gama de autonomía tenemos un SGBDD que da al usuario la impresión de ser un SGBD centralizado. Sólo hay un esquema conceptual, y todo acceso al sistema se hace a través de un cliente, de modo que no hay autonomía local. En el otro extremo nos encontramos con un tipo de SGBDD denominado **SGBDD federado** (o **sistema de múltiples bases de datos**). En un sistema así, cada servidor es un SGBD centralizado independiente y autónomo que tiene sus propios usuarios locales, transacciones locales y DBA, y por ende posee un alto grado de *autonomía local*. Cada servidor puede autorizar el acceso a porciones específicas de su base de datos definiendo un **esquema de exportación**, el cual especifica la parte de la base de datos a la cual puede tener acceso una cierta clase de usuarios no locales. En esencia, un cliente en un sistema así es una interfaz adicional para varios servidores (SGBD locales) que permite a un usuario de múltiples bases de datos (o global) tener acceso a información almacenada en varias de estas *bases de datos autónomas*. Observe que un sistema federado es un híbrido entre los sistemas distribuidos y los centralizados; es un sistema centralizado para los usuarios autónomos locales y es un sistema distribuido para los usuarios globales.

En un sistema heterogéneo de múltiples bases de datos, un servidor puede ser un SGBD relacional, otro un SGBD de red y un tercero un SGBD jerárquico; en tal caso, es necesario contar con un lenguaje de sistema canónico e incluir traductores del lenguaje en el cliente a fin de traducir las subconsultas del lenguaje canónico al lenguaje de cada servidor.

Un tercer aspecto que puede servir para clasificar las bases de datos distribuidas es el **grado de transparencia de la distribución** o, de manera alternativa, el **grado de integración de**

los **esquemas**. Si el usuario percibe un solo esquema integrado sin información alguna relativa a la fragmentación, replicación o distribución, se dice que el SGBDD tiene un *alto grado de transparencia de distribución* (o de integración de esquemas). Por otro lado, si el usuario puede ver toda la fragmentación, el reparto y la replicación, el SGBDD *no tiene transparencia de distribución* ni integración de esquemas. En este caso, el usuario debe hacer referencia a copias específicas de fragmentos en sitios específicos cuando formule una consulta, anexando el nombre del sitio como prefijo de cada nombre de relación o de fragmento.

Esto forma parte del complejo problema de **dar nombres** en los sistemas distribuidos. Hay casos en que el SGBDD no ofrece transparencia de distribución; si es así, es responsabilidad del usuario *especificar sin ambigüedad* el nombre de una copia de relación o de fragmento en particular. La tarea es más difícil en un sistema de múltiples bases de datos, porque es de su poner que cada servidor (SGBD local) se creó de manera independiente, y en consecuencia es muy posible que se hayan usado nombres conflictivos en diferentes servidores. En el caso de un SGBDD que provee un esquema integrado, en cambio, la asignación de nombres se convierte en un problema interno del sistema, porque el usuario percibe un solo esquema sin ambigüedades. El SGBDD debe almacenar en el **catálogo de distribución todas las correspondencias** entre los objetos del esquema integrado y los objetos distribuidos en los diversos procesadores de bases de datos.

23.5 Procesamiento de consultas en bases de datos distribuidas*

Ahora haremos un panorama sobre cómo un SGBDD procesa y optimiza una consulta. Primero analizaremos los costos de comunicación del procesamiento de una consulta distribuida; luego examinaremos una operación especial, llamada *semirreunión*, que sirve para optimizar algunos tipos de consultas en un SGBDD.

23.5.1 Costos de la transferencia de datos en el procesamiento de consultas distribuidas

En el capítulo 16 vimos las cuestiones implicadas en el procesamiento y optimización de una consulta en un SGBD centralizado. En un sistema distribuido varios factores adicionales complican aún más el procesamiento de consultas. El primero es el costo de transferir datos por la red. Estos datos incluyen los archivos intermedios que se transfieren a otros sitios para continuar su procesamiento, así como los archivos de resultado final que tal vez deban transferirse al sitio donde se necesita el resultado de la consulta. Aunque es posible que estos costos no sean demasiado altos si los sitios están conectados en una red de área local de alto rendimiento, adquieren una importancia considerable en otros tipos de redes. Por ello, los algoritmos de optimización de consultas de los SGBDD consideran el objetivo de reducir la *cantidad de transferencia de datos* como criterio de optimización al elegir una estrategia de ejecución de una consulta distribuida.

Ilustraremos lo anterior con dos ejemplos de consultas simples. Supongamos que las relaciones EMPLEADO y DEPARTAMENTO de la figura 6.5 están distribuidas como se muestra en la figura 23.5. Supondremos en este ejemplo que ninguna de las relaciones está fragmentada. De acuerdo con la figura 23.5, el tamaño de la relación EMPLEADO es $100 * 10\ 000 = 10^6$

SITIO 1:

EMPLEADO						
NOMBREP	INIC	APELLIDO	NSS	FECHAN	DIRECCIÓN	SEXO
10 000 registros						
cada registro tiene 100 bytes de longitud						
el campo NSS tiene 9 bytes						
el campo APELLIDO tiene 15 bytes						

SITIO 2:

DEPARTAMENTO			
NOMBREP	NÚMEROD	NSSGTE	FECHAINICGTE
100 registros			
cada registro tiene 35 bytes de longitud			
el campo NÚMEROD tiene 4 bytes			
el campo NSSGTE tiene 9 bytes			

Figura 23.5 Ejemplo para ilustrar el volumen de datos transferidos.

bytes, y el tamaño de la relación DEPARTAMENTO es $35 * 100 = 3500$ bytes. Consideremos la consulta C: "Para cada empleado, obtener su nombre y el nombre del departamento al que pertenece." Esto puede expresarse como sigue en el álgebra relacional:

C: $\pi_{\text{NOMBREP, APELLIDO, NOMBREP} \bowtie (\text{EMPLEADO} \bowtie_{\text{ND=NÚMEROD}} \text{DEPARTAMENTO})}$

El resultado de esta consulta incluirá 10 000 registros, si todo empleado está relacionado con un departamento. Supongamos que cada registro del resultado de la consulta tiene 40 bytes de longitud. La consulta se introduce en un sitio 3 distinto, que se denomina **sitio del resultado** porque es ahí donde se necesita el resultado de la consulta. Ninguna de las dos relaciones, EMPLEADO ni DEPARTAMENTO, reside en el sitio 3. Hay tres estrategias simples para ejecutar esta consulta distribuida:

1. Transferir las relaciones EMPLEADO y DEPARTAMENTO al sitio del resultado, y efectuar la reunión en el sitio 3. En este caso necesitamos transferir un total de $1\ 000\ 000 + 3500 = 1\ 003\ 500$ bytes.
2. Transferir la relación EMPLEADO al sitio 2, ejecutar la reunión en ese sitio y enviar el resultado al sitio 3. El tamaño del resultado de la consulta es $40 * 10\ 000 = 400\ 000$ bytes, de modo que debemos transferir $400\ 000 + 1\ 000\ 000 = 1\ 400\ 000$ bytes.
3. Transferir la relación DEPARTAMENTO al sitio 1, ejecutar la reunión en ese sitio y enviar el resultado al sitio 3. En este caso tenemos que transferir $400\ 000 + 3500 = 403\ 500$ bytes.

Si minimizar la cantidad de transferencia de datos es nuestro criterio de optimización, deberemos elegir la estrategia 3. Consideremos ahora otra consulta C': "Para cada departamento, obtener el nombre del departamento y el de su gerente." Esto puede expresarse como sigue en el álgebra relacional:

C': $\pi_{\text{NOMBREP, NOMBREP, APELLIDO} \bowtie_{\text{NSSGTE=NSS}} (\text{DEPARTAMENTO} \bowtie_{\text{NSSGTE=NSS}} \text{EMPLEADO})}$

Una vez más, supongamos que la consulta se introduce en el sitio 3. Las mismas tres estrategias de ejecución de la consulta C se aplican a C', excepto que el resultado de C' incluye sólo 100 registros, si suponemos que todos los departamentos tienen un gerente:

1. Transferir las relaciones EMPLEADO y DEPARTAMENTO al sitio del resultado, y efectuar la reunión ahí. En este caso necesitamos transferir un total de $1\ 000\ 000 + 3500 = 1\ 003\ 500$ bytes.
2. Transferir la relación EMPLEADO al sitio 2, ejecutar la reunión en ese sitio y enviar el resultado al sitio 3. El tamaño del resultado de la consulta es $40 * 100 = 4000$ bytes, de modo que debemos transferir $4000 + 1\ 000\ 000 = 1\ 004\ 000$ bytes.
3. Transferir la relación DEPARTAMENTO al sitio 1, ejecutar la reunión en ese sitio y enviar el resultado al sitio 3. En este caso tenemos que transferir $4000 + 3500 = 7500$ bytes.

Una vez más escogeríamos la estrategia 3, en este caso por un margen abrumador sobre las estrategias 1 y 2. Las tres estrategias anteriores son las más obvias para el caso en que el sitio del resultado (sitio 3) no es ninguno de los sitios que contienen archivos implicados en la consulta (sitios 1 y 2). Sin embargo, supongamos que el sitio de resultado es el 2; entonces tendremos dos estrategias simples:

4. Transferir la relación EMPLEADO al sitio 2, ejecutar la consulta y presentar el resultado al usuario en ese mismo sitio. Aquí, necesitaremos transferir el mismo número de bytes —1 000 000— para C y para C'.
5. Transferir la relación DEPARTAMENTO al sitio 1, ejecutar la consulta en ese sitio y devolver el resultado al sitio 2. En este caso deberemos transferir $400\ 000 + 3500 = 403\ 500$ bytes para C y $4000 + 3500 = 7500$ bytes para C'.

Una estrategia más compleja, que a veces funciona mejor que estas estrategias simples, utiliza una operación llamada **semirreunión**. A continuación estudiaremos esta operación y analizaremos la ejecución distribuida con semirreuniones.

23.5.2 Procesamiento de consultas distribuidas por semirreunión

El procesamiento de consultas distribuidas mediante la operación de semirreunión se basa en la idea de reducir el número de tuplas de una relación antes de transferirla a otro sitio. Intuitivamente, la idea es enviar la *columna de reunión* de una relación R al sitio donde se encuentra la otra relación S; esta columna se reúne entonces con S. Después de ello, los atributos de reunión, junto con los atributos requeridos en el resultado, se extraen por proyección y se devuelven al sitio original donde se reúnen con R. Así pues, sólo se transfiere la columna de reunión de R en una dirección, y un subconjunto de S que no contenga tuplas que no intervengan en el resultado se transfiere en la otra dirección. Si sólo una pequeña fracción de las tuplas de S participan en la reunión, esto puede ser una solución bastante eficiente para minimizar la transferencia de datos.

Como ilustración, consideremos la siguiente estrategia para ejecutar C o C':

1. Proyectar los atributos de reunión de DEPARTAMENTO en el sitio 2 y transferirlos al sitio 1. En el caso de C, transferimos $F = \pi_{\text{NÚMEROD}}(\text{DEPARTAMENTO})$, cuyo tamaño es $4 * 100 = 400$ bytes; en el caso de C', transferimos $F' = \pi_{\text{NSSGTE}}(\text{DEPARTAMENTO})$, cuyo tamaño es $9 * 100 = 900$ bytes.

2. Reunir el archivo transferido con la relación EMPLEADO en el sitio 1 y transferir los atributos requeridos del archivo resultante al sitio 2. En el caso de C, transferimos $R = \pi_{\langle A, ND, NOMBRE, APELLIDO \rangle}(F \bowtie_{\langle NÚMERO=ND \rangle} EMPLEADO)$, cuyo tamaño es $34 * 10\ 000 = 340\ 000$ bytes; en el caso de C', transferimos $R' = \pi_{\langle NSSGTE, NOMBRE, APELLIDO \rangle}(F' \bowtie_{\langle NSSGTE=NS \rangle} EMPLEADO)$, cuyo tamaño es $39 * 100 = 3900$ bytes.
3. Ejecutar la consulta reuniendo el archivo transferido R o R' con DEPARTAMENTO, y presentar el resultado al usuario en el sitio 2.

Si utilizamos esta estrategia, transferiremos 340 400 bytes en el caso de C y 4800 en el de C'. Limitamos los atributos y las tuplas de EMPLEADO transmitidos al sitio 2 en el paso 2 a sólo los que se van a reunir efectivamente con una tupla DEPARTAMENTO en el paso 3. En el caso de la consulta C, esto incluyó todas las tuplas de EMPLEADO, por lo que no se logró una mejora significativa. En el caso de C', en cambio, sólo se necesitaron 100 de las 10 000 tuplas EMPLEADO.

La operación de semirreunión se inventó para formalizar esta estrategia. Una operación de **semirreunión** $R \bowtie_{A=B} S$, donde A y B son atributos de dominio compatible de R y S, respectivamente, produce el mismo resultado que la expresión del álgebra relacional $\pi_{\langle R \rangle}(R \bowtie_{A=B} S)$. En un entorno distribuido en el que R y S residen en diferentes sitios, la semirreunión casi siempre se implementa transfiriendo primero $F = \pi_{\langle B \rangle}(S)$ al sitio donde R reside y reuniendo después F y R, lo que corresponde a la estrategia que acabamos de ver.

Cabe señalar que la operación de semirreunión no es conmutativa; es decir:

$$R \bowtie S \neq S \bowtie R$$

23.5.3 Descomposición de consultas y de actualizaciones

En un SGBDD *sin transparencia de distribución*, el usuario expresa su consulta directamente en términos de fragmentos específicos. Por ejemplo, consideremos otra consulta C: "Obtener los nombres y las horas por semana de cada uno de los empleados que trabajan en algún proyecto controlado por el departamento 5", que se especifica sobre la base de datos distribuida donde las relaciones que están en los sitios 2 y 3 son las que se muestran en la figura 23.3 y las que están en el sitio 1 son las que se muestran en la figura 6.6. Un usuario que introduzca una consulta así deberá especificar si hace referencia a las relaciones PROYS5 y TRABAJA_EN5 del sitio 2 (Fig. 23.3) o a las relaciones PROYECTO y TRABAJA_EN del sitio 1 (Fig. 6.6). El usuario deberá mantener también la consistencia de los elementos de información replicados cuando actualice un SGBDD *sin transparencia de replicación*.

Por otro lado, un SGBDD que ofrezca *transparencia de distribución, de fragmentación y de replicación completa* permitirá al usuario especificar una consulta o solicitud de actualización sobre el esquema de la figura 6.5, igual que si el SGBD fuera centralizado. En el caso de las actualizaciones, el sistema se encarga de mantener la consistencia entre los elementos replicados usando alguno de los algoritmos de control de concurrencia distribuida que se analizan en la sección 23.6. En el caso de las consultas, un módulo de **descomposición de consultas** deberá dividir o **descomponer** una consulta en **subconsultas** que se puedan ejecutar en los sitios individuales. Por añadidura, deberá generarse una estrategia para combinar los resultados de las subconsultas y formar el resultado de la consulta. Siempre que el SGBDD determine que un elemento al que se hace referencia en la consulta está replicado, deberá escoger o **materializar** una réplica específica a la que se hará referencia durante la ejecución.

Para determinar cuáles réplicas incluyen los elementos de información a los que se hace referencia en una consulta, el SGBDD consulta la información de fragmentación, replicación y distribución almacenada en su catálogo. En la fragmentación vertical, la lista de atributos de cada fragmento se guarda en el catálogo. En la fragmentación horizontal, se guarda una condición, a veces llamada **de guardia**, para cada fragmento. Ésta es básicamente una condición de selección que especifica cuáles tuplas están en el fragmento; se le llama *guardia* porque sólo las tuplas que satisfacen esta condición pueden estar almacenadas en el fragmento. En el caso de fragmentos mixtos, tanto la lista de atributos como la condición de guardia se mantienen en el catálogo.

En nuestro ejemplo anterior, las condiciones de guardia de los fragmentos del sitio 1 (Fig. 6.6) son TRUE (todas las tuplas), y las listas de atributos son * (todos los atributos). Para los fragmentos que se muestran en la figura 23.3, tenemos las condiciones de guardia y listas de atributos que aparecen en la figura 23.6. Cuando el SGBDD descompone una solicitud de actualización, puede determinar cuáles fragmentos deben actualizarse, para lo cual

(a) EMPD5	(b) EMPD4
lista de atributos: NOMBRE, INIC, APELLIDO, NSS, SALARIO, NSSUPER, ND	lista de atributos: NOMBRE, INIC, APELLIDO, NSS, SALARIO, NSSUPER, ND
condición de guardia: ND=5	condición de guardia: ND=4
DEP5	DEP4
lista de atributos: * (todos los atributos NOMBRE, NÚMEROD, NSSGTE, FECHAINICGTE)	lista de atributos: * (todos los atributos NOMBRE, NÚMEROD, NSSGTE, FECHAINICGTE)
condición de guardia: NÚMEROD=5	condición de guardia: NÚMEROD=4
LUGS_DEP5	LUGS_DEP4
lista de atributos: * (todos los atributos NÚMEROD, LUGAR)	lista de atributos: * (todos los atributos NÚMEROD, LUGAR)
condición de guardia: NÚMEROD=5	condición de guardia: NÚMEROD=4
PROYS5	PROYS4
lista de atributos: * (todos los atributos NOMBRE, NÚMEROP, LUGAR, NÚMD)	lista de atributos: * (todos los atributos NOMBRE, NÚMEROP, LUGAR, NÚMD)
condición de guardia: NÚMD=5	condición de guardia: NÚMD=4
TRABAJA_EN5	TRABAJA_EN4
lista de atributos: * (todos los atributos NSSE, NÚMP, HORAS)	lista de atributos: * (todos los atributos NSSE, NÚMP, HORAS)
condición de guardia: NSSE IN ($\pi_{\langle NSS \rangle}(EMPD5)$)	condición de guardia: NSSE IN ($\pi_{\langle NSS \rangle}(EMPD4)$)
OR NÚMP IN ($\pi_{\langle NÚMEROP \rangle}(PROYS5)$)	OR NÚMP IN ($\pi_{\langle NÚMEROP \rangle}(PROYS4)$)

Figura 23.6 Condiciones de guardia y listas de atributos para fragmentos. (a) Guardias y listas de atributos para los fragmentos de relaciones del sitio 2. (b) Guardias y listas de atributos para los fragmentos de relaciones del sitio 3.

examina sus condiciones de guardia. Por ejemplo, el SGBDD descompondría una solicitud de insertar una nueva tupla EMPLEADO <'Alejandro', 'B', 'Colmenares', '345671239', '22-ABR-64', '3306 Rosedal, Higueras, MX', 'M', '33000', '987654321', 4> en dos solicitudes de actualización: la primera insertaría la tupla anterior en el fragmento EMPLEADO del sitio 1 y la segunda insertaría la tupla proyectada <'Alejandro', 'B', 'Colmenares', '345671239', '33000', '987654321', 4> en el fragmento EMPD4 del sitio 3.

Para la descomposición de consultas, el SGBDD puede determinar cuáles fragmentos contienen las tuplas requeridas comparando la condición de la consulta y las condiciones de guardia. Por ejemplo, consideremos la consulta C: "Obtener los nombres y las horas por semana de cada uno de los empleados que trabajan en algún proyecto controlado por el departamento 5"; esto puede especificarse en SQL sobre el esquema de la figura 6.5 como sigue:

```
C:  SELECT  NOMBREP, APELLIDO, HORAS
      FROM    EMPLEADO, PROYECTO, TRABAJA_EN
      WHERE   NÚMD=5 AND NÚMEROP=NÚMP AND NSSE=NSS
```

Supongamos que la consulta se introduce en el sitio 2, y ahí mismo se va a necesitar el resultado de la consulta. El SGBDD puede determinar, a partir de la condición de guardia de PROYS5 y de TRABAJA_EN5, que todas las tuplas que satisfacen las condiciones (NÚMD = 5 AND NÚMEROP = NÚMP) residen en el sitio 2; por tanto, puede descomponer la consulta en las siguientes subconsultas del álgebra relacional:

```
T1 ← πNSSE (PROYS5 ⋈NÚMEROP=NÚMP TRABAJA_EN5)
T2 ← πNSSE, NOMBREP, APELLIDO (T1 ⋈NSSE=NSS EMPLEADO)
RESULTADO ← πNOMBREP, APELLIDO, HORAS (T2 * TRABAJA_EN5)
```

Esta descomposición puede servir para ejecutar la consulta mediante una estrategia de semirreunión. El SGBDD sabe, por las condiciones de guardia, que PROYS5 contiene exactamente las tuplas que satisfacen (NÚMD = 5) y que TRABAJA_EN5 contiene todas las tuplas que se reunirán con PROYS5; por tanto, la subconsulta T1 puede ejecutarse en el sitio 2, y la columna proyectada NSSE se puede enviar al sitio 1. La subconsulta T2 puede entonces ejecutarse en el sitio 1, y el resultado se puede enviar de vuelta al sitio 2, donde se calcula el resultado final de la consulta y se presenta al usuario. Una estrategia alternativa consistiría en enviar la consulta C completa al sitio 1, que contiene todas las tuplas de la base de datos, donde se ejecutaría localmente y desde donde se enviaría el resultado de registro al sitio 2. El optimizador de consultas estimaría los costos de ambas estrategias y escogería la que tuviera la estimación más baja.

23.6 Panorama sobre el control de concurrencia y la recuperación en bases de datos distribuidas*

En lo tocante al control de concurrencia y la recuperación en un entorno de SGBD distribuido, surgen numerosos problemas que no se encuentran en los entornos de SGBD centralizado. Entre ellos están:

- **Manejar múltiples copias de los elementos de información:** El método de control de concurrencia tiene la obligación de mantener la consistencia entre estas copias. El

método de recuperación debe cuidar que una copia sea consistente con todas las demás si el sitio en el que la copia estaba almacenada falla y se recupera posteriormente.

- **Fallo de sitios individuales:** El SGBDD debe continuar operando con sus sitios activos, si es posible, cuando fallen uno o más sitios individuales. Cuando un sitio se recupere, su base de datos local se deberá poner al día con los demás sitios antes de que se reincorpore al sistema.
- **Fallo de enlaces de comunicaciones:** El sistema debe ser capaz de manejar el fallo de uno o más de los enlaces de comunicaciones entre los sitios. Un caso extremo de este problema es que puede haber **partición de la red**. Esto divide los sitios en dos o más particiones dentro de las cuales los sitios pueden comunicarse entre sí, pero no con sitios de otras particiones.
- **Confirmación distribuida:** Puede haber problemas para confirmar una transacción que está teniendo acceso a bases de datos almacenadas en múltiples sitios si algunos de éstos fallan durante el proceso de confirmación. A menudo se utiliza el **protocolo de confirmación de dos fases** (véase la Sec. 19.5) para resolver este problema.
- **Bloqueo mortal distribuido:** Puede ocurrir un bloqueo mortal entre varios sitios, lo que hace necesario extender las técnicas de manejo de bloqueos mortales para que tengan esto en cuenta.

Las técnicas de control de concurrencia y de recuperación distribuidos deben resolver éstos y otros problemas. En las subsecciones que siguen repasaremos algunas de las técnicas que se han propuesto para manejar la recuperación y el control de concurrencia en los SGBDD.

23.6.1 Control de concurrencia distribuido basado en una copia distinguida de un elemento de información

Se han diseñado varios métodos de control de concurrencia que manejan los elementos de información replicados en una base de datos distribuida, que se basan en la extensión de las técnicas de control de concurrencia de las bases de datos centralizadas. Analizaremos estas técnicas en el contexto de extender el **bloqueo** centralizado. Extensiones similares se aplican a otras técnicas de control de concurrencia. La idea es designar una **copia determinada** de cada elemento de información como **copia distinguida**. Los candados para este elemento de información se asocian a la **copia distinguida**, y todas las solicitudes de bloqueo y desbloqueo se envían al sitio que contiene esa copia.

Hay varios métodos diferentes que se basan en esta idea, pero difieren en la forma en que escogen las copias distinguidas. En la técnica de **sitio primario**, todas las copias distinguidas se guardan en el mismo sitio. Una modificación de este enfoque es el sitio **primario con sitio de respaldo**. Otro enfoque es el método de **copia primaria**, en el que las copias distinguidas de los diversos elementos de información se pueden almacenar en diferentes sitios. Un sitio que incluye una copia distinguida de un elemento de información actúa básicamente como **sitio coordinador** para el control de concurrencia de ese elemento. Analizaremos estas técnicas una por una.

Técnica de sitio primario. En este método se designa un solo **sitio primario** como **sitio coordinador para todos los elementos de la base de datos**. Por tanto, todos los candados se mantienen en ese sitio, y todas las solicitudes de bloqueo y desbloqueo se envían a ese sitio. Así pues, este método es una extensión del enfoque de bloqueo centralizado. Por ejemplo, si todas las transacciones siguen el protocolo de bloqueo de dos fases, la seriabilidad está garantizada. La ventaja de este enfoque es que es una simple extensión del enfoque centralizado y por tanto no es demasiado complejo. Sin embargo, posee ciertas desventajas inherentes. Una de ellas es que todas las solicitudes de bloqueo se envían a un mismo sitio, con lo que posiblemente se sobrecargue ese sitio y se origine un cuello de botella del sistema. Otra desventaja es que un fallo del sitio primario paralizaría el sistema, ya que toda la información de bloqueo se mantiene en dicho sitio. Esto puede limitar la fiabilidad y la disponibilidad del sistema.

Aunque el acceso a todos los candados es en el sitio primario, el acceso a los elementos mismos puede darse en cualquier sitio en el que residan. Por ejemplo, una vez que una transacción obtiene del sitio primario un bloquear_lectura sobre un elemento de información, puede tener acceso a cualquier copia de dicho elemento. Sin embargo, una vez que una transacción obtiene un bloquear_escritura y actualiza un elemento de información, el SGBDD deberá encargarse de actualizar *todas las copias* del elemento antes de liberar el candado.

Sitio primario con sitio de respaldo. Este enfoque busca subsanar la segunda desventaja del método anterior; para ello se designa un segundo sitio como **sitio de respaldo**. Toda la información de bloqueo se mantiene tanto en el sitio primario como en el de respaldo. En caso de fallar el sitio primario, el de respaldo puede asumir las funciones de sitio primario, y se puede escoger un nuevo sitio de respaldo. Esto simplifica el proceso de recuperarse de un fallo del sitio primario, ya que el sitio de respaldo asume sus funciones y el procesamiento puede reanudarse una vez que se elija el nuevo sitio de respaldo y la información de estado de los candados se copia en él. Sin embargo, el proceso de adquisición de candados se hace más lento, porque todas las solicitudes de candados y la concesión de los mismos se deben asentar *tanto en el sitio primario como en el de respaldo* antes de enviar una respuesta a la transacción solicitante. El problema de que los sitios primario y de respaldo se sobrecarguen con solicitudes y hagan más lento el sistema no se alivia en absoluto.

Técnica de copia primaria. Este método intenta distribuir la carga de la coordinación de los candados entre varios sitios; lo hace manteniendo las copias distinguidas de diferentes elementos de información *almacenadas en diferentes sitios*. El fallo de un sitio afecta todas las transacciones que estén teniendo acceso a candados sobre elementos cuyas copias primarias residan en ese sitio, pero las demás transacciones no resultan afectadas. Este método también puede usar sitios de respaldo para elevar la fiabilidad y la disponibilidad.

Elección de un nuevo sitio coordinador en caso de fallo. Siempre que un sitio coordinador falle en cualquiera de las técnicas anteriores, los sitios que siguen activos deberán elegir un nuevo coordinador. En el caso del enfoque de sitio primario sin sitio de respaldo, será preciso abortar y reiniciar todas las transacciones en ejecución, y el proceso de recuperación será bastante tedioso. Parte de dicho proceso implica elegir un nuevo sitio primario y crear un proceso gestor de candados y un registro de toda la información de candados en ese sitio. En los métodos que usan sitios de respaldo, el procesamiento de transacciones se suspende mientras el sitio de respaldo se designa como nuevo sitio primario, se escoge un nuevo sitio de respaldo y se envían a él copias de toda la información de bloqueo del nuevo sitio primario.

Si el sitio de respaldo *X* está a punto de convertirse en el nuevo sitio primario, *X* puede escoger el nuevo sitio de respaldo entre los sitios activos del sistema. Sin embargo, si no había sitio de respaldo, o si no están activos ni el sitio primario ni el de respaldo, se puede seguir un proceso denominado **elección para escoger el nuevo sitio coordinador**. En este proceso, cualquier sitio *Y* que intente comunicarse repetidamente con el sitio coordinador y fracase puede suponer que el coordinador está inactivo e iniciar el proceso de elección enviando un mensaje a todos los sitios activos en el cual proponga que *Y* se convierta en el nuevo coordinador. Tan pronto como *Y* reciba una mayoría de votos afirmativos, puede declarar que es el nuevo coordinador. El algoritmo de elección mismo es bastante complejo, pero ésta es a grandes rasgos la idea básica. El algoritmo resuelve además cualquier intento, por parte de dos o más sitios, de convertirse al mismo tiempo en el coordinador. Las referencias en la bibliografía al final del capítulo analizan este proceso en detalle.

23.6.2 Control de concurrencia distribuido basado en votación

Todos los métodos de control de concurrencia para elementos replicados que acabamos de analizar se basan en la idea de una copia distinguida que mantiene los candados para ese elemento. En el **método de votación** no hay copia distinguida; más bien, cada solicitud de bloqueo se envía a todos los sitios que incluyan una copia del elemento de información. Cada copia mantiene su propio candado y puede conceder o rechazar la solicitud. Si la mayoría de las copias otorgan un candado a la transacción que lo solicita, ésta poseerá el candado e informará a *todas las copias* que le ha sido concedido. Si una transacción no recibe la mayoría de los votos de concesión del candado durante un cierto *periodo de tiempo predefinido*, cancelará su solicitud e informará de ello a todos los sitios.

El método de votación se considera un método de control de concurrencia verdaderamente distribuido, ya que la responsabilidad de la decisión recae en todos los sitios implicados. Se han hecho estudios de simulación que indican que la votación tiene un tráfico más alto de mensajes entre sitios que los métodos de copia distinguida. Si el algoritmo tiene en cuenta posibles fallos de sitios durante el proceso de votación, se vuelve extremadamente complejo.

23.6.3 Recuperación distribuida

El proceso de recuperación en las bases de datos distribuidas es bastante complicado. Aquí daremos sólo una idea de algunos de sus aspectos. En ciertos casos incluso es bastante difícil determinar si un sitio está inactivo sin intercambiar un gran número de mensajes con otros sitios. Por ejemplo, supongamos que el sitio *X* envía un mensaje al sitio *Y* y espera una respuesta de *Y* pero no la recibe. Hay varias explicaciones posibles:

- El mensaje no llegó a *Y* debido a un fallo de comunicación.
- El sitio *Y* está inactivo y no pudo responder.
- El sitio *Y* está activo y envió una respuesta, pero ésta no llegó.

Sin información adicional y sin el envío de mensajes adicionales, es difícil determinar qué sucedió realmente.

Otro problema con la recuperación distribuida es la confirmación distribuida. Cuando una transacción está actualizando datos en varios sitios, no puede confirmarse hasta estar

segura de que el efecto de la transacción en todos los sitios no puede perderse. Esto significa que cada sitio debe haber asentado primero permanentemente los efectos locales de la transacción en la bitácora local en el disco del sitio. A menudo se usa el protocolo de confirmación de dos fases, analizado en la sección 19.5, para garantizar la corrección de la confirmación distribuida.

23.7 Resumen

Este capítulo presenta una introducción a las bases de datos distribuidas. Como se trata de un tema muy amplio, sólo analizamos algunas de sus técnicas fundamentales. Primero vimos por qué es conveniente distribuir y cuáles son las ventajas potenciales de las bases de datos distribuidas respecto a los sistemas centralizados. Después describimos la arquitectura cliente-servidor de un SGBDD. Definimos los conceptos de fragmentación, replicación y distribución de los datos, y distinguimos entre los fragmentos horizontales y verticales de las relaciones. Analizamos el empleo de la replicación de datos para elevar la fiabilidad y la disponibilidad del sistema. También definimos el concepto de transparencia de la distribución y los conceptos relacionados de transparencia de la fragmentación y transparencia de la replicación. Clasificamos los SGBDD según criterios como el grado de homogeneidad de los módulos de software y el grado de autonomía local.

En la sección 23.5 ilustramos algunas de las técnicas para procesar consultas distribuidas. El costo de la comunicación entre los sitios se considera como un factor primordial en la optimización de consultas distribuidas. Comparamos diferentes técnicas para ejecutar reuniones y presentamos la técnica de semirreunión para reunir relaciones residentes en diferentes sitios.

En la sección 23.6 examinamos brevemente las técnicas de control de concurrencia y de recuperación empleadas en los SGBDD. Mencionamos algunos de los problemas adicionales que es preciso resolver en los entornos distribuidos y que no aparecen en un entorno centralizado.

Preguntas de repaso

- 23.1. ¿Cuáles son las principales razones para tener bases de datos distribuidas, y cuáles las posibles ventajas?
- 23.2. ¿Qué funciones adicionales efectúa un SGBDD que no realiza un SGBD centralizado?
- 23.3. ¿Cuáles son los módulos de software más importantes de un SGBDD? Analice las funciones principales de cada uno de estos módulos en el contexto de la arquitectura cliente-servidor.
- 23.4. ¿Qué es un fragmento de una relación? ¿Cuáles son los principales tipos de fragmentos? ¿Por qué la fragmentación es un concepto útil en el diseño de bases de datos distribuidas?
- 23.5. ¿Por qué es útil la replicación en los SGBDD? ¿Qué unidades de datos son las que se suelen replicar?
- 23.6. ¿Qué significa *reparto de los datos* en el diseño de bases de datos distribuidas? ¿Cuáles unidades de datos suelen distribuirse entre los sitios?

- 23.7. ¿Cómo se especifica una fragmentación horizontal de una relación? ¿Cómo puede reconstruirse una relación a partir de una fragmentación horizontal completa?
- 23.8. ¿Cómo se especifica una fragmentación vertical de una relación? ¿Cómo puede reconstruirse una relación a partir de una fragmentación vertical completa?
- 23.9. Explique el significado de los siguientes términos: *grado de homogeneidad de un SGBDD*, *grado de autonomía local de un SGBDD*, *SGBDD federado*, *transparencia de distribución*, *transparencia de fragmentación*, *transparencia de replicación*, *sistema de multibases de datos*.
- 23.10. Analice el problema de los nombres en las bases de datos distribuidas.
- 23.11. Explique las diferentes técnicas para ejecutar una equirreunión de dos archivos ubicados en diferentes sitios. ¿Qué factores locales son los que más afectan el costo de la transferencia de datos?
- 23.12. Analice el método de semirreunión para ejecutar una equirreunión de dos archivos ubicados en diferentes sitios. ¿En qué condiciones resulta eficiente una estrategia de semirreunión?
- 23.13. Analice los factores que afectan la descomposición de consultas. ¿Cómo se usan las condiciones de guardia y las listas de atributos de fragmentos durante el proceso de descomposición de consultas?
- 23.14. ¿Qué diferencia hay entre la descomposición de una solicitud de actualización y la descomposición de una consulta? ¿Cómo se usan las condiciones de guardia y las listas de atributos de los fragmentos durante la descomposición de una solicitud de actualización?
- 23.15. Analice los factores que no aparecen en los sistemas centralizados y que afectan el control de concurrencia y la recuperación en los sistemas distribuidos.
- 23.16. Compare el método de sitio primario con el de copia primaria para el control de concurrencia distribuido. ¿Cómo afecta a cada uno de ellos el empleo de sitios de respaldo?
- 23.17. ¿Cuándo se usan la votación y las elecciones en las bases de datos distribuidas?

Ejercicios

- 22.18. Considere la distribución de datos de la base de datos COMPANÍA, según la cual los fragmentos que están en los sitios 2 y 3 son los que se muestran en la figura 23.3 y los fragmentos del sitio 1 son los que se muestran en la figura 6.6. Para cada una de las siguientes consultas, indique por lo menos dos estrategias de descomposición y ejecución de la consulta. ¿En qué condiciones funcionaría bien cada una de sus estrategias?
 - a. Para cada empleado del departamento 5, obtener su nombre y los nombres de sus dependientes.
 - b. Imprimir los nombres de todos los empleados adscritos al departamento 5 que trabajan en algún proyecto que no está bajo el control del departamento 5.

Bibliografía selecta

Los libros de Ceri y Pelagatti (1984a) y Ozsu y Valduriez (1990) se ocupan de las bases de datos distribuidas. La fragmentación se analiza en Chang y Cheng (1980) y Ceri y Pelagatti (1984a). Los sistemas de bases de datos federadas se tratan en McLeod y Heimbigner (1985). Los principios de las redes de comunicaciones entre computadores se estudian en el libro de Tanenbaum (1981).

El procesamiento, la optimización y la descomposición de consultas distribuidas se tratan en los artículos de Hevner y Yao (1979), Apers *et al.* (1983), Ceri y Pelagatti (1984), Bodorick *et al.* (1992) y Kerschberg *et al.* (1982). Bernstein y Goodman (1981) analiza la teoría del procesamiento por semirreunión. Wong (1983) se ocupa del empleo de vínculos en la fragmentación de relaciones. Los esquemas de control de concurrencia y de recuperación se analizan y comparan en García-Molina (1978), Ries (1979), y Bernstein y Goodman (1981a). Las elecciones en los sistemas distribuidos se examinan en García-Molina (1982).

Una técnica de control de concurrencia para datos replicados que se basa en la votación se presenta en Thomas (1979). Gifford (1979) propone el empleo de votación ponderada, y Paris (1986) describe un método llamado votación con testigos. Jajodia y Mutchler (1990) explica la votación dinámica. Bernstein y Goodman (1984) propone una técnica llamada *copia disponible*, y una que utiliza la idea de grupo se presenta en ElAbbadi y Toueg (1988). Otros trabajos recientes que tratan los datos replicados son los de Gladney (1989), Agrawal y ElAbbadi (1990), ElAbbadi y Toueg (1990), Kumar y Segev (1993), Mukkamala (1989) y Wolfson y Milo (1991). Schlageter (1981), Bassiouni (1988) y Ceri y Owicki (1983) analizan métodos optimistas para el control de concurrencia de BDD. García-Molina (1983) y Kumar y Stonebraker (1987) explican una técnica que aprovecha la semántica de las transacciones. Menasce *et al.* (1980) y Minoura y Wiederhold (1982) presentan técnicas de control de concurrencia distribuido basadas en bloqueo y copias distribuidas. Obermark (1982) presenta algoritmos para la detección de bloqueos mortales distribuidos. Lamport (1978) estudia los problemas de generar marcas de tiempo únicas en un sistema distribuido.

Kohler (1981) hace un repaso de las técnicas de recuperación en los sistemas distribuidos. Skeen (1981) estudia protocolos de confirmación sin bloqueo, y Reed (1983) trata las acciones atómicas sobre datos distribuidos. Un libro editado por Bhargava (1987) presenta diversos enfoques y técnicas para la recuperación y el control de concurrencia distribuidos.

Elmasri *et al.* (1986), Hayne y Ram (1990) y Motro (1987) presentan técnicas para la integración de esquemas en bases de datos federadas. Onuegbue *et al.* (1983) analiza el problema de reducción y optimización local. Elmagarmid y Helal (1988) y Gamal-Eldin *et al.* (1988) tratan el problema de la actualización en SGBDD heterogéneos. Las bases de datos distribuidas heterogéneas también se estudian en Kaul *et al.* (1990), Hsiao y Kamel (1989) y Perrizo *et al.* (1991).

Las técnicas para el reparto de datos distribuidos se propusieron en Blankenship *et al.* (1991), Morgan y Levin (1977), Ramamoorthy y Wöh (1979), Chu y Hurley (1982), y Kamel y King (1985), entre otros. El diseño de bases de datos distribuidas con fragmentación horizontal y vertical, reparto y replicación se trata en Ceri *et al.* (1983), Navathe *et al.* (1984), Ceri *et al.* (1982), Wilson y Navathe (1986) y Elmasri *et al.* (1987).

En fechas recientes, los sistemas de multibases de datos y la interoperabilidad son temas que han adquirido importancia. Las técnicas para resolver incompatibilidades semánticas entre múltiples bases de datos se examinan en DeMichiel (1989), Siegel y Madnick (1991), Krishnamurthy *et al.* (1991) y Wang y Madnick (1989). El procesamiento de transacciones en multibases de datos se analiza en Mehrotra *et al.* (1992), Georgakopoulos *et al.* (1991), Elmagarmid *et al.* (1990) y Brietbart *et al.* (1990), entre otros.

Se han implementado varios SGBD distribuidos experimentales. Entre ellos están INGRES distribuido (Epstein *et al.* 1978), DOTS (Devor y Weeldreyer 1980), SDD-1 (Rothnie *et al.* 1980), System R* (Lindsay *et al.* 1984), SIRIUS-DELTA (Fertner y Stangret 1982) y MULTIBASE (Smith *et al.* 1981). El

sistema OMNIBASE (Rusinkiewicz *et al.* 1988) es un SGBDD federado. Muchos proveedores de SGBD comerciales han anunciado versiones distribuidas de sus sistemas. Los SGBDD comerciales están aplicando la arquitectura cliente-servidor en sus versiones más nuevas. Algunas cuestiones relativas a los sistemas cliente-servidor se estudian en Carey *et al.* (1991), DeWitt *et al.* (1990) y Wang y Rowe (1991).

CAPÍTULO 24

Bases de datos deductivas

24.1 Introducción a las bases de datos deductivas

En un sistema de bases de datos deductivas por lo regular se usa un lenguaje declarativo para especificar reglas. Con **lenguaje declarativo** queremos decir un lenguaje que define lo que un programa desea lograr, en vez de especificar los detalles de cómo lograrlo. Una **máquina de inferencia** (o **mecanismo de deducción**) dentro del sistema puede deducir hechos nuevos a partir de la base de datos interpretando dichas reglas. El modelo empleado en las bases de datos deductivas está íntimamente relacionado con el modelo de datos relacional, y sobre todo con el formalismo del cálculo relacional (véase el Cap. 8). También está relacionado con el campo de la **programación lógica** y el lenguaje **Prolog**. Los trabajos sobre bases de datos deductivas basadas en lógica han utilizado Prolog como punto de partida. Con un subconjunto de Prolog llamado **Datalog** se definen reglas declarativamente junto con un conjunto de relaciones existentes que se tratan como literales en el lenguaje. Aunque la estructura gramatical de Datalog se parece a la de Prolog, su semántica operativa —esto es, la forma como debe ejecutarse un programa en Datalog— queda abierta.

Una base de datos deductiva utiliza principalmente dos tipos de especificaciones: hechos y reglas. Los hechos se especifican de manera similar a como se especifican las relaciones, excepto que no es necesario incluir los nombres de los atributos. Recuerde que una tupla en una relación describe algún hecho del mundo real cuyo significado queda determinado en parte por los nombres de los atributos. En una base de datos deductiva, el significado del valor de un atributo en una tupla queda determinado exclusivamente por su posición dentro de la tupla. Las reglas se parecen un poco a las vistas relacionales. Especifican relaciones virtuales que no están almacenadas realmente, pero que se pueden formar a partir de los hechos aplicando mecanismos de inferencia basados en las especificaciones de las reglas. La principal diferencia entre las reglas y las vistas es que en las primeras puede haber recursión y por tanto pueden producir vistas que no es posible definir en términos de las vistas relacionales estándar.

La evaluación de los programas en Prolog se basa en una técnica llamada *encadenamiento hacia atrás*, que implica una evaluación descendente de los objetivos. En las bases de datos deductivas que emplean Datalog, el enfoque es hacia el manejo de grandes volúmenes de datos almacenados en una base de datos relacional; por ello, se han inventado técnicas de evaluación que se parecen a las de una evaluación ascendente. Prolog tiene la limitación de que el orden de especificación de los hechos y las reglas es significativo para la evaluación; es más, el orden de las literales dentro de una regla es significativo. Las técnicas de ejecución para los programas en Datalog intentan subsanar estos problemas.

Ya estudiamos las bases de datos orientadas a objetos (BDOO) en el capítulo 22. Es conveniente poner las bases de datos deductivas (BDD) en un contexto apropiado respecto a las BDOO. Estas últimas han procurado proveer un mecanismo de modelado natural para los objetos del mundo real encapsulando su estructura junto con su comportamiento. En contraste, las BDD buscan derivar nuevos conocimientos a partir de datos existentes proporcionando interrelaciones del mundo real en forma de reglas. Tradicionalmente, las BDOO han dejado al programador la tarea de optimizar las consultas de búsqueda, en tanto que las BDD utilizan mecanismos internos para la evaluación y la optimización. Han comenzado a aparecer indicios de un casamiento entre estas dos diferentes mejoras de las bases de datos tradicionales, que se ha hecho patente en la adición de capacidades deductivas a las BDOO y de interfaces con lenguajes de programación como C++ a las BDD.

El resto del capítulo está organizado como sigue. En la sección 24.2 presentaremos la notación Prolog/Datalog; Datalog es un lenguaje de consulta deductivo similar a Prolog,

Un sistema de bases de datos que tenga la capacidad de definir reglas con las cuales deducir o inferir información adicional a partir de los hechos almacenados en las bases de datos se llama sistema de bases de datos deductivas. Puesto que parte de los fundamentos teóricos de algunos sistemas de esta especie es la lógica matemática, a menudo se les denomina **bases de datos lógicas**. También existen los llamados **sistemas de bases de datos expertos** y **sistemas basados en conocimientos** que además incluyen capacidades de razonamiento e inferencia; estos sistemas emplean técnicas que se desarrollaron en el campo de la inteligencia artificial, como las redes semánticas, los marcos y las reglas para capturar conocimientos específicos de un dominio. Son dos las diferencias principales entre estos sistemas y los que estudiaremos aquí:

1. Los sistemas expertos basados en conocimientos han supuesto tradicionalmente que los datos requeridos residen en la memoria principal, por lo que la gestión de almacenamiento secundario no viene al caso. Los sistemas de bases de datos deductivas intentan modificar esta restricción de modo que un SODB se amplíe para manejar una interfaz con un sistema experto o que un sistema experto se amplíe para manejar datos que residen en almacenamiento secundario.
2. Los conocimientos de un sistema experto o basado en conocimientos se extraen de expertos en la aplicación y se refieren a un dominio de aplicación más que a conocimientos inherentes en los datos.

Aunque existe una estrecha relación entre estos sistemas y las bases de datos deductivas que vamos a analizar, su análisis detallado rebasa el alcance de esta obra. En este capítulo sólo hablaremos de los sistemas basados en lógica y ofreceremos un panorama sobre los fundamentos formales de los sistemas de bases de datos deductivas.

Si sólo desea una breve introducción a las bases de datos deductivas, el lector puede limitarse a las secciones 24.1 y 24.2.

pero más adecuado para aplicaciones de bases de datos. En la sección 24.3 trataremos interpretaciones teóricas del significado de las reglas. Los dos enfoques estándar de los mecanismos de inferencia en los lenguajes de programación lógica, llamados *encadenamiento hacia adelante* y *encadenamiento hacia atrás*, se examinan en la sección 24.4. La sección 24.5 presenta conceptos relacionados con los programas en Datalog, su evaluación y su ejecución. En la sección 24.6 se analiza un sistema comercial llamado sistema de bases de datos deductivos IDI; después, en la sección 24.7, cubriremos brevemente dos sistemas de investigación universitaria denominados CORAL y NAIL!

24.2 Notación Prolog/Datalog

En esta sección presentaremos la notación básica para escribir reglas en Prolog/Datalog. Luego analizaremos las dos principales interpretaciones teóricas del significado de las reglas en la sección 24.3. La notación de Prolog/Datalog se basa en proveer **predicados** con nombres únicos. Un predicado tiene un significado implícito, sugerido por su nombre, y un número fijo de **argumentos**. Si todos los argumentos son valores constantes, el predicado simplemente dice que un determinado hecho es verdadero. Por otro lado, si el predicado tiene variables como argumentos, se le considera una consulta o bien parte de una regla o restricción. En todo este capítulo adoptaremos la convención de Prolog de que todos los **valores constantes** en un predicado son *numéricos* o bien son cadenas de caracteres que comienzan exclusivamente con *letras minúsculas*, en tanto que los **nombres de variables** siempre comienzan con una *letra mayúscula*.

24.2.1 Un ejemplo

Consideremos el ejemplo que se muestra en la figura 24.1, el cual se refiere a la base de datos relacional de la figura 6.6, pero en forma muy simplificada. Hay tres nombres de predicados: *superisar*, *superior* y *subordinado*. El predicado *superisar* se define mediante un conjunto de hechos, cada uno de los cuales tiene dos argumentos: un nombre de supervisor, seguido del nombre de un supervisado *directo* (subordinado) de ese supervisor. Estos hechos corresponden a los datos reales almacenados en la base de datos, y puede considerarse que constituyen un conjunto de tuplas de una relación SUPERISAR con dos atributos, cuyo esquema es:

SUPERISAR (Supervisor,Supervisado)

Así pues, *superisar(X,Y)* expresa el hecho de que “X supervisa a Y”. Observe la omisión de los nombres de atributos en la notación Prolog. Estos nombres se representan exclusivamente en virtud de la posición de cada argumento en un predicado: el primer argumento representa el supervisor, y el segundo, un subordinado directo.

Los otros dos nombres de predicados se definen mediante reglas. La contribución principal de las bases de datos deductivas es la capacidad de especificar reglas recursivas y de proporcionar un marco de referencia para inferir información nueva con base en las reglas especificadas. Una regla tiene la forma **cabeza** :- **cuerpo** y por lo regular tiene un **solo predicado** a la izquierda del símbolo :- (llamado **cabeza**, o miembro izquierdo (LHS: *left-hand side*), o conclusión de la regla) y **uno o más predicados** a la derecha del símbolo :- (llamados **cuerpo**, o miembro derecho (RHS: *right-hand side*), o premisa(s) de la regla). Decimos

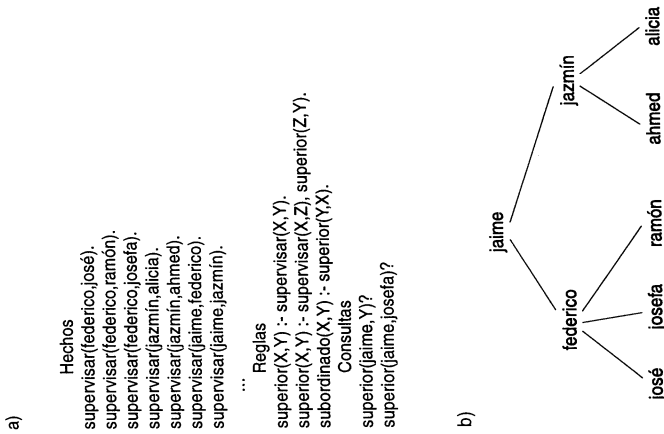


Figura 24.1 (a) Notación de Prolog para hechos, reglas y consultas. (b) El árbol de supervisión basado en los hechos dados.

que un predicado cuyos argumentos son constantes es un predicado **base**; también nos referimos a él como **predicado de ejemplares**. Los argumentos de los predicados que aparecen en una regla por lo regular incluyen un cierto número de símbolos variables, aunque los predicados también contienen constantes como argumentos. Una regla especifica que, si una asignación o **enlace** particular de valores constantes a las variables del cuerpo (predicados RHS) hace que *todos* los predicados RHS sean **verdaderos**, también hace que la cabeza (predicado LHS) sea verdadera usando la misma asignación de valores constantes a las variables. Por tanto, una regla nos ofrece una forma de generar hechos nuevos que son ejemplares de la cabeza de la regla. Estos hechos nuevos se basan en hechos que ya existen y que corresponden a los ejemplares de los predicados del cuerpo de la regla.

Observe que, al incluir múltiples predicados en el cuerpo de una regla, aplicamos implícitamente el operador **and lógico** a esos predicados. Así, las comas entre los predicados RHS pueden leerse como conjunciones.

Consideremos la definición del predicado superior en la figura 24.1, cuyo primer argumento es un nombre de empleado y cuyo segundo argumento es un empleado que es un subordinado *directo* o *indirecto* del primer empleado. Con *subordinado indirecto* queremos decir el subordinado de algún subordinado, hasta cualquier número de niveles. Así, decir el subordinado de algún subordinado, hasta cualquier número de niveles. Así, superior(X,Y) representa el hecho de que “X es un superior de Y” a través de supervisión directa o indirecta. Podemos escribir dos reglas que en conjunto especifican el significado

del nuevo predicado. La primera regla bajo el título Reglas en la figura expresa que, para todo valor de X y Y , si se cumple supervisar(X, Y) —el cuerpo de la regla— también se cumple superior(X, Y) —la cabeza de la regla—, ya que Y sería un subordinado directo de X (un nivel más abajo). Esta regla puede servir para generar todos los vínculos directos superior/subordinado a partir de los hechos que definen el predicado supervisar. La segunda regla recursiva expresa que, si se cumplen tanto supervisar(X, Z) como superior(Z, Y), entonces también se cumple superior(X, Y). Esto es un ejemplo de **regla recursiva**, en la que uno de los predicados del cuerpo de la regla es también el predicado de la cabeza de la regla. En general, el cuerpo de la regla define varias premisas tales que, si todas se cumplen, podemos deducir que también se cumple la conclusión de la cabeza de la regla. Advertirá que si tenemos dos (o más) reglas con la misma cabeza (predicado LHS), esto equivale a decir que el predicado se cumple (esto es, que puede tener un ejemplar) si *cualquiera* de los cuerpos se cumple; y, por tanto, equivale a una operación **OR** lógico. Por ejemplo, si tenemos dos reglas $X \vdash Y$ y $X \vdash Z$, equivalen a una regla $X \vdash Y \vee Z$. Sin embargo, esta última forma no se usa en los sistemas deductivos porque no adopta la forma estándar de una regla, la llamada cláusula de Horn, como veremos en la sección 24.2.3.

Un sistema Prolog contiene varios predicados **integrados** que el sistema puede interpretar directamente. Estos suelen incluir el operador de comparación de igualdad $=(X, Y)$, que devuelve el valor verdadero si X y Y son idénticos y que también puede escribirse $X = Y$ usando la notación infija estándar.¹ Otros operadores de comparación para números, como $<$, $\leq$, $>$ y $\geq$, se pueden tratar como funciones binarias. Las funciones aritméticas como $+$, $-$, $*$ y $/$ se pueden usar como argumentos en los predicados en Prolog, aunque Datalog (en su forma básica) no permite funciones del tipo de las operaciones aritméticas como argumentos; de hecho, ésta es una de las diferencias principales entre Prolog y Datalog. No obstante, se han propuesto extensiones a Datalog que incluyen funciones.

Por lo regular, una **consulta** contiene un símbolo de predicado con algunos argumentos variables, y su significado (o “respuesta”) es deducir las diferentes combinaciones de constantes que, cuando se **enlazan** (assign) a las variables, pueden hacer que se cumpla el predicado. Por ejemplo, la primera consulta de la figura 24.1 solicita los nombres de todos los subordinados de “jaime” en cualquier nivel. Un tipo diferente de consulta, que sólo tiene símbolos constantes como argumentos, devuelve un resultado falso o verdadero, dependiendo de si los argumentos provistos se pueden deducir de los hechos y las reglas. Por ejemplo, la segunda consulta de la figura 24.1 devuelve el valor verdadero si es posible deducir superior(jaime, josefa), y devuelve falso en caso contrario.

24.2.2 Notación de Datalog

Aquí presentaremos algunos conceptos básicos y una notación asociados a Datalog. En Datalog, al igual que en otros lenguajes basados en lógica, los programas se construyen a partir de objetos básicos llamados **fórmulas atómicas**. Para definir la sintaxis de los lenguajes basados en lógica se acostumbra describir la sintaxis de las fórmulas atómicas e indicar cómo pueden combinarse para formar un programa. En Datalog, las fórmulas atómicas son **literales** de la forma $p(a_1, a_2, \dots, a_n)$, donde p es el nombre del predicado y n es el número de argumentos de dicho predicado. Diferentes símbolos de predicado pueden tener diferentes números de argumentos, y al número de argumentos n de un predicado p se le llama la **aridad** de p . Los argumentos pueden ser valores constantes o nombres de variables. Como se

¹Los sistemas Prolog por lo regular tienen varios predicados de igualdad diferentes con distintas interpretaciones.

mencionó antes, usamos la convención de que los valores constantes o bien son numéricos o bien comienzan con una *letra minúscula*, en tanto que los nombres de variables siempre comienzan con una *letra mayúscula*.

Datalog incluye varios **predicados integrados** que también pueden servir para construir fórmulas atómicas. Estos predicados son de dos tipos principales: los predicados de comparación binarios $<$ (less), $\leq$ (less_or_equal), $>$ (greater) y $\geq$ (greater_or_equal) sobre dominios ordenados; y los predicados de comparación $=$ (equal) y $\neq$ (not_equal) sobre dominios ordenados o no ordenados. Éstos pueden usarse como predicados binarios con la misma sintaxis que los demás predicados —por ejemplo, escribiendo $\text{less}(X, 3)$ — o se pueden especificar empleando la notación infija acostumbrada $X < 3$. Cabe señalar que, como los dominios de estos predicados son potencialmente infinitos, se deben usar con cuidado en las definiciones de reglas. Por ejemplo, el predicado $\text{greater}(X, 3)$, si se usa solo, genera un conjunto infinito de valores para X que satisfacen el predicado (todos los números enteros mayores que 3). Analizaremos este problema un poco más adelante.

Una **literal** es una fórmula atómica precedida por **not**. Esta última es una fórmula **literal positiva**—o bien una fórmula atómica precedida por **not**. Esta última es una fórmula atómica negada, denominada **literal negativa**. Los programas en Datalog pueden considerarse como un *subconjunto* de las fórmulas del **cálculo de predicados**, que son un tanto parecidas a las fórmulas del cálculo relacional de dominios (véase el Cap. 8). En Datalog, empero, estas fórmulas se convierten primero en lo que se conoce como **forma clausal** antes de expresarse en Datalog; y sólo pueden usarse en Datalog fórmulas dadas en una forma clausal restringida, llamadas **cláusulas de Horn**.¹

24.2.3 Forma clausal y cláusulas de Horn

Recuerde de la sección 8.1.2 que una fórmula del cálculo relacional es una condición que contiene predicados llamados **átomos** (basados en nombres de relaciones). Además, una fórmula puede tener cuantificadores: el *cuantificador universal* (“para todo”) y el *cuantificador existencial* (“existe”). En la forma clausal, una fórmula se debe convertir en otra fórmula con las siguientes características:

- Todas las variables de la fórmula están cuantificadas universalmente. Por tanto, no es necesario incluir los cuantificadores universales explícitamente; los cuantificadores se eliminan y todas las variables de la fórmula quedan cuantificadas *implícitamente* por el cuantificador universal.
- En forma clausal, la fórmula se compone de varias **cláusulas**, y cada **cláusula** se compone de varias **literales** conectadas exclusivamente por conectores lógicos OR. Así pues, toda **cláusula** es una **disyunción** de literales.
- Las **cláusulas** mismas se conectan exclusivamente mediante conectores lógicos AND, para constituir una fórmula. Así pues, la **forma clausal de una fórmula** es una **conjunción** de cláusulas.

Es posible demostrar que cualquier fórmula puede convertirse a la forma clausal. Para nuestros fines, nos interesa principalmente la forma de las cláusulas individuales, cada una de las cuales es una disyunción de literales. Recuerde que las literales pueden ser positivas o negativas. Consideremos una cláusula de la forma:

¹Nombre dado en honor del matemático Alfred Horn.

$$\text{not}(P_1) \text{ OR } \text{not}(P_2) \text{ OR } \dots \text{OR } \text{not}(P_n) \text{ OR } Q_1 \text{ OR } Q_2 \text{ OR } \dots \text{OR } Q_m \quad (1)$$

Esta cláusula tiene n literales negativas y m literales positivas. Una cláusula así puede transformarse en la siguiente fórmula lógica equivalente:

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{AND } P_n = > Q_1 \text{ OR } Q_2 \text{ OR } \dots \text{OR } Q_m \quad (2)$$

donde $= >$ es el símbolo **implica**. Las fórmulas (1) y (2) son equivalentes, lo que significa que sus valores de verdad son siempre los mismos. Esto es así porque, si todas las literales P_i ($i = 1, 2, \dots, n$) son verdaderas, la fórmula (2) es verdadera sólo si por lo menos una de las Q_i es verdadera, y éste es precisamente el significado del símbolo $= >$ (implica). De manera similar, para la fórmula (1), si todas las literales P_i ($i = 1, 2, \dots, n$) son verdaderas, sus negaciones son todas falsas; por tanto, la fórmula (1) es verdadera sólo si por lo menos una de las Q_i es verdadera. En Datalog, las reglas se expresan como una forma restringida de cláusulas llamadas **cláusulas de Horn**, en las que una cláusula puede contener *cuando más una* literal positiva. Así pues, una cláusula de Horn tiene la forma

$$\text{not}(P_1) \text{ OR } \text{not}(P_2) \text{ OR } \dots \text{OR } \text{not}(P_n) \text{ OR } Q \quad (3)$$

o bien la forma

$$\text{not}(P_1) \text{ OR } \text{not}(P_2) \text{ OR } \dots \text{OR } \text{not}(P_n) \quad (4)$$

La cláusula de Horn (3) se puede transformar en la cláusula

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{AND } P_n = > Q \quad (5)$$

que se escribe en Datalog como sigue:

$$Q :- P_1, P_2, \dots, P_n. \quad (6)$$

La cláusula de Horn (4) se puede transformar en

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{AND } P_n = > \quad (7)$$

que se escribe en Datalog como sigue:

$$P_1, P_2, \dots, P_n. \quad (8)$$

Así, una regla de Datalog, como la (6), es una cláusula de Horn; y su significado (con base en la fórmula (5)) es que, si todos los predicados P_1 y P_2 y $\dots$ y P_n son verdaderos para un determinado enlace con sus argumentos variables, Q también será verdadero y por tanto podrá ser inferido. La expresión Datalog (8) puede considerarse como una restricción de integridad, donde todos los predicados deben ser verdaderos para satisfacer la consulta.

En general, una consulta en Datalog tiene dos componentes:

- Un programa en Datalog, que es un conjunto finito de reglas.
- Una literal $P(X_1, X_2, \dots, X_n)$, donde cada X_i es una variable o una constante.

Un sistema Prolog o Datalog cuenta con una **máquina de inferencia** interna que puede servir para procesar y calcular los resultados de tales consultas. Las máquinas de inferencia de Prolog por lo regular devuelven un resultado de la consulta (esto es, un conjunto de valores para las variables de la consulta) a la vez, siendo necesario pedir explícitamente la devolución de resultados adicionales.

24.3 Interpretación de reglas*

Existen dos alternativas principales para interpretar el significado teórico de las reglas: por la teoría de demostraciones y por la teoría de modelos. En los sistemas prácticos, el mecanismo de inferencia que tiene el sistema define la interpretación exacta, que pudiera no coincidir con ninguna de las dos interpretaciones teóricas. El mecanismo de inferencia es un procedimiento computacional y por tanto provee una interpretación computacional del significado de las reglas. En esta sección estudiaremos primero las dos interpretaciones teóricas. Después analizaremos brevemente los mecanismos de inferencia como una forma de definir el significado de las reglas. En la sección 24.4 trataremos con mayor detalle mecanismos de inferencia específicos.

Una interpretación es la llamada interpretación de reglas **por la teoría de demostraciones**. En ella, consideramos los hechos y las reglas como enunciados verdaderos, o **axiomas**. Los **axiomas base** no contienen variables. Los hechos son axiomas base que se dan por ciertos. Las reglas se llaman **axiomas deductivos**, ya que pueden servir para deducir hechos nuevos. Con los axiomas deductivos podemos construir demostraciones que deriven hechos nuevos a partir de los ya existentes. Por ejemplo, la figura 24.2 muestra la forma de demostrar el hecho superior (jaimé, ahmed) a partir de las reglas y hechos dados en la figura 24.1. La interpretación por la teoría de demostraciones nos ofrece un enfoque por procedimientos o computacional para calcular una respuesta a la consulta Datalog. Al proceso de demostrar si un determinado hecho (teorema) se cumple se le conoce también como **demostración de teoremas**.

El segundo tipo de interpretación se llama interpretación **por la teoría de modelos**. Aquí, dado un dominio finito¹ o infinito de valores constantes, asignamos a un predicado todas las combinaciones posibles de valores como argumentos. Después debemos determinar si el predicado es verdadero o falso. En general, basta con especificar las combinaciones de argumentos que hacen que el predicado sea verdadero, y decir que todas las demás combinaciones hacen que sea falso. Si esto se hace con todos los predicados, hablamos de una **interpretación** del conjunto de predicados. Por ejemplo, consideremos la interpretación que se muestra en la figura 24.3 para los predicados superior y superior. Esta interpretación asigna un valor de verdad (verdadero o falso) a cada combinación posible de valores de los argumentos (de un dominio finito) para los dos predicados.

A una interpretación se le llama **modelo** para un *conjunto específico de reglas* si esas reglas *siempre se cumplen* en esa interpretación; es decir, para cualesquier valores que se asignen a las variables de las reglas, la cabeza de las reglas es verdadera cuando sustituimos los valores de verdad asignados a los predicados en el cuerpo de la regla según esa interpretación. De este modo, siempre que se aplica una sustitución (enlace) a las variables de las reglas, si todos los predicados del cuerpo de una regla son verdaderos en esa interpretación, el predicado de la cabeza de la regla también debe ser verdadero. La interpretación de la figura 24.3 es un modelo para las dos reglas que se muestran, ya que nunca puede hacer que se violen dichas reglas. Cabe señalar que una regla se viola si un determinado enlace de constantes a las variables hace verdaderos todos los predicados del cuerpo de la regla, pero hace que el predicado de la cabeza de la regla sea falso. Por ejemplo, si *superior(a,b)* y *superior(b,c)* son ambas verdaderas en alguna interpretación, pero *superior(a,c)* no es verdadera, la interpretación no puede ser un modelo de la regla recursiva:

$$\text{superior}(X,Y) :- \text{superior}(X,Z), \text{superior}(Z,Y)$$

¹El dominio que suele escogerse es finito y se llama "universo de Herbrand".

1. superior(X,Y) :- superior(X,Y). (regla 1)
2. superior(X,Y) :- superior(X,Z), superior(Z,Y). (regla 2)
3. supervisor(jazmin,ahmed). (axioma base, dado)
4. supervisor(jaime,jazmin). (axioma base, dado)
5. superior(jazmin,ahmed). (aplicar la regla 1 a 3)
6. superior(jaime,ahmed) :- supervisor(jaime,jazmin), superior(jazmin,ahmed). (aplicar la regla 2 a 4 y 5)

Figura 24.2 Demostración de un hecho nuevo.

En el enfoque según la teoría de modelos, el significado de las reglas se establece proporcionando un modelo para dichas reglas. Podemos considerar que la cabeza de una regla es una consulta definida por su cuerpo (predicados de RHS). El resultado de la consulta es el conjunto de valores (enlazados para los argumentos de los predicados) que por definición hacen que el predicado sea verdadero.

Se dice que un modelo es un **modelo mínimo** para un conjunto de reglas si no es posible cambiar ningún hecho de verdadero a falso y seguir teniendo un modelo para esas reglas. Por ejemplo, consideremos la interpretación de la figura 24.3, y supongamos que el predicado supervisor está definido por un conjunto de hechos conocidos, en tanto que el predicado superior está definido como una interpretación (modelo) para las reglas. Supongamos que añadimos el predicado superior(jaime,bruno) a los predicados verdaderos. Esto

Reglas

superior(X,Y) :- supervisor(X,Y).
superior(X,Y) :- supervisor(X,Z), superior(Z,Y).

Interpretación

Hechos conocidos:

superior(federico,josé) es **verdadero**.
superior(federico,ramón) es **verdadero**.
superior(federico,joseta) es **verdadero**.
superior(jazmin,alicia) es **verdadero**.
superior(jazmin,ahmed) es **verdadero**.
superior(jaime,federico) es **verdadero**.
superior(jaime,jazmin) es **verdadero**.
superior(X,Y) es **falso** para todas las demás combinaciones posibles (X,Y).

Hechos derivados:

superior(federico,josé) es **verdadero**.
superior(federico,ramón) es **verdadero**.
superior(federico,joseta) es **verdadero**.
superior(jazmin,alicia) es **verdadero**.
superior(jazmin,ahmed) es **verdadero**.
superior(jaime,federico) es **verdadero**.
superior(jaime,jazmin) es **verdadero**.
superior(jaime,josé) es **verdadero**.
superior(jaime,ramón) es **verdadero**.
superior(jaime,joseta) es **verdadero**.
superior(jaime,alicia) es **verdadero**.
superior(jaime,ahmed) es **verdadero**.

superior(X,Y) es **falso** para todas las demás combinaciones posibles (X,Y).

Figura 24.3 Una interpretación que es un modelo para las dos reglas indicadas.

sigue siendo un modelo para las reglas que se muestran, pero no es un modelo mínimo, porque podemos cambiar el valor de verdad de superior(jaime,bruno) de verdadero a falso y seguiremos teniendo un modelo para las reglas. El modelo que se muestra en la figura 24.3 es el modelo mínimo para el conjunto de hechos definidos por el predicado supervisor.

En general, el modelo mínimo que corresponde a un conjunto dado de hechos en la interpretación por la teoría de modelos debe ser lo mismo que los hechos generados por la interpretación según la teoría de demostraciones para el mismo conjunto original de axiomas base y deductivos. Sin embargo, esto sólo es cierto en general para reglas con una estructura simple. Si permitimos la negación en la especificación de las reglas, la correspondencia entre las interpretaciones *deja de cumplirse*. De hecho, con la negación son posibles muchos modelos mínimos para un conjunto dado de hechos.

Un tercer enfoque para interpretar el significado de las reglas implica definir un mecanismo de inferencia que el sistema utilice para deducir hechos a partir de las reglas. Este mecanismo de inferencia definirá una **interpretación computacional** del significado de las reglas. El lenguaje de programación lógica Prolog se vale de su mecanismo de inferencia para definir el significado de las reglas y hechos de un programa en Prolog. No todos los programas en Prolog corresponden a las interpretaciones por la teoría de demostraciones o por la teoría de modelos; depende del tipo de reglas que haya en el programa. Sin embargo, en el caso de muchos programas en Prolog simples, el mecanismo de inferencia de Prolog infiere los hechos que corresponden ya sea a la interpretación por la teoría de demostraciones o a un modelo mínimo en la interpretación por la teoría de modelos.

24.4 Mecanismos básicos de inferencia para programas de lógica*

En esta sección analizaremos los dos principales tipos de mecanismos de inferencia computacional que se basan en la interpretación de reglas por la teoría de las demostraciones. En la sección 24.4.1 hablaremos de los mecanismos de inferencia ascendente, en los que la inferencia parte de los hechos dados y genera hechos adicionales que se comparan con el objetivo de una consulta. En la sección 24.4.2 veremos los mecanismos de inferencia descendente, en los que la inferencia parte del objetivo de una consulta y trata de encontrar valores constantes que la hagan verdadera. Este último enfoque se ha utilizado en Prolog.

24.4.1 Mecanismos de inferencia ascendente (encadenamiento hacia adelante)

En la inferencia ascendente, que también se llama **encadenamiento hacia adelante** o **solución ascendente**, la máquina de inferencia parte de los hechos y aplica las reglas para generar hechos nuevos. Al generarse éstos, se comparan con el predicado que es el objetivo de la consulta para ver si coinciden. El término *encadenamiento hacia adelante* indica que la inferencia avanza de los hechos hacia el objetivo. Por ejemplo, consideremos la primera consulta de la figura 24.1, y supongamos que los hechos y reglas que se muestran son los únicos que se cumplen. En la inferencia ascendente, el mecanismo de inferencia verifica primero si cualquiera de los hechos existentes coincide directamente con la consulta dada, superior(jaime,Y)? Puesto que todos los hechos corresponden al predicado supervisor, no hay coincidencia; por tanto, se aplica ahora la primera regla a los hechos existentes para generar hechos nuevos. Esto provoca que la primera regla (no recursiva) genere los hechos

del predicado superior en el orden que se muestra en la figura 24.3. Conforme se genera cada hecho, se compara con el predicado de la consulta. No hay coincidencias hasta que se genera el hecho superior(jaime,federico), con lo que se produce la primera respuesta de la consulta, a saber, $Y=federico$.

En un sistema tipo Prolog, en el que se genera una respuesta a la vez, es preciso introducir peticiones adicionales para buscar la siguiente respuesta; en este caso, el sistema sigue generando hechos nuevos y devuelve la siguiente respuesta, $Y=jazmín$, del hecho generado superior(jaime,jazmín). En este punto, se han agotado todas las posibles aplicaciones de la primera regla (no recursiva), habiéndose generado los primeros siete hechos del predicado superior que se muestran en la figura 24.3. Si se requieren resultados adicionales, la inferencia continúa con la siguiente regla (recursiva) para generar hechos adicionales. Ahora deberá compararse cada hecho supervisor con cada hecho superior, buscando una coincidencia del segundo argumento de supervisor con el primer argumento de superior, a fin de satisfacer ambos predicados RHS: $superior(X,Z)$ y $superior(Z,Y)$. Con ello se generaran los hechos subsecuentes que se listan en la figura 24.4, y en las respuestas adicionales $Y=josé$, $Y=ramón$, $Y=joséfa$, $Y=alicia$ y $Y=ahmed$. Cualesquier hechos nuevos superior que se generen recursivamente también deberán compararse empleando la regla recursiva hasta que no sea posible generar más hechos. En este ejemplo en particular, no se generan más hechos.

En el enfoque ascendente conviene usar una estrategia de búsqueda para generar sólo los hechos que sean pertinentes a una consulta; de lo contrario, se generarán todos los hechos posibles en algún orden que nada tiene que ver con la consulta en cuestión, cosa que puede ser muy ineficiente si los conjuntos de reglas y hechos son grandes.

24.4.2 Mecanismos de inferencia descendente (encadenamiento hacia atrás)

El mecanismo de inferencia descendente es el que se usa en los intérpretes de Prolog. La inferencia descendente, también llamada **encadenamiento hacia atrás** y **resolución descendente**, parte del predicado que es el objetivo de la consulta e intenta encontrar coincidencias con las variables que conduzcan a hechos válidos de la base de datos. El término *encadenamiento hacia atrás* indica que la inferencia retrocede desde el objetivo buscado para determinar hechos que lo satisficieran. En este enfoque no se generan explícitamente hechos, como en el encadenamiento hacia adelante. Por ejemplo, al procesar la consulta superior(jaime,Y)?, el sistema busca primero todos los hechos con el predicado superior cuyo primer argumento coincida con "jaime". Si existe alguno, el sistema genera los resultados en el mismo orden en el que se especificaron los hechos. Puesto que no hay tales hechos en nuestro ejemplo, el sistema buscará entonces la primera regla cuya cabeza (LHS) tenga el mismo nombre de predicado que la consulta, dando lugar a la regla (no recursiva):

superior(X,Y) :- supervisor(X,Y)

A continuación, el mecanismo de inferencia asigna jaime a X, dando lugar a la regla:

superior(jaime,Y) :- supervisor(jaime,Y)

Ahora se dice que la variable X está **enlazada** al valor "jaime". Acto seguido, el sistema sustituye superior(jaime,Y) por supervisor(jaime,Y) y busca hechos que coincidan con supervisor(jaime,Y) para encontrar una respuesta de Y. Los hechos se examinan en el orden en que aparecen en el programa, conduciendo a la primera coincidencia $Y=federico$, seguida de la coincidencia $Y=jazmín$. En este punto, la búsqueda con la primera regla se

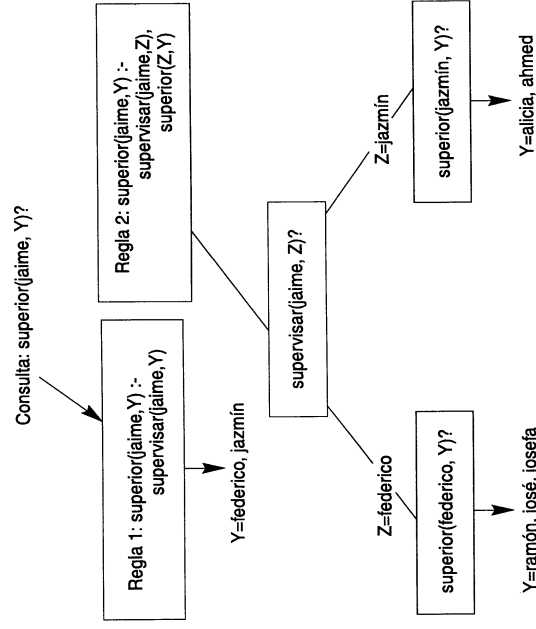


Figura 24.4 Evaluación descendente de una consulta.

agota, por lo que el sistema busca la siguiente regla cuya cabeza (LHS) tenga el nombre de predicado superior, y esto conduce a la regla recursiva. El mecanismo de inferencia enlaza entonces X a "jaime", lo que da como resultado la regla modificada:

superior(jaime, Y) :- supervisor(jaime, Z), superior(Z, Y)

En seguida sustituye el LHS por el RHS y comienza a buscar hechos que satisfagan ambos predicados del RHS. Éstos se llaman ahora **subobjetivos** de la consulta. En este caso, para encontrar una coincidencia con la consulta, el sistema deberá hallar hechos que satisfagan más de un predicado —supervisor(jaime,Z) y superior(Z,Y)— lo que se conoce como **objetivo compuesto**. Una estrategia estándar para satisfacer un objetivo compuesto consiste en utilizar una búsqueda de **primero en profundidad**, lo que significa que el programa intenta primero hallar un enlace que haga verdadero el primer predicado, y luego se dedica a buscar una coincidencia correspondiente para el segundo predicado. Si el enlace del primer predicado no produce ninguna coincidencia que haga verdadero el segundo predicado, el sistema **volverá sobre su rastro** y buscará el siguiente enlace que haga verdadero el primer predicado, después de lo cual continuará la búsqueda como antes.

En nuestro ejemplo, el sistema encuentra la coincidencia supervisor(jaime,federico) para el primer subobjetivo, lo cual **enlaza** Z a "federico", dando como resultado $Z=federico$. Luego busca una coincidencia con superior(federico,Y) para el segundo subobjetivo, que continúa el proceso de comparación utilizando de nuevo la primera regla (no recursiva) y devolviendo en última instancia $Y=josé$, $Y=ramón$ y $Y=joséfa$. A continuación el proceso se repite con $Z=jazmín$, devolviendo $Y=alicia$ y $Y=ahmed$. Esto se muestra gráficamente en la figura 24.4.

En este ejemplo no hay vínculos superiores adicionales “de tercer nivel”, pero si los hubiera también se generarían en el orden apropiado dentro del proceso de inferencia (véase el ejercicio 24.1).⁷

Se ha investigado intensamente en torno a la creación de mecanismos de inferencia más eficientes en el campo de la programación lógica. En particular, es posible usar técnicas de búsqueda de **primero en amplitud**, en vez de primero en profundidad, para objetivos compuestos, pues la búsqueda de coincidencias con múltiples subobjetivos puede efectuarse en paralelo. También se han propuesto técnicas de optimización diseñadas para guiar la búsqueda empleando primero las reglas más prometedoras durante la inferencia.

El mecanismo descendente de primero en profundidad da pie a ciertos problemas debido a su dependencia respecto al orden en que se escriben las reglas y los hechos. Por ejemplo, cuando se escriben reglas para definir un predicado de manera recursiva, como en la definición del predicado superior, hay que escribir los subobjetivos en el orden que se muestra para que no haya una recursión infinita durante el proceso de inferencia. Otro problema se presenta cuando en las definiciones de las reglas interviene la negación, cosa que no podremos analizar en detalle, salvo por una breve mención en la sección 24.5.5. Estos problemas han propiciado la definición de algunos mecanismos de inferencia o estrategias de evaluación de consultas diferentes para el lenguaje Datalog en el caso de aplicaciones en las que intervienen bases de datos.

24.5 Programas en Datalog y su evaluación*

Hay dos métodos principales para definir los valores de verdad de los predicados en los programas Datalog reales. Los **predicados definidos por hechos** (o **relaciones**) se definen mediante una lista de todas las combinaciones de valores (las tuplas) que hacen verdadero el predicado. Éstas corresponden a las relaciones base cuyos contenidos se almacenan en un sistema de bases de datos. La figura 24.5 muestra los predicados definidos por hechos empleado, hombre, mujer, departamento, supervisor, proyecto y trabajaen, que corresponden a una parte de la base de datos relacional que se muestra en la figura 6.6. Los **predicados definidos por reglas** (o **vistas**) se definen al ser la cabeza (LHS) de una o más reglas Datalog; corresponden a *relaciones virtuales* cuyos contenidos puede inferir la máquina de inferencia. La figura 24.6 muestra varios predicados definidos por reglas que hacen referencia a la base de datos de la figura 24.5.

24.5.1 Seguridad de los programas

Se dice que un programa o una regla es **seguro** si genera un conjunto *finito* de hechos. El problema teórico general de determinar si un conjunto de reglas es o no seguro es indecidible. Sin embargo, es posible determinar la seguridad de formas restringidas de reglas. Por ejemplo, las reglas que se muestran en la figura 24.5 son seguras. Obtendremos reglas inseguras, que pueden generar un número infinito de hechos, cuando una de las variables de la regla pueda abarcar un dominio infinito de valores y esa variable no esté limitada a abarcar una relación finita. Por ejemplo, consideremos la regla:

salario_grande(Y) :- Y>60000

⁷Advierta que, en nuestro ejemplo, el orden de búsqueda es muy similar tanto para el encadenamiento hacia adelante como para el encadenamiento hacia atrás. Sin embargo, esto no es la regla en casos más complejos.

empleado(josé).	hombre(josé).
empleado(federico).	hombre(federico).
empleado(alicia).	hombre(ramón).
empleado(jazmín).	hombre(ahmed).
empleado(ramón).	hombre(jaime).
empleado(josefa).	mujer(alicia).
empleado(ahmed).	mujer(jazmín).
empleado(jaime).	mujer(josefa).
salario(josé,30000).	proyecto(productox).
salario(federico,40000).	proyecto(productoy).
salario(alicia,25000).	proyecto(productoz).
salario(jazmín,43000).	proyecto(automatización).
salario(ramón,38000).	proyecto(reorganización).
salario(josefa,25000).	proyecto(nuevasprestaciones).
salario(ahmed,25000).	trabajaen(josé,productox,32).
salario(jaime,55000).	trabajaen(josé,productoy,8).
departamento(josé,investigación).	trabajaen(ramón,productox,40).
departamento(federico,investigación).	trabajaen(josefa,productox,20).
departamento(alicia,administración).	trabajaen(josefa,productoy,20).
departamento(jazmín,administración).	trabajaen(federico,productoy,10).
departamento(ramón,investigación).	trabajaen(federico,productoz,10).
departamento(josefa,investigación).	trabajaen(federico,automatización,10).
departamento(ahmed,administración).	trabajaen(federico,reorganización,10).
departamento(jaime,dirección).	trabajaen(alicia,nuevasprestaciones,30).
supervisor(federico,josé).	trabajaen(alicia,automatización,10).
supervisor(federico,ramón).	trabajaen(ahmed,automatización,35).
supervisor(federico,josefa).	trabajaen(ahmed,nuevasprestaciones,5).
supervisor(jazmín,alicia).	trabajaen(jazmín,nuevasprestaciones,20).
supervisor(jazmín,ahmed).	trabajaen(jazmín,reorganización,15).
supervisor(jaime,federico).	trabajaen(jaime,reorganización,10).
supervisor(jaime,jazmín).	

Figura 24.5 Predicados definidos por hechos que describen parte de la base de datos de la figura 6.6.

Aquí, podemos obtener un resultado infinito si Y abarca todos los enteros posibles. Pero si ponemos que cambiamos la regla así:

salario_grande(Y) :- empleado(X),salario(X,Y),Y>60000

En la segunda regla el resultado no es infinito, porque ahora los valores a los que se puede enlazar Y están limitados a valores que sean el salario de algún empleado de la base de datos; supuestamente, un conjunto finito de valores. También podemos reescribir la regla como sigue:

salario_grande(Y) :- Y>60000,empleado(X),salario(X,Y)

superior(X,Y) :- supervisor(X,Y).
superior(X,Y) :- supervisor(X,Z), superior(Z,Y).
subordinado(X,Y) :- superior(Y,X).
supervisor(X) :- empleado(X), supervisor(X,Y).
emp_sobre_40K(X) :- empleado(X), salario(X,Y), Y>=40000.
supervisor_bajo_40K(X) :- supervisor(X), not(emp_sobre_40K(X)).
emp_principal_prod(X) :- empleado(X), trabajaen(X,productox,Y), Y>=20.
presidente(X) :- empleado(X), not(supervisor(Y,X)).

Figura 24.6 Predicados definidos por reglas.

En este caso la regla sigue siendo, en teoría, segura. Cabe señalar, empero, que, en Prolog o en cualquier otro sistema que use un mecanismo de inferencia descendente de primero en profundidad, la regla creará un ciclo infinito, pues primero se busca un valor para Y y luego se verifica si es el salario de un empleado. El resultado es la generación de un número infinito de valores Y , a pesar de que éstos, después de un cierto punto, no pueden producir un conjunto de predicados RHS verdaderos. Una definición de Datalog considera que ambas reglas son seguras, ya que no depende de un mecanismo de inferencia particular. No obstante, en general es aconsejable escribir este tipo de reglas en la forma más segura, colocando primero los predicados que restringen los posibles enlaces de las variables. Como ejemplo adicional de regla insegura, considere la siguiente regla:

tiene_algo(X,Y) :- empleado(X)

Aquí también es posible generar un número infinito de valores Y , puesto que la variable Y aparece sólo en la cabeza de la regla y por tanto no está limitada a un conjunto finito de valores. Para definir reglas seguras de manera más formal, utilizamos el concepto de variable limitada. Una variable X es **limitada** en una regla si: (a) aparece en un predicado normal (no integrado) en el cuerpo de la regla; (b) aparece en un predicado de la forma $X=c$ o $c=X$ o $(c1 < X$ and $X < c2)$ en el cuerpo de la regla, donde c , $c1$ y $c2$ son valores constantes; o (c) aparece en un predicado de la forma $X=Y$ o $Y=X$ en el cuerpo de la regla, donde Y es una variable limitada. Se dice que una regla es **segura** si todas sus variables son limitadas.

24.5.2 Empleo de operaciones relacionales

Resulta sencillo especificar muchas operaciones del álgebra relacional en forma de reglas de Datalog que definen el resultado de aplicar estas operaciones a las relaciones de la base de datos (predicados de hechos). Esto significa que las consultas relacionales y vistas basadas en consultas se pueden especificar fácilmente en Datalog. La capacidad adicional que Datalog ofrece radica en la especificación de consultas recursivas y de vistas basadas en dichas consultas. En esta sección, veremos cómo especificar algunas de las operaciones relacionales estándar en forma de reglas de Datalog. Supongamos que están disponibles las relaciones base (predicados definidos por hechos) rel_uno , rel_dos y rel_tres , cuyos esquemas se muestran en la figura 24.7, y que las operaciones relacionales se aplican a esas relaciones. Observe que, en Datalog, no es preciso especificar los nombres de atributos de la figura 24.7; más bien, la aridad de cada predicado es el aspecto importante. En un sistema práctico, el dominio (tipo de datos) de cada atributo también es importante en operaciones como UNIÓN, INTERSECCIÓN y REUNIÓN, y suponemos que los tipos de los atributos son compatibles para las diversas operaciones, como se explicó en el capítulo 6.

La figura 24.7 ilustra varias operaciones relacionales básicas. Observe que, si el modelo Datalog está basado en el modelo relacional y por ende supone que los predicados (relaciones de hechos y resultados de consultas) especifican conjuntos de tuplas, se eliminarán automáticamente las tuplas duplicadas en un mismo predicado. Esto puede o no ser cierto, dependiendo de la máquina de inferencia Datalog. Sin embargo, definitivamente *no* es el caso en Prolog, por lo que cualquiera de las reglas de la figura 24.7 que implique la eliminación de duplicados no será correcta para Prolog. Por ejemplo, si queremos especificar reglas Prolog para la operación UNIÓN con eliminación de duplicados, deberemos reescribirlas como sigue:

```
rel_uno(A,B,C).
rel_dos(D,E,F).
rel_tres(G,H,I,J).

seleccionar_uno_A_igual_c(X,Y,Z) :- rel_uno(c,Y,Z).
seleccionar_uno_B_menor_5(X,Y,Z) :- rel_uno(X,Y,Z), Y<5.
seleccionar_uno_A_igual_c_y_B_menor_5(X,Y,Z) :- rel_uno(c,Y,Z), Y<5.

seleccionar_uno_A_igual_c_o_B_menor_5(X,Y,Z) :- rel_uno(c,Y,Z).
seleccionar_uno_A_igual_c_o_B_menor_5(X,Y,Z) :- rel_uno(X,Y,Z), Y<5.

proyectar_tres_sobre_G_H(W,X) :- rel_tres(W,X,Y,Z).

unión_uno_dos(X,Y,Z) :- rel_uno(X,Y,Z).
unión_uno_dos(X,Y,Z) :- rel_dos(X,Y,Z).

intersección_uno_dos(X,Y,Z) :- rel_uno(X,Y,Z), rel_dos(X,Y,Z).

diferencia_dos_uno(X,Y,Z) :- rel_dos(X,Y,Z), not(rel_uno(X,Y,Z)).

prod_cart_uno_tres(T,U,V,W,X,Y,Z) :- rel_uno(T,U,V), rel_tres(W,X,Y,Z).

reunión_natural_uno_tres_C_igual_G(U,V,W,X,Y,Z) :-
    rel_uno(U,V,W), rel_tres(W,X,Y,Z).
```

Figura 24.7 Predicados para ilustrar operaciones relacionales.

```
unión_uno_dos(X,Y,Z) :- rel_uno(X,Y,Z).
unión_uno_dos(X,Y,Z) :- rel_dos(X,Y,Z), not(rel_uno(X,Y,Z)).
```

Sin embargo, las reglas que aparecen en la figura 24.6 sí deberán funcionar para Datalog, si se eliminan automáticamente los registros duplicados. De manera similar, las reglas para la operación Proyectar que aparecen en la figura 24.7 deberán funcionar para Datalog en este caso, pero no son correctas para Prolog, pues aparecerían duplicados en el segundo caso.

24.5.3 Evaluación de consultas no recursivas en Datalog

Las implementaciones de Prolog se han basado en el enfoque de encadenamiento hacia atrás, en el que el ordenamiento de los predicados es significativo. Puesto que Datalog se ha definido como un subconjunto de Prolog pueden usarse con Datalog los mecanismos de inferencia para los lenguajes de programación lógica, como los encadenamientos hacia adelante o hacia atrás. Sin embargo, si Datalog ha de usarse en un sistema de bases de datos deductivas, convendrá definir un mecanismo de inferencia basado en los conceptos de procesamiento de consultas de las bases de datos relacionales. En el procesamiento de consultas relacionales, la estrategia inherente implica una evaluación ascendente, partiendo de las relaciones base; el orden de las operaciones se mantiene flexible y sujeto a optimización. En esta sección analizaremos un mecanismo de inferencia basado en operaciones relacionales, que se puede aplicar a consultas Datalog **no recursivas**. Usaremos la base de hechos y reglas que aparece en las figuras 24.5 y 24.6 para ilustrar nuestro análisis.

Si una consulta implica sólo predicados definidos por hechos, la inferencia se reduce a buscar el resultado de la consulta entre los hechos. Por ejemplo, una consulta como `departamento(X,investigación)?`

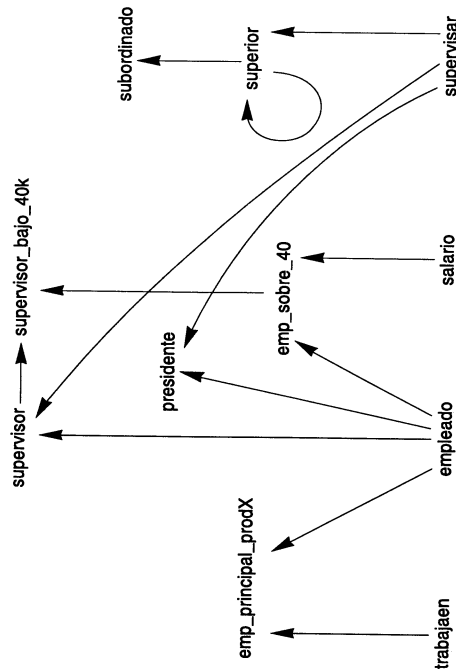


Figura 24.8 Grafo de dependencia de predicados para la figura 24.6.

es una selección de todos los nombres de empleado X que trabajan para el departamento 'investigación', y puede responderse examinando el predicado definido por hechos departamento (X, Y). Una consulta así es similar a una operación SELECT relacional sobre una relación base, y le podemos aplicar las técnicas de procesamiento de consultas y de optimización que vimos en el capítulo 16.

Cuando en una consulta intervienen predicados definidos por reglas, el mecanismo de inferencia debe calcular el resultado con base en las definiciones de las reglas. Si una consulta no es recursiva e implica un predicado p que aparece como cabeza de una regla $p :- p_1, p_2, \dots, p_n$, la estrategia consiste en calcular primero las relaciones que corresponden a $p_1, p_2, \dots, p_n$ y luego calcular la que corresponde a p , con la que puede generarse el resultado de la consulta. Para determinar el vínculo entre los predicados, resulta útil seguir la pista a la dependencia entre los predicados de una base de datos deductiva en un **grafo de dependencia de predicados**. La figura 24.8 muestra el grafo de dependencia para los predicados de hechos y reglas ilustrados en las figuras 24.5 y 24.6. El grafo de dependencia contiene un **nodo** por cada predicado. Siempre que un predicado A se especifique en el cuerpo (RHS) de una regla, y la cabeza (LHS) de esa regla sea el predicado B , decimos que B **depende de** A , y dibujamos una arista dirigida de A a B . Esto indica que, para calcular los hechos del predicado B (la cabeza de la regla), debemos primero calcular los hechos de todos los predicados A en el cuerpo de la regla. Si el grafo de dependencia no tiene ciclos, el conjunto de reglas es **no recursivo**. Si hay por lo menos un ciclo, se dice que el conjunto de reglas es **recursivo**. En la figura 24.8 hay un predicado definido recursivamente —a saber, superior— que tiene una arista recursiva que apunta a sí mismo, pues su regla está definida recursivamente. Además, como el predicado subordinado depende de superior, también requiere recursión para calcular su resultado.

Una consulta que sólo contiene predicados no recursivos es una **consulta no recursiva**. En esta sección únicamente nos ocuparemos de los mecanismos de inferencia para consultas no recursivas. En la figura 24.8, cualquier consulta en la que no intervengan los predicados subordinado ni superior será no recursiva. En el grafo de dependencia de

predicados, los nodos que corresponden a predicados definidos por hechos no tienen aristas dirigidas hacia ellos, ya que todos estos predicados tienen sus hechos almacenados en una relación de la base de datos. Se puede calcular el contenido de un predicado definido por hechos independientemente de otros predicados, obteniendo directamente las tuplas de la relación correspondiente.

La función principal de un mecanismo de inferencia es calcular los hechos que corresponden a predicados de consultas. Esto puede lograrse generando una **expresión relacional** con operadores de bases de datos relacionales como SELECCIONAR, PROYECTAR, REUNIR, UNIÓN Y DIFERENCIA DE CONJUNTOS (con las precauciones apropiadas para tener reglas seguras), la cual, al ejecutarse, produzca el resultado de la consulta. La consulta puede entonces llevarse a cabo utilizando las operaciones internas de procesamiento y optimización de consultas de un sistema de gestión de bases de datos relacionales. Siempre que el mecanismo de inferencia necesite calcular el conjunto de hechos correspondiente a un predicado no recursivo definido por reglas, p , primero localizará todas las reglas que tengan a p como cabeza. La idea es calcular el conjunto de hechos para cada una de estas reglas y luego aplicar la operación de *unión* a los resultados, ya que la unión corresponde a una operación OR lógica. El grafo de dependencia indica todos los predicados q de los cuales depende cada p , y como suponemos que el predicado no es recursivo, siempre podremos determinar un orden parcial entre tales predicados q . Antes de calcular el conjunto de hechos para p , calcularemos los conjuntos de hechos para los predicados q de los cuales p depende, con base en su orden parcial. Por ejemplo, si en una consulta interviene el predicado supervisor_bajo_40K, deberemos calcular primero tanto supervisor como emp_sobre_40K. Dado que estos dos últimos dependen sólo de los predicados definidos por hechos empleado, salario y supervisor, se pueden calcular directamente a partir de las relaciones de la base de datos almacenada.

También debemos manejar los predicados integrados, como los operadores de comparación $>$, $<$ y $=$, si aparecen en el cuerpo de alguna regla. Al especificar un algoritmo de inferencia que calcula el conjunto de hechos de cualquier consulta no recursiva, es común convertir las reglas a una forma canónica llamada **reglas rectificadas**. Se dice que una regla está *rectificada* si todos los argumentos del predicado que es la cabeza de la regla son *variables distintas*. Si la cabeza de una regla tiene constantes, o si una variable se repite en la cabeza de la regla, es fácil rectificarla: una constante c se reemplaza por una variable X , y se añade un predicado equal(X, c) al cuerpo de la regla. De manera similar, si una variable Y aparece dos veces en la cabeza de una regla, una de esas ocurrencias se reemplaza por otra variable Z , y se añade un predicado equal(Y, Z) al cuerpo de la regla.

Advierta que, en el caso de las consultas no recursivas, la evaluación de la consulta se puede expresar como un árbol cuyas hojas sean las relaciones base. Lo que se necesita es una aplicación apropiada de las operaciones relacionales SELECCIONAR, PROYECTAR Y REUNIR, junto con las operaciones de conjuntos UNIÓN Y DIFERENCIA DE CONJUNTOS, hasta que se evalúe el predicado de la consulta. A grandes rasgos, un algoritmo de inferencia OBT_EXPR(C) que genera una expresión relacional para calcular el resultado de una consulta Datalog $C = p(\text{arg}_1, \text{arg}_2, \dots, \text{arg}_n)$ se puede expresar informalmente así:

1. Localizar todas las reglas S cuya cabeza sea el predicado p . Si no hay tales reglas, entonces p es un predicado definido por hechos que corresponde a alguna relación de la base de datos R_p ; en este caso, se devuelve una de las siguientes expresiones y el algoritmo termina (usamos la notación $\$i$ para referirnos al nombre del i -ésimo atributo de la relación R_i):

- Si todos los argumentos son variables distintas, la expresión relacional devuelta es R_p .
- Si algunos argumentos son constantes o si la misma variable aparece en más de una posición de argumento, la expresión devuelta es

$$\text{SELECCIONAR}_{\text{condiciones}}(R_p),$$

donde la <condición> de selección es una condición conjuntiva formada por varias condiciones simples conectadas por AND y construida como sigue:

- Si una constante c aparece como el argumento i , incluir una condición simple ($\$i = c$) en la conjunción.
 - Si la misma variable aparece en las dos posiciones de argumentos j y k , incluir una condición ($\$j = \k) en la conjunción.
 - Si un argumento no está presente en ningún predicado, se construye para él una relación unaria que contenga valores que satisfagan todas las condiciones. Como se supone que la regla es segura, esta relación unaria será finita.
2. En este punto tenemos una o más reglas S_i , $i = 1, 2, \dots, n$, $n > 0$, con el predicado p como cabeza. Para cada una de estas reglas S_i , generar una expresión relacional como sigue:

- Aplicar operaciones de selección sobre los predicados del RHS para cada una de estas reglas, como se explicó en el paso 1.
- Construir una reunión natural entre las relaciones que corresponden a los predicados del cuerpo de la regla S_i , sobre las variables comunes. En los argumentos que generaron las relaciones unarias en el paso 1(c), las relaciones correspondientes se introducen como miembros en la reunión natural. Sea R_i la relación resultante de esta reunión.
- Si se definió cualquier predicado integrado X theta Y sobre los argumentos X y Y , el resultado de la reunión se somete a una selección adicional:

$$\text{SELECCIONAR}_{X \text{ theta } Y}(R_i),$$

- Repetir el paso 2(c) hasta que no existan más predicados integrados.
- Obtener la UNIÓN de las expresiones generadas en el paso 2 (si hay más de una regla cuya cabeza sea el predicado p).

24.5.4 Conceptos de procesamiento de consultas recursivas en Datalog

En general, para las consultas de bases de datos deductivas, el problema de optimización se puede descomponer en dos problemas:

- El problema de evaluar la consulta, que surge en relación con un programa que produce una respuesta a la consulta.
- Una parte de estrategia que surge cuando la ejecución óptima de las reglas implica reescribir dichas reglas.

Se han presentado muchos enfoques para las consultas tanto recursivas como no recursivas. Ya vimos un enfoque para evaluar consultas no recursivas; aquí definiremos primero algunos términos relacionados con las consultas recursivas y luego describiremos brevemente los enfoques simple y semisimple para evaluar consultas que atañen al primero de los problemas mencionados y el enfoque de conjunto mágico que atañe al segundo.

Ya hemos visto ejemplos que implican reglas recursivas en las que el mismo predicado aparece en la cabeza y en el cuerpo de la regla. Otro ejemplo es

$$\text{antepasado}(X, Y) :- \text{antepasado}(X, Z), \text{padre}(Z, Y)$$

que dice que Y es un antepasado de X si Z es un antepasado de X y Y es un padre de Z .

Se dice que una regla es **linealmente recursiva** si el predicado recursivo (antepasado) aparece una y sólo una vez en el miembro derecho de la regla. Por ejemplo,

$$\text{mg}(X, Y) :- \text{padre}(X, XP), \text{padre}(Y, YP), \text{mg}(XP, YP)$$

es una regla lineal en la que el predicado mg (primos de la misma generación) se usa sólo una vez en el miembro derecho. La regla dice que X y Y son primos de la misma generación si sus padres son primos de la misma generación. Observe que la regla

$$\text{antepasado}(X, Y) :- \text{antepasado}(X, Z), \text{antepasado}(Z, Y)$$

no es linealmente recursiva. Se cree que la mayor parte de las reglas de la "vida real" se pueden describir como reglas recursivas lineales; se han definido algoritmos para ejecutar de manera eficiente conjuntos lineales de reglas. Las definiciones anteriores se hacen más complicadas cuando se considera un conjunto de reglas con predicados que aparecen tanto en el miembro izquierdo como en el derecho de las reglas.

Dado un programa Datalog con relaciones correspondientes a los predicados, el símbolo $:-$ se puede reemplazar por un signo de igualdad para formar **ecuaciones Datalog**, sin pérdida de significado. Un **punto fijo** de un conjunto de ecuaciones respecto a las relaciones $R_1, R_2, \dots, R_n$ es una solución para estas relaciones que satisface las ecuaciones dadas. Así, un punto fijo constituye un modelo de las reglas a partir de las cuales se definieron las ecuaciones. Es posible que en un conjunto dado de ecuaciones haya dos conjuntos solución, $S_1 < S_2$, tales que cada uno de los predicados de S_1 sea un subconjunto de (o sea igual a) el predicado correspondiente de la solución S_2 . Una solución S_0 es el **menor punto fijo** si $S_0 < S$ para cualquier solución S de esas ecuaciones.

Si representamos un grafo dirigido con el predicado $\text{arista}(X, Y)$ tal que $\text{arista}(X, Y)$ sea verdadero si y sólo si hay una arista del nodo X al nodo Y en el grafo, los caminos del grafo se pueden expresar con las siguientes reglas:

$$\text{camino}(X, Y) :- \text{arista}(X, Y)$$

$$\text{camino}(X, Y) :- \text{camino}(X, Z), \text{camino}(Z, Y)$$

Cabe señalar que hay otras formas de definir caminos recursivamente. Supongamos que las relaciones C y A corresponden a los predicados camino y arista en las reglas anteriores. La **cerradura transitiva** de la relación C contiene todos los posibles pares de nodos que tienen un camino entre ellos, y corresponde a la solución del menor punto fijo correspondiente a las ecuaciones que resultan de las reglas anteriores.[†]

Se ha propuesto un gran número de estrategias para evaluar un conjunto de reglas (ecuaciones) Datalog que puede contener reglas recursivas, pero sus detalles rebasan el

[†]Si se desea un análisis detallado de los puntos fijos, consúltense Ullman (1988).

alcance de este libro. Aquí ilustraremos tres técnicas importantes: la estrategia simple, la estrategia semisimple y el empleo de conjuntos mágicos.

Estrategia simple. La entrada es un conjunto de reglas Datalog con un conjunto de relaciones base que corresponden a los predicados definidos por hechos y otro conjunto de relaciones que corresponden a los predicados definidos por reglas. La salida es la solución del menor punto fijo (LFP: *least fixed point*) de las ecuaciones Datalog que corresponden a estas reglas.

En este procedimiento primero hay que preparar las ecuaciones para las reglas. Se usan valores permanentes en las relaciones definidas por hechos y valores actuales en las relaciones definidas por reglas, a fin de calcular nuevos valores para las relaciones definidas por reglas. El proceso se repite hasta que, en un punto determinado, ninguno de los predicados definidos por reglas presente cambios en sus valores. En este momento, se supone que se ha llegado a la solución LFP. Si el lector desea conocer los detalles del procedimiento, encontrará referencias sobre el tema en las notas bibliográficas al final del capítulo.

Estrategia semisimple. El problema principal de la evaluación simple es que, en cada iteración de la evaluación de las relaciones definidas por reglas, hay un cálculo redundante de las mismas tuplas. Más bien, nos debemos concentrar en el "cambio incremental" del predicado definido por reglas en cada ronda y utilizarlo para calcular tuplas adicionales en la siguiente ronda. La estrategia semisimple es eficiente gracias a su enfoque de calcular el "diferencial" de las tuplas de cada uno de los predicados definidos por reglas en cada iteración. Si el lector desea mayores detalles, deberá consultar las notas bibliográficas al final del capítulo.

Otros enfoques para el procesamiento de consultas recursivas son la estrategia de consulta/subconsulta recursiva de Velle, que es una estrategia descendente interpretada, y la estrategia iterativa descendente compilada de Henschen-Naqvi.[†]

El segundo problema es la optimización de las consultas, cuyo propósito es subsanar las siguientes deficiencias de los enfoques anteriores:

- Las estrategias simple y semisimple no aprovechan como debe ser los enlaces de la consulta.
- Estas estrategias repiten muchos cálculos.

Se han propuesto muchas técnicas que pueden calificarse como técnicas de reescritura de consultas, con el fin de contrarrestar estas dos desventajas. A continuación describiremos una de dichas técnicas.

La técnica de reescritura de reglas de conjunto mágico. El problema que intenta resolver la técnica de reescritura de reglas de conjunto mágico es que a menudo una consulta no solicita toda la relación que corresponde a un predicado intensional, sino un subconjunto pequeño de dicha relación. Consideremos el siguiente programa:

```
mg(X, Y) :- plana(X, Y).
mg(X, Y) :- arriba(X, U), mg(U, V), abajo(V, Y).
```

Aquí, mg es un predicado ("primero de la misma generación"), y la cabeza de ambas reglas es la fórmula atómica mg(X, Y). Los demás predicados de las reglas son plana, arriba y abajo.

[†]Estas se describen en Velle (1986) y en Henschen y Naqvi (1984).

Es de suponer que éstos están almacenados extensionalmente como hechos, en tanto que la relación mg es intensional; esto es, definida sólo por las reglas. En una consulta como mg(josé, Z) —es decir, "¿quiénes son los primos de la misma generación de José?"— nuestra respuesta a la consulta se deberá obtener examinando sólo la parte de la base de datos que sea *pertinente*; a saber, la parte relacionada con individuos que de alguna manera están emparentados con José.

Una búsqueda descendente o de encadenamiento hacia atrás partiría de la consulta como objetivo y usaría las reglas de la cabeza hacia el cuerpo para crear más objetivos; todas éstas serían pertinentes para la consulta, aunque algunas podrían hacernos explorar caminos que no lleven a ningún lado.

Por otra parte, una búsqueda ascendente o de encadenamiento hacia adelante, que procede desde los cuerpos de las reglas hacia las cabezas, nos haría inferir hechos mg que jamás se considerarían en la búsqueda descendente. No obstante, la evaluación ascendente es deseable porque evita los problemas de ciclos y cálculos repetidos que son inherentes al enfoque descendente. Además, los enfoques ascendentes nos permiten utilizar operaciones que afectan conjuntos completos, como las reuniones relacionales, que son muy eficientes en el contexto de datos residentes en disco, en tanto que los métodos descendentes puros utilizan operaciones que afectan una tupla a la vez.

La reescritura de reglas de conjuntos mágicos es una técnica que nos permite reescribir las reglas en función exclusivamente de la forma de la consulta; es decir, considera cuáles argumentos del predicado están enlazados a constantes, y cuáles son variables, de modo que se combinen las ventajas de los métodos descendente y ascendente. La técnica se centra en el objetivo inherente de la evaluación descendente, pero la combina con la ausencia de ciclos, la facilidad de establecer la terminación y la eficiencia de la evaluación ascendente. En vez de presentar el método, del cual se conocen muchas variaciones que se usan en la práctica, explicaremos la idea con un ejemplo.

Dadas las reglas antes expresadas y la consulta mg(josé, Z), una transformación representativa de las reglas a conjuntos mágicos sería:

```
mg(X, Y)      :- mg-mágico(X), plana(X, Y).
mg(X, Y)      :- mg-mágico(X), arriba(X, U), mg(U, V), abajo(V, Y).
mg-mágico(U)  :- mg-mágico(X), arriba(X, U).
mg-mágico(josé).
```

Intuitivamente, podemos ver que los hechos mg-mágico corresponden a consultas o subobjetivos. La definición del predicado mg-mágico imita la forma como se generan objetivos en la evaluación descendente. El conjunto de hechos mg-mágico se utiliza como filtro en las reglas que definen mg, para evitar generar hechos que no sean respuestas de algún subobjetivo. Así, una evaluación exclusivamente ascendente y de encadenamiento hacia adelante del programa reescrito logra una restricción de la búsqueda similar a la que se logra con una evaluación descendente del problema original. Dar más detalles de esta técnica rebasaría el alcance de esta obra.

Aunque la técnica de conjuntos mágicos originalmente fue creada para manejar consultas recursivas, también es aplicable a las no recursivas. De hecho, se ha adaptado para manejar consultas SQL (que contienen funciones como la agrupación, la agregación, las condiciones aritméticas y las relaciones multiconjuntos que no están presentes en las consultas lógicas puras), y ha resultado útil para evaluar consultas SQL "anidadas" no recursivas.

24.5.5 Negación estratificada

Los lenguajes de consulta de las bases de datos deductivas pueden ampliarse permitiendo literales negadas en los cuerpos de las reglas de los programas. Sin embargo, una vez que permitimos literales negadas en las reglas, perdemos una propiedad importante de éstas denominada *modelo mínimo*, de la que ya hablamos antes. Este modelo se calcula mediante evaluación simple o semisimple, como se explicó en la sección anterior. Sin embargo, es posible que un programa no tenga un modelo mínimo si hay literales negadas. Por ejemplo, el programa

$$p(a) :- \text{not } p(b).$$

tiene dos modelos mínimos: $\{p(a)\}$ y $\{p(b)\}$.

No es nuestro propósito ofrecer una historia detallada del concepto de negación, pero para fines prácticos conviene entender la importante noción de *negación estratificada*, que se usa en las implementaciones de sistemas deductivos.

El significado de un programa con negación por lo regular está dado por algún modelo "propuesto"; lo difícil es crear algoritmos para escoger un modelo propuesto que:

1. Tenga sentido para el usuario de las reglas.
2. Permita responder consultas acerca del modelo de manera eficiente.

En particular, es deseable que el modelo trabaje bien con la transformación de conjuntos mágicos, en el sentido de que podamos modificar las reglas con alguna generalización adecuada de conjuntos mágicos, y las reglas resultantes permitan calcular de manera eficiente (sólo) la porción pertinente del modelo seleccionado. (La alternativa es crear otras técnicas de evaluación eficientes.)

Una clase de negaciones importante que se ha estudiado mucho es la **negación estratificada**. Un programa está **estratificado** si no hay recursión a través de una negación. Los programas de esta clase tienen una semántica muy intuitiva y se pueden evaluar de manera eficiente. El ejemplo que sigue describe un programa estratificado. Consideremos el programa P_2 :

$$\begin{aligned} r_1: & \text{antepasado}(X, Y) :- \text{padre}(X, Y). \\ r_2: & \text{antepasado}(X, Y) :- \text{padre}(X, Z), \text{antepasado}(Z, Y). \\ r_3: & \text{nociclo}(X, Y) :- \text{antepasado}(X, Y), \text{not } \text{antepasado}(Y, X). \end{aligned}$$

Advierta que la tercera regla tiene una literal negativa en su cuerpo. Este programa está estratificado porque la definición del predicado *nociclo* depende (negativamente) de la definición de *antepasado*, pero la de *antepasado* no depende de la de *nociclo*. No estamos en condiciones de dar una definición más formal sin presentar notaciones y definiciones adicionales. Una evaluación ascendente de P_2 calcularía primero un punto fijo de las reglas r_1 y r_2 (las reglas que definen *antepasado*). La regla r_3 se aplica sólo cuando se conocen todos los hechos *antepasado*.

Una extensión natural de los programas estratificados es la clase de programas localmente estratificados. En términos intuitivos, un programa P está *localmente estratificado* para una base de datos determinada si, cuando sustituimos constantes por variables de todas las formas posibles, las reglas con ejemplares resultantes no tienen recursión a través de una negación.

24.6 El sistema LDL*

El proyecto Logic Data Language (lenguaje lógico de datos: LDL) de Microelectronics and Computer Technology Corporation (MCC) se inició en 1984 con dos objetivos primarios:

- Crear un sistema que extendiera el modelo relacional y que a la vez aprovechara algunas de las características positivas de un SCBDR (sistema de gestión de bases de datos relacionales).
- Mejorar la funcionalidad de un SCBD de modo que operara como SCBD deductivo y además permitiera la creación de aplicaciones de propósito general.

Ahora el sistema resultante es un SCBD deductivo que se encuentra en el mercado. En esta sección haremos un breve repaso de los puntos principales del enfoque adoptado por LDL y consideraremos sus características importantes.

24.6.1 Antecedentes, motivación y panorámica

El diseño del lenguaje LDL puede considerarse como una extensión basada en reglas de los lenguajes basados en el cálculo de dominios. Ya vimos el cálculo de dominios en la sección 8.3 y analizamos el lenguaje QBE en la sección 8.4. El sistema LDL ha intentado alcanzar el poder de expresión y deducción de Prolog y al mismo tiempo igualar la facilidad de uso de QBE. Otro reto al que se enfrentó fue el de combinar la capacidad de expresión de Prolog con la funcionalidad y los recursos de un SCBD de propósito general. Esto último incluye el modelado y almacenamiento de datos, que tratamos en las primeras dos partes de este libro, y el procesamiento de transacciones y la optimización de consultas que analizamos en la parte 4 del libro. MCC construyó un sistema integrado, en vez de acoplar Prolog a bases de datos relacionales.

La desventaja principal que experimentaron los primeros sistemas que acoplaron Prolog a un SCBDR es que Prolog es un lenguaje de recorrido (navegación) en tanto que en los SCBDR el usuario formula una consulta correcta y deja al sistema la optimización de su ejecución. La naturaleza de recorrido de Prolog es evidente en el ordenamiento de las reglas y los objetivos para lograr una ejecución y terminación óptimas. Se dispone de dos opciones:

- Hacer que Prolog sea más "parecido a las bases de datos" mediante la adición de características de gestión de bases de datos por recorrido (se habló de esto en el capítulo 10).
- Modificar Prolog para convertirlo en un lenguaje de lógica declarativo de aplicación general.

En LDL se eligió la segunda opción, produciendo un lenguaje que es diferente de Prolog en sus construcciones y estilo de programación.

LDL difiere del estilo de programación Prolog/Datalog en los siguientes aspectos:

- Las reglas se compilan en LDL.
- Existe una noción de "esquema" de la base de hechos en LDL en el momento de la compilación. Esta base se actualiza libremente en el momento de la ejecución. Prolog, en cambio, trata los hechos y las reglas de manera idéntica, y somete los hechos a interpretación cuando son modificados.

- LDL no sigue la técnica de resolución y unificación que se emplea en los sistemas Prolog basados en encadenamiento hacia atrás.
- El modelo de ejecución de LDL es más simple, basado en la operación de comparación y el cálculo de "menor punto fijo". Estos operadores, a su vez, utilizan extensiones simples del álgebra relacional.

Las primeras implementaciones de LDL, completadas en 1987, se basaban en un lenguaje llamado FAD. El prototipo de implementación actual, concluida en 1988, se llama SALAD y está sufriendo cambios adicionales al probarse con las aplicaciones de la "vida real" descritas en la sección 24.6.4. El prototipo actual es un sistema portátil eficiente para UNIX que supone una interfaz de una sola tupla tipo "obtener el siguiente" entre el programa LDL compilado y un gestor de hechos subyacente.

24.6.2 El modelo de datos y lenguaje LDL

Dada la filosofía de diseño de LDL de combinar el estilo declarativo de los lenguajes relacionales con el poder de expresión de Prolog, las construcciones de Prolog como negación, conjunto-de, actualizaciones y recortar se han desechado. En su lugar, se extendió la semántica declarativa de las cláusulas de Horn para manejar términos complejos mediante el uso de símbolos de función, llamados funtores en Prolog.

Así, podemos definir un registro de empleado específico como:

```
Empleado (Nombre (Juan Pérez), Puesto(V P),
          Educación(((Bachillerato, 1961),
                    (Universidad (Madrid, lic, matemáticas), 1965),
                    (Universidad (Salamanca, doc, ie), 1975))))
```

En el registro anterior, VP es un término simple, en tanto que educación es un término complejo que consiste en un término para el bachillerato y una relación anidada que contiene el término para la universidad y el año de graduación. De este modo, LDL maneja objetos complejos con una estructura arbitrariamente compleja, incluidos listas, términos de conjuntos, árboles y relaciones anidadas. Podemos visualizar un término compuesto como una estructura Prolog con el símbolo de función como funtor.

LDL permite actualizaciones en los cuerpos de las reglas. Por ejemplo, una regla

```
feliz (Depto, Aumento, Nombre) <-
emp (Nombre, Depto, Sal), Nuevosal = Sal + Aumento,
-emp (Nombre, Depto, -), +emp (Nombre, Depto, Nuevosal).
```

combinada con

```
?feliz(software, 1000, Nombre).
```

concede un aumento de \$1000 a todos los empleados del departamento de software y devuelve los nombres de esos felices empleados. Se considera que esta consulta es una transacción indivisible.

LDL ofrece una construcción if-then-else de semántica declarativa nítida, para poder expresar claramente e implementar con eficiencia reglas mutuamente disyuntivas. Por añadidura, cuenta con un predicado "choice" ("elección") no orientado a procedimientos para situaciones en las que cualquier respuesta será satisfactoria. El predicado de elección puede servir para obtener una respuesta única, en vez de la solución con todas las respuestas que

que representa la contestación por omisión de las consultas LDL. En la semántica declarativa de LDL, la negación se ha tratado empleando estratificación y no determinismo, lo que se logra a través de la misma construcción llamada "choice".

24.6.3 Evaluación y optimización de consultas en LDL

Aunque la semántica de LDL se define de manera ascendente (por ejemplo, mediante estratificación), el implementador puede usar cualquier ejecución que se apegue a esta semántica declarativa. En particular, la ejecución puede ser ascendente o descendente, o bien híbrida. Estas opciones permiten al compilador/optimizador ser selectivo al adaptar los modos de ejecución más apropiados para el programa dado.

Como primera aproximación, es fácil concebir la ejecución LDL como un cálculo ascendente que emplea el álgebra relacional. Por ejemplo, sea $p(\dots)$ la consulta con la siguiente regla, donde p_1 y p_2 son predicados ya sea de la base de datos o derivados:

$$p(X, Y) \leftarrow \neg p_1(X, Z), p_2(Z, Y).$$

Esta consulta se puede contestar calculando primero las relaciones que representan p_1 y p_2 y calculando luego su reunión, seguida de una proyección. En realidad, el optimizador y el compilador de LDL pueden seleccionar e implementar la regla anterior con cualquiera de estos cuatro modos de ejecución:

- La *ejecución segmentada* calcula sólo las tuplas de p_2 que se reúnen con las tuplas de p_1 en forma segmentada. Esto evita el cálculo de cualquier tupla de p_2 que no se reúna con p_1 (con lo que *no hay trabajo superfluo*); en contraste, si una tupla de p_2 se reúne con muchas tuplas de p_1 , se calculará muchas veces.
- La *ejecución segmentada perezosa* es una ejecución segmentada en la que, conforme se generan las tuplas para p_2 , se almacenan en una relación temporal, digamos rp_2 , para su uso posterior. Por tanto, cualquier tupla de p_2 se calcula exactamente una vez, incluso si se le usa muchas veces (con lo que se realiza *trabajo amortizado*, además de la ausencia de trabajo superfluo de la ejecución segmentada). Por añadidura, como estas dos ejecuciones segmentadas calculan las tuplas p_2 una por una, es posible evitar cálculos residuales en caso de haber marcha atrás inteligente; esto se denomina *ventaja de marcha atrás*.
- La *ejecución materializada perezosa* procede como en el caso de la segmentada perezosa excepto que, para un valor de Z dado, todas las tuplas de p_2 que se reúnen con la tupla de p_1 se calculan y almacenan en una relación antes de continuar. La ventaja principal de esta ejecución es que es *reentrante* (una propiedad que resulta importante en el contexto de la recursión), mientras que las dos formas anteriores de ejecución segmentada no lo son, puesto que calculan tuplas de p_2 una por una. Por otro lado, esta ejecución no tiene la ventaja de marcha atrás.
- La *ejecución materializada* calcula todas las tuplas de p_2 y las almacena en una relación (digamos, rp_2). Luego la ejecución continúa, empleando las tuplas de rp_2 . Cabe señalar que esto posee las ventajas de trabajo amortizado y de ser reentrante, pero carece de las ventajas de marcha atrás y de trabajo superfluo.

La ejecución segmentada resulta útil si la columna de reunión es una clave de p_1 , pero la ejecución materializada es la mejor si todos los valores Z de p_2 se reúnen con alguna tupla

p_i . En ambos casos, la evaluación perezosa respectiva implica un mayor gasto extra, debido a que se hace necesaria la verificación de cada una de las tuplas de p_i . La propiedad de ser reentrante es útil sobre todo si el predicado está en el alcance de una consulta recursiva que se está calculando en forma descendente. Por tanto, en estos casos la ejecución materializada perezosa es preferible a la ejecución segmentada perezosa. En otros casos se prefiere la ejecución segmentada perezosa, a fin de aprovechar la propiedad de marcha atrás.

El análisis previo puede generalizarse a cualquier ocurrencia de un predicado con un conjunto (posiblemente vacío) de argumentos enlazados. Aunque hemos limitado nuestro análisis a una sola regla no recursiva, se puede generalizar para incluir cualquier número de reglas con recursión.

Las consultas LDL presentan una nueva serie de problemas, que surgen de las siguientes observaciones: En primer lugar, el modelo de datos se ha mejorado para incluir objetos complejos (como jerarquías y datos heterogéneos para un mismo atributo). En segundo lugar, se necesitan operadores nuevos no sólo para operar sobre datos complejos, sino también para efectuar nuevas operaciones como la recursión y la negación. Así, la complejidad de los datos, además del conjunto de operaciones, subrayan la necesidad de contar con información estadística adicional de la base de datos y estimaciones nuevas de costos.

24.6.4 Aplicaciones de LDL

El sistema LDL se ha utilizado en los siguientes dominios de aplicación:

- **Modelado de empresas:** Este dominio implica modelar la estructura, los procesos y las restricciones dentro de una empresa. Los datos relacionados con ella pueden resultar en un modelo ER extendido que contiene cientos de entidades y vínculos y miles de atributos. Es posible desarrollar varias aplicaciones útiles para los diseñadores de nuevas aplicaciones (así como para los gerentes) a partir de esta "metabase de datos" (véase el Cap. 15), que contiene información tipo diccionario acerca de toda la empresa.
- **Prueba de hipótesis o dragado de datos:** Este dominio implica formular una hipótesis, traducirla a un conjunto de reglas LDL y una consulta, y luego ejecutar la consulta contra los datos dados para probar la hipótesis. El proceso se repite reformulando las reglas y la consulta. Esto se ha aplicado al análisis de datos de genoma (véase la Sec. 25.2.3, donde se proporcionan mayores detalles) en el campo de la microbiología. El dragado de datos consiste en identificar las secuencias de DNA a partir de autorradiografías digitalizadas de bajo nivel obtenidas de experimentos con bacterias *E. coli*.
- **Reutilización de software:** El grueso del software para una aplicación se desarrolla en código estándar por procedimientos, y una pequeña fracción se basa en reglas y se codifica en LDL. Las reglas dan origen a una base de conocimientos que contiene los siguientes elementos:
 - Una definición de cada módulo C empleado en el programa.
 - Un conjunto de reglas que define las formas en que los módulos pueden exportar/importar funciones, restricciones, etcétera.

La "base de conocimientos" puede servir para tomar decisiones referentes a la reutilización de subconjuntos del software. Los módulos pueden recombinarse para satisfacer

tareas específicas, en tanto se satisfagan las reglas pertinentes. Se está experimentando con esto en el software bancario.

24.7 Otros sistemas de bases de datos deductivas*

24.7.1 El sistema CORAL

El sistema CORAL, creado en la University of Wisconsin en Madison, aprovecha las experiencias adquiridas con el proyecto LDL. Al igual que LDL, el sistema provee un lenguaje declarativo basado en cláusulas de Horn con una arquitectura abierta. Sin embargo, hay muchas diferencias importantes, tanto en el lenguaje como en su implementación. El sistema CORAL puede considerarse como un lenguaje de programación que combina características importantes de SQL y Prolog.

Desde el punto de vista del lenguaje, CORAL adapta el elemento de agrupación de conjuntos de LDL para acercarse más al elemento GROUP BY de SQL. Por ejemplo, consideremos:

```
presupuesto(Nombred,sum(<Sal>)) :- depto(Nombred,Nombree,Sal).
```

Esta regla calcula una tupla presupuesto por cada departamento, y cada valor de salario se cuenta tantas veces como haya personas con ese salario en el departamento dado. Esto es exactamente lo que haría la consulta SQL correspondiente. En LDL no es posible combinar la agrupación y la operación de suma en un paso; lo que es más importante, la agrupación produce por definición un *conjunto* de salarios para cada departamento; por tanto, el cálculo del presupuesto es más difícil en LDL. Un aspecto relacionado es que SQL maneja una semántica de *multiconjuntos* para las consultas cuando no se especifica la cláusula DISTINCT. CORAL maneja también una semántica de multiconjuntos similar. Así, puede definirse la siguiente regla para calcular un conjunto de tuplas o un multiconjunto de tuplas en CORAL, al igual que en SQL:

```
presup2(Nombred,Sal) :- depto(Nombred,Nombree,Sal).
```

Esto da pie a una pregunta importante: ¿cómo puede un usuario especificar cuál semántica (conjunto o multiconjunto) desea? En SQL se usa la palabra reservada DISTINCT; de manera similar CORAL ofrece una *anotación*. De hecho, CORAL maneja varias anotaciones que sirven para elegir una semántica deseada o proporcionar sugerencias de optimización al sistema CORAL. La mayor complejidad de las consultas en un lenguaje recursivo dificulta la optimización, y el empleo de anotaciones a menudo significa una diferencia importante en la calidad del plan de evaluación optimizado.

CORAL maneja una clase de programas con negación y agrupación que es estrictamente más grande que la clase de programas estratificados. El problema de lista de materiales, en el que el costo de una parte compuesta se define como la suma de los costos de todas las partes atómicas, es un ejemplo de problema que requiere esta generalidad adicional.

CORAL está más cerca de Prolog que de LDL en cuanto al manejo de tuplas no base; así la tupla $\text{equal}(X, X)$ se puede guardar en la base de datos y denota que toda tupla binaria en la que el primero y el segundo valores de campos sean iguales está en la relación llamada equal. Desde el punto de vista de la evaluación, las principales técnicas de evaluación de CORAL se basan en una evaluación ascendente, muy distinta de la evaluación descendente

de Prolog. Sin embargo, CORAL también ofrece un modo de evaluación descendente similar al de Prolog.

Puesto que se manejan muchas semánticas y métodos de evaluación distintos, CORAL cuenta con un mecanismo de *módulos* para organizar los programas. Cada módulo exporta un predicado de consulta y puede considerarse simplemente como una definición de dicho predicado. Un módulo contiene una o más reglas que definen el predicado exportado y tal vez algunos predicados locales. La semántica y la evaluación de un módulo puede controlarse agregando anotaciones para cada módulo, y los controles de cada uno son completamente independientes de los de otros módulos. Esto permite mezclar la evaluación descendente en un módulo con la evaluación ascendente en otro, por ejemplo.

Desde la perspectiva de la implementación, CORAL lleva a cabo varias optimizaciones para manejar de manera eficiente las tuplas no base, además de usar técnicas como la de patrones mágicos para incluir selecciones en consultas recursivas, incluir proyecciones, y optimizaciones especiales de diferentes clases de programas lineales (derechos e izquierdos). También proporciona una forma eficiente de calcular consultas no estratificadas. Se utiliza un enfoque de "compilación somera", por el cual el sistema interpreta el plan compilado durante la ejecución. Esto hace que la compilación de programas, incluso de los muy grandes, sea extremadamente rápida. CORAL utiliza el gestor de almacenamiento EXODUS para manejar las relaciones residentes en disco. También cuenta con una buena interfaz con C++ y es extensible, lo que permite al usuario adaptar el sistema a aplicaciones especiales añadiendo nuevos tipos de datos o implementaciones de relaciones, por ejemplo. Una característica interesante es el paquete de *explicación*, con el cual el usuario puede examinar gráficamente la forma como se genera un hecho; esto resulta útil para depurar y para proveer explicaciones. En resumen, CORAL maneja consultas declarativas, extendiendo los lenguajes de consulta relacionales y los programas de lógica, pero también es un lenguaje de programación avanzado para aplicaciones que manejan muchos datos. La tendencia actual es hacia la extensión del modelo de datos con características de orientación a objetos, para crear un sistema deductivo y orientado a objetos.

24.7.2 El proyecto NAIL!

El proyecto NAIL! (Not Another Implementation of Logic!) no una implementación de lógica más!) se inició en Stanford en 1985. El objetivo inicial era estudiar la optimización de la lógica mediante el modelo de "todas las soluciones" orientado a bases de datos. En colaboración con el grupo MCC, este proyecto produjo el primer artículo sobre los conjuntos mágicos y el primer trabajo sobre recursiones regulares. Por añadidura, aportó muchas contribuciones importantes para el manejo de la negación y la agregación en las reglas lógicas. También se desarrollaron la negación estratificada, la negación bien fundada y la negación modularmente estratificada en relación con este proyecto.

Se construyó un prototipo de sistema inicial, que posteriormente se abandonó porque el paradigma puramente declarativo resultó inadecuado para muchas aplicaciones. El sistema revisado utiliza un lenguaje central, llamado GLUE, que se compone esencialmente de reglas lógicas individuales, con la capacidad de enunciados SQL, envueltas en construcciones de lenguaje convencionales como ciclos, procedimientos y módulos. El lenguaje NAIL! original se convirtió en un mecanismo de vistas para GLUE; permite especificaciones totalmente declarativas en situaciones en que esto es apropiado.

24.8 Resumen

En este capítulo presentamos una introducción a una rama relativamente nueva de la gestión de bases de datos: los *sistemas de bases de datos deductivos*. Este campo acusa la influencia de los lenguajes de programación de lógica, sobre todo de Prolog. Un subconjunto de Prolog, llamado Datalog, que contiene cláusulas de Horn libres de funciones, es el que se usa primordialmente como fundamento de los trabajos con bases de datos deductivas en la actualidad. En este capítulo presentamos conceptos de Datalog. Estudiamos el mecanismo de inferencia estándar de encadenamiento hacia atrás de Prolog, y una estrategia ascendente de encadenamiento hacia adelante. Esta última se ha adaptado para evaluar consultas sobre relaciones (bases de datos extensionales) mediante operaciones relacionales estándar junto con Datalog. Analizamos de manera informal procedimientos para evaluar consultas recursivas y no recursivas. El manejo de la negación presenta dificultades especiales en estas bases de datos deductivas. Al respecto, se presentó un concepto muy utilizado llamado *negación estratificada*.

Examinamos a grandes rasgos un sistema comercial de bases de datos deductivos llamado LDL, creado por MCC, y otros sistemas experimentales llamados CORAL y NAIL!. El área de las bases de datos deductivas todavía está en una etapa experimental. Su adopción por parte de la industria impulsará su desarrollo. Al respecto, mencionamos aplicaciones prácticas en las que ya se está usando LDL.

Ejercicios

24.1. Añada los siguientes hechos a la base de datos de ejemplo de la figura 24.3:

supervisar(ahmed,bruno), supervisar(federico,gabriela).

En primer término, modifique el árbol de supervisión de la figura 24.1 (b) de modo que refleje este cambio. Luego modifique el diagrama de la figura 24.4 de modo que represente la evaluación descendente de la consulta *supervisor(jaime,Y)*.

24.2. Considere el siguiente conjunto de hechos para la relación *padre(X, Y)*, donde Y es el padre de X:

**padre(a,aa), padre(a,ab), padre(aa,aaa), padre(aa,aab), padre(aaa,aaaa),
padre(aaa,aaab).**

Considere las reglas

r₁: antepasado(X, Y) :- padre(X, Y)

r₂: antepasado(X, Y) :- padre(X, Z), antepasado(Z, Y)

que definen el antepasado Y de X, igual que antes.

(a) Muestre cómo resolver la consulta *Datalog*

antepasado(aa,X)?

empleando la estrategia simple. Anote todos los pasos.

(b) Muestre la misma consulta calculando sólo los cambios en la relación *antepasado* y usándolos en la regla 2 en cada ocasión.

[Esta pregunta se deriva de (Bancilhon y Ramakrishnan 1986).]

24.3. Considere una base de datos deductiva con las siguientes reglas:

$\text{antepasado}(X, Y) :- \text{padre}(X, Y)$
 $\text{antepasado}(X, Y) :- \text{padre}(X, Z), \text{antepasado}(Z, Y)$

Observe que "padre(X, Y)" significa que Y es el padre de X; "antepasado(X, Y)" significa que Y es el antepasado de X. Considere la base de hechos
 $\text{padre}(\text{Enrique}, \text{Isaac}), \text{padre}(\text{Isaac}, \text{Juan}), \text{padre}(\text{Juan}, \text{Carlos}).$

(a) Construya una interpretación de las reglas anteriores según la teoría de modelos, empleando los hechos dados.

(b) Considere que una base de datos contiene las relaciones padre(X, Y) anteriores, otra relación hermano(X, Y) y una tercera relación nacimiento(X, B), donde B es la fecha de nacimiento de la persona X. Exprese una regla que calcule los primeros primos tales que sus padres sean hermanos.

(c) Escriba un programa completo en Datalog con literales basadas en hechos y basadas en reglas para calcular la siguiente relación: lista de pares de primos, donde la primera persona nació después de 1960 y la segunda después de 1970. Se puede usar "mayor que" como predicado integrado. [Nota: también deben mostrarse hechos de ejemplo para hermano, nacimiento y persona.]

24.4. Considere las siguientes reglas:

$\text{alcanzable}(X, Y) :- \text{vuelo}(X, Y)$
 $\text{alcanzable}(X, Y) :- \text{vuelo}(X, Y), \text{alcanzable}(Y, Z)$

donde alcanzable(X, Y) significa que es posible llegar de la ciudad X a la ciudad Y, y vuelo(X, Y) significa que hay un vuelo a la ciudad Y desde la ciudad X.

(a) Construya predicados de hechos que describan lo siguiente:

(i) Los Ángeles, Nueva York, Chicago, Atlanta, Francfort, París, Singapur, Sydney son ciudades.

(ii) Existen los siguientes vuelos: Los Ángeles a Nueva York, Nueva York a Atlanta, Atlanta a Francfort, Francfort a Atlanta, Francfort a Singapur y Singapur a Sydney. [Nota: No puede suponerse que hay automáticamente un vuelo en la dirección opuesta.]

(b) ¿Son cíclicos los datos dados? De ser así, ¿en qué sentido?

(c) Construya una interpretación según la teoría de modelos (esto es, una interpretación similar a la que se muestra en la figura 24.3) de los hechos y reglas anteriores.

(d) Considere la consulta:

$\text{alcanzable}(\text{Atlanta}, \text{Sydney})?$

¿Cómo se ejecutará esta consulta empleando evaluación simple y semisimple? Haga una lista de los pasos que dará.

(e) Considere los siguientes predicados definidos por reglas:

$\text{alcanzable-viaje-redondo}(X, Y) :- \text{alcanzable}(X, Y), \text{alcanzable}(Y, X)$
 $\text{duración}(X, Y, Z)$

Dibuje un grafo de dependencia de predicados para los predicados anteriores. [Nota: duración(X, Y, Z) significa que un vuelo de X a Y tarda Z horas.]

(f) Considere la consulta: ¿cuáles ciudades son alcanzables en 12 horas desde Atlanta? Indique cómo se expresaría esto en Datalog. Suponga predicados integrados como mayor-que(X, Y): ¿Es posible convertir esto en un enunciado del álgebra relacional de manera directa? Explique su respuesta.

(g) Considere el predicado población(X, Y), donde Y es la población de la ciudad X. Considere la consulta: listar todos los posibles enlaces del predicado par(X, Y), donde Y es una ciudad a la que se puede llegar en dos vuelos desde la ciudad X, la cual tiene más de un millón de habitantes. Exprese esta consulta en Datalog. Dibuje un árbol de consulta correspondiente en términos del álgebra relacional.

Bibliografía selecta

En relación con los primeros trabajos sobre el enfoque de lógica y bases de datos, vale la pena estudiar un repaso de Gallaire *et al.* (1984), la reconstrucción hecha por Reiter (1986) de la teoría de las bases de datos relacionales, y el análisis de los conocimientos incompletos a la luz de la lógica hecho por Levesque (1986). Gallaire y Minker (1978) es uno de los primeros libros sobre el tema. En Ullman (1989), vol. II, aparece un tratamiento detallado de la lógica y las bases de datos, y también hay un capítulo relacionado en el volumen I (1988). Ceri, Gottlob y Tanca (1990) presenta un tratamiento completo pero conciso de la lógica y las bases de datos.

Entre los artículos sobre bases de datos deductivas y procesamiento de consultas deductivas hay unos que son excelentes: Bancilhon y Ramakrishnan (1986), Warren (1992) y Ramakrishnan y Ullman (1993). Aho y Ullman (1979) presenta uno de los primeros algoritmos para manejar consultas recursivas, empleando el operador del menor punto fijo. Bancilhon y Ramakrishnan (1986) ofrece una excelente descripción detallada de las estrategias para procesar consultas recursivas, con ejemplos detallados de los enfoques simple y semisimple. Una descripción completa del enfoque semisimple basado en el álgebra relacional se da en Bancilhon (1985). El artículo original sobre conjuntos mágicos es el de Bancilhon *et al.* (1986). Beeri y Ramakrishnan (1987) lo extiende. Mumick *et al.* (1990) muestra la aplicabilidad de los conjuntos mágicos a consultas SQL anidadas no recursivas. Otros enfoques de la optimización de reglas sin reescritura aparecen en Velle (1986, 1987), Kiefer y Lozinski (1986) propone una técnica diferente. Bry (1990) analiza la forma de conciliar los enfoques descendente y ascendente. Whang y Navathe (1992) describe una técnica de forma normal extendida disyuntiva para manejar la recursión en expresiones del álgebra relacional a fin de crear una interfaz de sistema experto sobre un SGBD relacional.

Chang (1981) describe uno de los primeros sistemas para combinar reglas deductivas con bases de datos relacionales. El prototipo del sistema LDL se describe en Chimenti *et al.* (1990); un panorama sobre el sistema apareció en Chimenti (1987). Krishnamurthy y Naqvi (1988) se ocupa de las actualizaciones en LDL; Krishnamurthy y Naqvi (1989) presenta la noción "choice" en LDL. Zaniolo (1988) analiza cuestiones de lenguaje para el sistema LDL.

En Ramakrishnan *et al.* (1992) se ofrece un panorama sobre el lenguaje de CORAL, y la implementación se describe en Ramakrishnan *et al.* (1993). Una extensión para manejar características orientadas a objetos, llamada CORAL++, se describe en Srivastava *et al.* (1993). Ullman (1985) provee la base para el sistema NAIL!, que se describe en Morris *et al.* (1987). Phipps *et al.* (1991) describe el sistema de bases de datos deductivas GJUE-NAIL!. La primera implementación de EDDUCE en EORC se describe en Bocca (1986a, 1986b); más adelante se distribuyó con el nombre Megalog.

CAPÍTULO 25

Nuevas tecnologías y aplicaciones de bases de datos

Hasta aquí hemos tratado las diversas técnicas para modelar datos, diseñar bases de datos e implementarlas en los computadores. Con el término **tecnología de bases de datos** nos referimos colectivamente a las técnicas *comprobadas* que se emplean *a gran escala* en la industria y en el gobierno, así como por usuarios individuales de bases de datos personales, para realizar estas funciones de manera *cotidiana*. Casi todo lo que hemos estudiado ya forma parte de la tecnología actual de las bases de datos. En todas las disciplinas del quehacer humano, la tecnología siempre va a la zaga de las investigaciones actuales por varios años. La tecnología de bases de datos no es la excepción. Por tanto, los trabajos actuales de investigación que tienen el potencial de convertirse en una tecnología viable no estarán disponibles para el consumo general durante otros 5 o 10 años. En este libro es imposible incluir detalles de muchos de estos prometedores esfuerzos de investigación que se están realizando, pero sí queremos presentar al lector en este capítulo una perspectiva amplia de lo más moderno en cuestiones de investigación sobre bases de datos y sistemas de bases de datos. Intentaremos destacar los aspectos más importantes y señalar los problemas más difíciles, en vez de analizar con detalle las soluciones propuestas. Se darán referencias a la literatura pertinente siempre que sea apropiado. En la bibliografía aparecen unas cuantas referencias más.

Este capítulo está organizado como sigue: La sección 25.1 es un panorama sobre los avances en la tecnología de bases de datos, aproximadamente durante los últimos 30 años. En la sección 25.2 se estudian las nuevas áreas de aplicación no convencionales que están aprovechando esta tecnología y que le seguirán exigiendo soluciones. En la sección 25.3 se describen varias tecnologías nuevas que, es de esperarse, se incorporarán en los SCBD futuros. Concluimos el capítulo con la sección 25.4, donde bosquejaremos algunas tecnologías incipientes y comentaremos las líneas de investigación que necesitaremos en el futuro próximo.

Una advertencia: este capítulo *no* debe ser considerado como un resumen exhaustivo de las tendencias actuales de la investigación. Las omisiones en algunas áreas de investigación no deben considerarse como implicaciones de alguna preferencia. El amplísimo campo de la investigación sobre la gestión de bases de datos, del que cada año se dan a conocer al menos 200 o 300 artículos en conferencias con arbitraje y publicaciones periódicas, no se puede condensar en este pequeño capítulo.

25.1 Avances de la tecnología de bases de datos

La tabla 1 resume los pasos más destacados en la evolución de la tecnología de bases de datos durante las últimas tres décadas, aproximadamente. Las tres primeras columnas de la tabla corresponden aproximadamente a los tres periodos identificables. La última columna lista los avances que se esperan para el futuro. Analizaremos esta tabla fila por fila.

25.1.1 Modelos de datos

Los modelos de red y jerárquico surgieron en la década de 1960. Desde su introducción en 1970, el modelo relacional ha provocado un interés creciente debido a sus propiedades deseables, como son su base formal, su homogeneidad, su conjunto bien definido de operaciones algebraicas y sus lenguajes basados en el cálculo. Las desventajas del modelo relacional en términos de su poder expresivo semántico hicieron surgir el interés por los modelos semánticos. Este interés ha crecido desde finales de los años setenta, sobre todo con la aparición de los modelos de entidad-vínculo (ER) (Chen, 1976), de jerarquía semántica (Smith y Smith 1977) y de datos semántico (Hammer y McLeod 1981). En su modelo RM/1, Codd (1979) sugirió que el modelo de datos relacional necesitaba un mayor poder de expresión, para lo cual había que incorporar ciertas abstracciones de los modelos precedentes. El interés por los modelos semánticos continuó en los años ochenta. Hoy día, cuando los horizontes de las aplicaciones de bases de datos se encuentran en una clara fase de expansión (Sec. 25.2), la necesidad de contar con las abstracciones que estudiamos en el capítulo 21 se está sintiendo de manera aún más intensa.

Una de las principales tendencias en los próximos años será hacia los modelos de datos orientados a objetos (OO) y los sistemas de gestión de bases de datos orientadas a objetos (SCBDOO), los cuales tratamos con detalle en el capítulo 22. Los modelos de datos OO poseen un poder de expresión similar al de los modelos semánticos, pero los rebasan en el sentido de que modelan explícitamente el comportamiento. Se asegura que son más fáciles de implementar y usar cuando se trata de crear aplicaciones, debido a la modularidad integrada, el encapsulamiento y la reutilización de código.

Se ha propuesto la *lógica* como modelo de datos fundamental para los esquemas de representación relacionales y de otro tipo. El cálculo relacional (véase el Cap. 8) se basa en una rama de la lógica denominada cálculo de predicados de primer orden, de modo que ya tiene tiempo que la lógica se utiliza para caracterizar consultas relacionales. La relación entre la lógica y las bases de datos relacionales se pone de manifiesto en Gallaire *et al.* (1984). La lógica ofrece un formalismo para los lenguajes de consulta, el modelado de integridad, la evaluación de consultas, el tratamiento de valores nulos, el manejo de información incompleta, etc. La lógica nos lleva también a un entendimiento formal de la deducción en las bases de datos, que analizamos en el capítulo 24.

Es probable que en el futuro proliferen más los SCBD orientados a objetos, se utilice más la lógica para las bases de datos deductivas y se combinen la representación de conocimientos,

Tabla 25.1 Tendencias de la tecnología de bases de datos

[illegible]

los lenguajes de programación y los modelos de datos (Brodie *et al.* 1984; Bancilhon y Buneman 1990, v Atkinson y Buneman 1987).

Otra tendencia clara en la creación de SGBD comerciales es la de producir sistemas unificados con modelos de datos "híbridos". Un ejemplo de ello es el sistema UniSQL que maneja un modelo de datos completo orientado a objetos que es totalmente compatible con el modelo relacional. El producto ODB abierto de Hewlett-Packard, basado en el SGBD IRIS creado anteriormente, combina un modelo de datos orientado a objetos con uno funcional. Los proveedores de productos de SGBD convencionales ya han comenzado a implementar una interfaz relacional basada en SQL que permitirá la creación concurrente de aplicaciones relacionales. Tal es el caso de los SGBD en red como DMS 1100 de UNISYS e IDMS de Computer Associates, y ello se constituirá como una tendencia.

25.1.2 Hardware de bases de datos

Durante los últimos 30 años se ha venido dando una verdadera revolución en cuanto al crecimiento de las capacidades de almacenamiento de los computadores, la microminiaturización de los circuitos y el aumento en las velocidades de procesamiento. El costo de los componentes ha bajado de manera continua e impresionante. La cantidad de datos que se puede procesar en un intervalo de tiempo dado ha estado aumentando. Por añadidura, se están haciendo factibles bases de datos cada vez más grandes en equipo más pequeño. Hasta hace unos cuantos años, las bases de datos de tamaño apreciable sólo podían manejarse con macrocomputadores y minicomputadores. Ahora que contamos con equipo más potente, ya es posible implementar productos de SGBD importantes en estaciones de trabajo y computadores personales. Ya están disponibles, en estas plataformas, SGBD relacionales como DB2/2, ORACLE e INFORMIX, y SGBD orientados a objetos como ObjectStore y O2 (que vimos en el capítulo 22). La funcionalidad de los productos de base de datos de baja capacidad, como Lotus 1-2-3, la serie dBase, ACCESS y Paradox es objeto de una expansión continua, con equipo más potente que ofrece capacidades de ventanas y gráficos, así como velocidades más altas.

Aunque los costos de almacenamiento han declinado y las capacidades de la memoria principal de los sistemas han crecido de manera considerable, el rendimiento global de los sistemas de bases de datos sigue siendo una preocupación importante. Además, los adelantos en el software, que incorpora modelado de datos avanzado, han sido muy graduales (como veremos en la sección 25.2) en comparación con los de los procesadores y la arquitectura de E/S. Esto se debe a la diferencia entre las velocidades de acceso en los dispositivos de almacenamiento secundario y los de las unidades centrales de procesamiento, dando lugar a retrasos inevitables en el procesamiento de grandes volúmenes de datos almacenados en disco. Comparada con los avances en las velocidades de los procesadores (atribuida a nuevas tecnologías de lógica, así como a la disponibilidad del procesamiento en paralelo), la tasa de incremento de las velocidades de los dispositivos de almacenamiento secundario (principalmente discos) ha sido baja.

A fin de resolver estos problemas, se han hecho varias sugerencias alternativas para crear hardware especial enfocado a las funciones de gestión de datos. Estas alternativas, cuyo nombre genérico es **máquinas** (o computadores) **de bases de datos**, incluyen procesadores dorsales, dispositivos inteligentes (de lógica por pista), sistemas multiprocesadores, sistemas de memoria asociativa y procesadores de propósito especial. La Intelligent Database Machine (IDM), introducida por Britton-Lee en 1982 fue la primera máquina de bases de datos comercial. La DBC/1012 de Teradata es de reciente aparición; es el único computador de bases de datos multiprocesador disponible comercialmente que cuenta con una arquitectura

reconfigurable muy flexible. Y ya ha aparecido otra máquina de bases de datos llamada Teradata DBC/1012. Existe una implementación de sistema de gestión de bases de datos en paralelo del SGBD relacional ORACLE en el hardware de procesamiento en paralelo KSR (Kendall Square Research), así como en el procesador Ncube. En la actualidad es permanente la investigación sobre arquitecturas alternativas y la simulación de su desempeño. Su (1988) proporciona un excelente estudio sobre las investigaciones en torno a los computadores de bases de datos. Lamentablemente, la mayoría no ha pasado de ser diseños en papel o pequeñas implementaciones prototípicas, debido a la falta de recursos para construir hardware real a escala normal o al menos prototipos.

Debido a las exigencias del procesamiento de transacciones de alto volumen y el constante aumento en la velocidad de los procesadores (sobre todo en los supercomputadores), en el futuro se prestará mucha atención a las máquinas de bases de datos y a las arquitecturas de procesamiento en paralelo. Sin embargo, dada la experiencia acumulada en los últimos 15 años, es difícil predecir cuáles serían las soluciones económicamente viables y comercialmente factibles que podrían surgir, si es que lo hacen.

En lo tocante a nuevos equipos de almacenamiento, los dispositivos de almacenamiento óptico (que actualmente son del tipo de escribir una vez y leer un número indefinido de veces) se están modificando para convertirlos en dispositivos de lectura y escritura y pronto serán costeables económicamente. Su capacidad es enorme (del orden de 10^{12} bytes), lo que deja abierta la posibilidad de almacenar gigantescas cantidades de datos grabando la información actualizada sin borrar la información antigua. Ya existen en el mercado discos ópticos de lectura/escritura. Incluso con los discos ópticos de costo moderado disponibles en la actualidad, en los que sólo puede escribirse una vez y cuya velocidad de escritura física es del orden de una página por segundo, un disco utilizado con una carga de trabajo normal puede durar un año (Gait 1988). Esto nos permite vislumbrar varias posibilidades interesantes para el almacenamiento y la obtención de datos históricos (véanse las aplicaciones temporales en la sección 25.3.4).

25.1.3 Interfaces con el usuario

En el capítulo 1 describimos la amplia gama de personas que suelen utilizar los SGBD. Cuando estos sistemas aparecieron por primera vez a finales de los años sesenta, los usuarios finales no tenían acceso directo a las bases de datos. Sus interacciones se limitaban a la comunicación verbal de sus requerimientos a los programadores. Llenar formas fue uno de los primeros modos interactivos de introducción de datos adecuado para una población grande de usuarios finales, como los empleados de ventas o de aseguradoras. La mayor parte de la interacción seguía efectuándose a través de programas por lotes escritos en lenguajes por procedimientos.

Los lenguajes de consulta se popularizaron en la década de 1970. Ya estudiamos los lenguajes de consulta SQL (Cap. 7) y QBE y QUEL (Cap. 8) para el modelo de datos relacional. Resulta extraño que un producto de SGBD tan importante como IMS (véanse los detalles en el Cap. 11) prácticamente no cuente con un lenguaje de consulta. El modelo de datos de red también ha sobrevivido sin un apoyo de lenguaje de consulta sustancial, excepto por las órdenes de usuario especializadas, de alto nivel, que los clientes y proveedores puedan haber incorporado por su cuenta. La disponibilidad de estaciones de trabajo gráficas y los sistemas de ventanas han elevado tremendamente el potencial para crear interfaces no sintácticas con los SGBD, en las que el usuario no está obligado a usar una sintaxis específica.

Es muy probable que las interfaces que utilizan ampliamente ventanas, menús desplegables e iconos se desarrollen y adquieran capacidades más avanzadas.

En casi todos los SGBD se contempla la adición de recursos llamados de **consulta por formas**. Con ellos el usuario podría invocar consultas "parametrizadas" a las que se suministrarán parámetros durante la ejecución; así, una consulta predefinida se recompilaría o reinterpretaría y acto seguido se ejecutaría. Otros recursos para los usuarios incluyen las interfaces basadas en iconos, en las que el usuario toca iconos o imágenes en la pantalla para formular una consulta, y las interfaces basadas en ratón, en las que un *ratón* o una *pantalla sensible al tacto* permite mover el cursor durante la especificación de una consulta. Algunas interfaces tienen acceso a datos reales y presentan los valores de la base de datos en una ventana desplazable para que puedan usarse como constantes en las consultas.

De un tiempo acá se han explorado las interfaces de **lenguaje natural**. Hay tres enfoques básicos para interpretar el lenguaje natural:

- **Extracción de palabras clave o comparación de patrones:** En los sistemas de palabras clave, el programa relaciona palabras del lenguaje natural con campos específicos de una base de datos, y el creador de la aplicación define los vínculos. Sin embargo, la comparación de patrones sin una base gramatical no tiene sino una utilidad limitada.
- **Análisis sintáctico:** El análisis sintáctico convierte una frase expresada en un lenguaje natural, potencialmente ambigua, a un formato interno que deberá representar con precisión la consulta que desea hacer el usuario. Hay dos posibles variaciones de este enfoque: una basada en una gramática del lenguaje natural y la otra basada en la semántica del lenguaje en términos de un léxico y una serie de reglas de producción. El sistema LADDER (Hendrix, 1978) se valió de una gramática semántica para tener acceso a bases de datos de la Marina de los Estados Unidos. El sistema INTELLECT de Harris (1978), que ahora es un producto comercial, también se basa en reglas gramaticales. El sistema Rendezvous de Codd (1978) usaba un léxico y convertía consultas expresadas en lenguaje natural a expresiones del cálculo relacional con base en una gramática de frases.

- **Transformación de consultas:** Se utiliza una base de conocimientos o un "modelo del mundo" con representaciones canónicas de enunciados de un cierto dominio de aplicación, junto con conocimientos lingüísticos y conocimientos de transformación, para transformar consultas expresadas en lenguaje natural en un nivel conceptual a consultas de base de datos expresadas en un lenguaje específico. El sistema KID (Ishikawa 1987) es un ejemplo de este enfoque.

Los sistemas de lenguaje natural actuales tienen el defecto de que representan un gasto extra de proceso excesivo, requieren la construcción de un léxico específico para cada base de datos y permiten ambigüedades en la especificación de consultas. En consecuencia, es posible que un usuario no se dé cuenta de cómo el sistema ha interpretado realmente una consulta, y el sistema no tiene forma de predecir con exactitud lo que el usuario desea. Casi siempre es igual de fácil operar las interfaces basadas en iconos y ratón, y resultan mucho más económicas.

La tecnología de *reconocimiento* y *comprensión de la voz* todavía no ha avanzado lo suficiente como para que la entrada hablada sea viable en algún sentido importante en el futuro cercano. No obstante, este enfoque tiene un enorme potencial para abrir las puertas de las bases de datos a personas de todas las edades y todos los niveles de educación, y para poner al alcance de las masas la instrucción asistida por computador apoyada en bases de datos.

25.1.4 Interfaces con programas

Incluso hoy día, el grueso de la programación en el mundo comercial, que puede representar entre el 70 y el 80% (una estimación aproximada) del procesamiento real de bases de datos, se efectúa en lenguajes de programación convencionales, principalmente en COBOL, PL/I y (ahora) C. Estos lenguajes, junto con FORTRAN, lenguaje ensamblador y ALGOL, han sido el sostén principal de las aplicaciones de bases de datos. El lenguaje C++ ha entrado en escena hace poco y está ganando terreno rápidamente, sobre todo en los SGBD orientados a objetos.

La década de 1970 fue testigo de la aparición de lenguajes de consulta incorporados en lenguajes de programación para producir programas de aplicación a gran escala. Vimos ejemplos de SQL incorporado en el capítulo 7.

Una divergencia respecto a esta tendencia apareció en dos áreas. En primer término, los lenguajes de cuarta generación (4GL: *fourth-generation languages*, un término acuñado comercialmente y sin una definición precisa) se popularizaron en los años ochenta. Éstos permiten al usuario introducir una especificación de alto nivel de una aplicación. A continuación, el sistema (o una herramienta) genera automáticamente el código de la aplicación. Por ejemplo, INGRES y la mayor parte de los SGBD ofrecen una interfaz de "aplicación por formas", en la que el diseñador crea una aplicación empleando formas que aparecen en la pantalla, en vez de escribir un programa. Sin embargo, los lenguajes de cuarta generación no han tenido una aceptación uniforme para la gestión de datos de propósito general y para la mayor parte de las aplicaciones de generación de informes. El manejo integrado de bases de datos con que cuentan es mínimo.

Otro avance importante en la década de 1980 ha sido la estandarización de SQL, primero las normas ANSI SQL y SQL 89 y, en fechas más recientes, SQL2 y SQL3. El capítulo 7 adopta la sintaxis de SQL2. Hoy día, casi todos los proveedores de SGBD procuran ajustarse a estas normas, lo que facilitará la interoperabilidad de estos sistemas. SQL3 estaba todavía en la etapa de borrador en el momento de escribirse este libro; combina muchas características orientadas a objetos (véase el Cap. 22) con SQL. Muchos proveedores de SGBD no relacionales han optado por incluir interfaces de programación SQL en sus SGBD; por ejemplo, en el sistema IDMS, basado en el modelo de red (véase la Sec. 10.7), y en el sistema ADABAS de Software AG.

El empleo de lenguajes de programación lógica para la gestión de bases de datos deductivas⁷ (véase el Cap. 24) no ha tenido todavía un impacto considerable. Una razón es la falta de sistemas comerciales a gran escala. Ya estudiamos el único que existe, LDL, en la sección 24.7. Es de esperar que la tendencia será hacia los sistemas de programación integrada, en los que la inferencia y recursión de los lenguajes de programación lógica se acoplará con la manipulación eficiente de hechos en el SGBD. Con la fusión de los modelos de datos orientado a objetos y relacional, o con la integración de una interfaz relacional a los SGBD de red, los sistemas futuros seguirán ofreciendo múltiples opciones de interfaz de programación para la creación de aplicaciones.

25.1.5 Presentación y exhibición

Los dispositivos de salida originales para los sistemas de cómputo eran impresoras de líneas y perforadoras de tarjetas. Casi todas las salidas se presentaban como informes de línea por

⁷No es nuestro propósito aquí definir formalmente las bases de datos deductivas. Baste con decir que una base de datos convencional se vuelve deductiva al añadirsele una teoría con ciertos axiomas y leyes deductivas. Véase Gallaire et al. (1984).

línea o como valores calculados perforados en tarjetas que debían procesarse tiempo después. El lenguaje RPG (*Report Generation Language*: lenguaje de generación de informes) fue el principio de una tendencia hacia la conversión de la generación de informes en una actividad especializada que se podía especificar en términos de órdenes de alto nivel para crear informes.

Hoy día, los generadores de informes son un componente estándar de casi todos los SGBD. Como ejemplos podemos citar el generador de informes de INGRES, el paquete CURPRIT de IDMS y los procesadores lsr y rpt de UNIFY. Hay lenguajes especiales para definir informes en los que el usuario especifica el formato, el espaciado, la paginación, los niveles de los totales, las cabeceras, la justificación, etcétera.

Con el acceso interactivo a las bases de datos para una gran variedad de usuarios, se hizo posible producir informes mediante la invocación de un generador de informes o la especificación de informes a través de formas (por ejemplo, Report by Forms de INGRES). Con la aparición de los computadores personales, los usuarios se han acostumbrado a recibir una gran cantidad de información en la pantalla. Casi todos los paquetes de SGBD ofrecen salidas de gráficos de negocios de diversas formas, incluidas curvas xy (gráficas de dispersión) empleando regresión lineal, gráficas de barras, diagramas de pastel y gráficas de líneas (con aproximaciones a curvas mediante segmentos de rectas que conectan series de puntos). Gracias a los gráficos en color se han generalizado presentaciones que, además de ser estéticamente agradables, son impresionantes. Estos tipos de presentaciones requieren el procesamiento de las salidas de las consultas por parte de un gestor de presentación.

Al integrar la tecnología de imágenes al procesamiento de bases de datos, y al almenar la voz digitalizada como datos, algunos de los sistemas experimentales actuales han podido producir salidas con imágenes y voz. El almacenamiento de imágenes digitalizadas ya es cosa de todos los días, y es posible que en unos cuantos años esté disponible en el mercado equipo especializado para la síntesis de voz.

25.1.6 Naturaleza del procesamiento

La tecnología de bases de datos se introdujo originalmente como una respuesta a los problemas del procesamiento de archivos y a sus desventajas asociadas (véase la Sec. 1.3). Las aplicaciones principales se dedicaron al procesamiento de datos de negocios: control de inventarios, nóminas, contabilidad general, facturación, proceso de pedidos, informes de ventas, etcétera.

Durante los años setenta se comenzó a aplicar la tecnología a diversos dominios de distintas disciplinas. Se generaron aplicaciones de alto nivel que resumían los datos a fin de generar información adecuada para la planificación táctica y estratégica. Los SGBD confiables y seguros se convirtieron en una fuente de información centralizada con la cual podía interactuar un gran número de aplicaciones/usuarios geográficamente separados. Esto ha sido posible gracias a los avances en la tecnología de comunicación de datos, entre ellos las redes de comunicaciones de banda ancha y por satélite. Los principales usuarios de la tecnología de SGBD en la actualidad la usan para el procesamiento de transacciones.

Las líneas aéreas, las principales cadenas hoteleras, las compañías de seguros, los bancos, las cadenas de almacenes de venta al detalle, los concesionarios de automóviles y aparatos domésticos, y otros, funcionan actualmente con procesamiento de transacciones inmediato y confiable y actualizaciones en línea en tiempo real. Hoy día es posible reservar un asiento en un teatro de Broadway desde un remoto pueblo en Oregon. Estas aplicaciones continuarán exigiendo cada vez más a la tecnología de procesamiento de transacciones en línea.

25.1.7 Tendencias actuales de la tecnología

La tendencia de la tecnología para el futuro cercano estará dominada por los siguientes temas:

- **Entornos heterogéneos y distribuidos:** Las grandes bases de datos centralizadas serán sustituidas por bases de datos distribuidas. Estas bases de datos se implementarán con SCBD convencionales de red y jerárquicos, con los SCBD relacionales actuales, o con los SCBD orientados a objetos cuya popularidad va en aumento. La arquitectura "cliente-servidor" también está adquiriendo gran popularidad y ofrecerá a los usuarios diversas opciones respecto a la funcionalidad de los SCBD y la ubicación de los datos.
- **Sistemas abiertos:** Varios esfuerzos de estandarización, como SQL Access Group (grupo de acceso SQL), ANSI/X3H7, Object Data Management Group (ODMG: grupo de gestión de datos de objetos) de Object Management Group (OMG), ODBS de Microsoft, IDAPI y ODAPI (Borland) están tratando de mejorar la "apertura" de los sistemas y aplicaciones de bases de datos futuros. Los sistemas que se ajusten a estas normas podrán comunicarse sin dificultad entre sí. Las normas SQL2 y SQL3 desempeñarán un papel importante a este respecto. La tendencia será hacia el diseño de aplicaciones globales que se ejecutarán en los sistemas de los clientes y obtendrán datos de diversos servidores con protocolos estandarizados de acceso a datos.
- **Mayor funcionalidad:** La funcionalidad de los SCBD irá más allá de las operaciones convencionales de definir, almacenar, acceder, consultar e informar. Se incluirán la capacidad deductiva descrita en el capítulo 24, el procesamiento de metadatos y una mejor especificación del comportamiento a través de métodos (Cap. 22). Además, la capacidad "activa" (que describiremos en la Sec. 25.3) para hacer más responsables las bases de datos, la gestión de bases de datos de multimedia, el mejor procesamiento de datos temporales y espaciales, los SCBD especiales para usuarios científicos, etc., probablemente dominarán el desarrollo de la tecnología de bases de datos en los años venideros. Examinaremos algunas de estas áreas y sus necesidades especiales en la sección 25.3.

- **Gestión de bases de datos en paralelo:** Es probable que se ofrezca a escala comercial una nueva clase de sistemas de gestión de bases de datos basada en las arquitecturas MIMD (múltiples instrucciones, múltiples caminos de datos), como el hipercubo IPSC de Intel o la máquina de Kendal Square Research. Los trabajos actuales implican nuevos algoritmos para asignación de datos en las estrategias de consultas paralelas y un aprovechamiento eficaz del sistema global a fin de minimizar el costo por transacción. Casi no se ha trabajado sobre la forma de hacer paralela una serie existente de aplicaciones de bases de datos para aprovechar los SCBD paralelos del futuro.

25.2 Nuevas aplicaciones de las bases de datos

Las aplicaciones de la tecnología de bases de datos están en continua expansión. Como los datos son centrales en cualquier sistema de información, conforme crece la disponibilidad de computadores cada disciplina llegará a tener su propio conjunto único de aplicaciones. No es nuestro propósito examinar con detalle tales listas de aplicaciones. Queremos identificar las principales categorías de aplicaciones que actualmente se cree presentan un buen potencial además de un desafío importante para la tecnología de bases de datos. Se destacarán las características de modelado especiales de cada área de aplicación.

25.2.1 Diseño de ingeniería y fabricación

El difícil objetivo del diseño y la fabricación integrados al computador requiere el manejo eficaz de la información de diseño y fabricación. Este tema abarca subáreas denotadas por varios acrónimos: CAD (diseño asistido por computador: *computer-aided design*), CAM (fabricación asistida por computador: *computer-aided manufacturing*), CAE (ingeniería asistida por computador: *computer-aided engineering*) y CIM (fabricación integrada por computador: *computer-integrated manufacturing*). Un enfoque integrado para manejar la información de fabricación requiere una representación y manipulación compatibles de la información en las diferentes fases del ciclo de vida del producto. Esto incluye aplicaciones de negocios como son pronósticos de ventas, procesamiento de pedidos, planificación de productos, control de producción, control de inventarios y contabilidad de costos; diseño e ingeniería del producto junto con la planificación de requerimientos de materiales; aplicaciones relacionadas con la fabricación, como la planificación y el control de recursos de fabricación, y aplicaciones de la tecnología de grupos para la clasificación de componentes, robótica y control del procesamiento de fabricación (como la NC: programación de control numérico) (Bray 1988).

Las ventajas estándar de la tecnología de bases de datos ayudan a integrar estas áreas (véanse las Secs. 1.6 y 1.7). Un buen número de investigaciones se dedican a estos problemas, tanto del diseño como de la fabricación, y una de las prioridades actuales es la creación de SCBD especialmente adaptados a las aplicaciones de diseño asistido por computador. La figura 25.1 muestra la diversidad de los datos pertinentes y el apoyo que puede darse a varias aplicaciones en el entorno de diseño una vez capturados estos datos.

El *diseño de ingeniería* es un proceso de exploración e iteración. La actividad de diseño de ingeniería para sistemas complejos como los aviones o los automóviles lo efectúan equipos de proyecto, y el diseño de un componente o subsistema evoluciona continuamente bajo un conjunto de pautas de diseño, limitaciones de recursos y restricciones de diseño. Cada cierto tiempo, los diseños se verifican contra otros diseños que evolucionan de manera independiente, y al final se almacenan diseños o versiones "permanentes".

Papel de la gestión de bases de datos en CAD. Dos importantes áreas de investigación son el diseño de sistemas electrónicos VLSI (*very large-scale integration*: integración a muy grande escala) y el diseño de estructuras y sistemas mecánicos. Un desafío fundamental en el diseño mecánico es el **modelado geométrico**, que se refiere a la representación de la forma física de los componentes mecánicos. La forma de los componentes constituye un hilo común a través del ciclo de diseño, análisis y fabricación de dichos componentes. Los diferentes sistemas CAD emplean diferentes técnicas para el modelado geométrico, incluidos el modelado con armazones de alambre, el modelado de superficies y el modelado tridimensional. Resulta una experiencia traumática para el usuario transportar datos entre diferentes sistemas CAD o de un sistema CAD a uno CAM. El National Bureau of Standards (Oficina nacional de normas) de Estados Unidos definió una interfaz o formato de datos común neutral llamado Initial Graphics Exchange Specification (*especificación de intercambio de gráficos inicial*, IGES 1983) para facilitar este intercambio de datos. IGES define entidades de diseño estándar para dibujos en dos dimensiones y objetos en tres dimensiones, así como diagramas eléctricos. En el caso de los cuerpos tridimensionales, son más comunes dos tipos de representaciones, denominados representación de frontera y geometría constructiva de cuerpos.

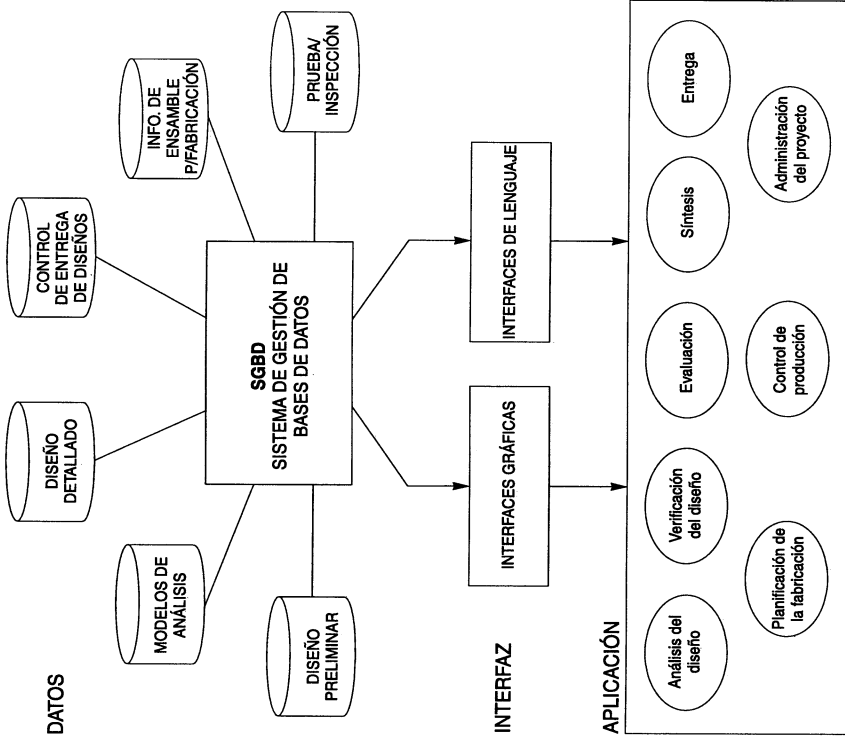


Figura 25.1 Empleo de un sistema de gestión de bases de datos en el diseño y las aplicaciones de ingeniería.

La ingeniería estructural se ocupa de problemas que van desde el diseño de armaduras para edificios hasta el diseño de plataformas de lanzamiento complejas para vehículos espaciales. Ya sea que se trate de edificios, ensambles mecánicos, puentes o naves espaciales, las estructuras están limitadas por geometrías y exigen que se seleccione un conjunto óptimo de miembros que satisfaga las restricciones de peso y los límites de diseño, y ofrezca fiabilidad bajo diferentes cargas. Las bases de datos pueden desempeñar un papel importante al almacenar información en un centro para usarla con herramientas analíticas (como el análisis de elementos finitos), gráficos (por lo regular gráficos tridimensionales en los sistemas CAD), simulaciones y algoritmos de diseño óptimo. En la ingeniería química se usan las bases de datos para el diseño de plantas y el control de procesos. Son dos las principales razones para usar un sistema de bases de datos centralizado para los datos de diseño:

- Parte de un diseño se puede sintetizar, analizar, coordinar y documentar al tiempo que equipos de proyecto individuales trabajan con diferentes partes del diseño.

- Es posible verificar e imponer automáticamente restricciones relacionadas con normas, diseños y especificaciones idénticos, así como otras propiedades físicas, estilos de diseño e interrelaciones topológicas.

Los datos de diseño de ingeniería son difíciles de capturar y representar con los modelos de datos y con SGBD convencionales (Caps. 6-11) por las siguientes razones:

- Los datos de diseño de ingeniería contienen una colección heterogénea de objetos de diseño. Los modelos de datos clásicos manejan colecciones homogéneas.
- Los SGBD clásicos son buenos para manejar datos con formato (escalares), cadenas cortas y registros de longitud fija. Los diseños digitalizados son cadenas largas; tienen registros de longitud variable o información textual; a menudo contienen vectores y matrices de valores.
- Las interrelaciones temporales y espaciales (véase la Sec. 25.3.4) son importantes en los diseños para las operaciones de disposición, colocación y ensamble.
- Los datos de diseño se caracterizan por un gran número de tipos, cada uno con un número reducido de ejemplares. Las bases de datos convencionales se enfrentan a una situación diametralmente opuesta.
- Los esquemas *evolucionan constantemente* porque los diseños pasan por un largo período de evolución.
- Las transacciones en las bases de datos de diseño son de larga duración; un diseñador puede "sacar" un objeto de diseño y trabajar con él durante varias semanas antes de devolverlo (en su forma modificada) a la base de datos. Además, las actualizaciones son de largo alcance debido a las interrelaciones topológicas, las interrelaciones funcionales, las tolerancias, etc. Es probable que un cambio afecte un gran número de objetos de diseño.
- Es necesario conservar las versiones anteriores y crear nuevas versiones del mismo objeto. Hay que mantener una bitácora de diseño para rastrear la evolución de un diseño y posiblemente dar marcha atrás en él.

- Hacer permanente un diseño, entregarlo a producción, archivarlo, etc., son funciones especializadas en una base de datos de diseño.

- Los datos de diseño no deben duplicarse en niveles de diseño inferiores; dado que los elementos de diseño (como las computas en los circuitos eléctricos y las tuercas y pernos en el diseño mecánico) se utilizan de manera muy repetitiva, se requiere un control de redundancia para suprimir ciertos atributos comunes. Por ejemplo, cuando se usan pernos idénticos en dos lugares, se almacenan las coordenadas de posición pero no se repiten los demás atributos. En tales casos, se puede mantener una biblioteca de objetos de diseño.

Debido a estas exigencias, se están proponiendo nuevos enfoques de modelado de datos para controlar los diseños VLSI, así como los diseños mecánicos. Los modelos orientados a objetos son los preferidos porque poseen las siguientes características (si se desea un análisis detallado, véase Spooner 1986 y Katabchi 1986):

- *Modelo común.* Se puede establecer una correspondencia uno a uno entre el minimundo del diseñador y los objetos de la base de datos.

- **Interfaz uniforme:** Todos los objetos se tratan de manera uniforme, teniendo acceso a ellos a través de métodos (u operaciones definidas por el usuario).
- **Manejo de objetos complejos:** Los modelos orientados a objetos para los diseños de ingeniería deben permitir la creación de objetos arbitrariamente complejos que impliquen jerarquías y retículas (Batory y Buchmann 1984).
- **Ocultación de información y manejo de abstracciones:** El mecanismo de abstracción puede proporcionar las características externas esenciales de los objetos para el diseño al tiempo que oculta la representación interna o los detalles de implementación. Es fácil manejar la generalización y la agregación (véase Ahmed y Navathe 1991).
- **Creación de versiones:** Un objeto de diseño puede evolucionar a través de una serie de versiones; también puede tener diferentes representaciones. Es necesario tratar de manera independiente un objeto genérico y sus versiones. Al mismo tiempo, no debe exagerarse el uso de la operación "crear versión" en vez de una actualización. ORION (Chou y Kim 1986) maneja versiones transitorias y versiones de trabajo. Ahmed y Navathe (1991) define un esquema de versiones basado en las propiedades intrínsecas y extrínsecas de los objetos de diseño. Se crea una versión cuando las propiedades intrínsecas no cambian, pero sí lo hacen las propiedades extrínsecas. El esquema de versiones propuesto evita la proliferación innecesaria de versiones.
- **Modularidad, flexibilidad, extensibilidad y adaptabilidad:** Las bases de datos orientadas a objetos apoyan la evolución de los esquemas mejor que los modelos convencionales; resulta sencillo añadir nuevos objetos u operaciones, y modificar o eliminar los más antiguos.

Entre los primeros trabajos en esta dirección se cuentan un proyecto puesto en práctica en la University of Southern California (Afsarmanesh *et al.* 1985), el sistema ORION (Kim *et al.* 1987) en Microelectronics and Computer Technology Corporation, y un trabajo en la University of California, Berkeley (Karz 1985). En fechas más recientes Hardwick y Spooner (1989) crearon el sistema ROSE (Relational Object System for Engineering: sistema relacional de objetos para ingeniería) en el Rensselaer Polytechnic. Es un SGBD estructuralmente orientado a objetos que maneja agregación, generalización y asociación. Tiene una arquitectura abierta con recursos para proveer herramientas frontales y dorsales.

Hay informes sobre extensiones de los modelos relacional y de red para manejar datos de CAD, pero no han tenido mucho éxito (Wiederhold *et al.* 1982; Stonebraker *et al.* 1983). Las transacciones largas y el control de concurrencia de las transacciones de diseño siguen siendo temas de investigación.

Otro avance importante es la estandarización de los datos de productos para fines de intercambio entre proveedores, diseñadores y fabricantes. La norma PDES (Product Data Exchange using STEP: intercambio de datos de productos empleando STEP, el cual es una norma ISO para intercambiar datos de modelos de productos) está atrayendo muchos seguidores (PDES Inc. 1991). Se aplica en diversas industrias, como la de la construcción, la de componentes electrónicos y en la arquitectura. El lenguaje que la acompaña, llamado EXPRESS, también está adquiriendo gran popularidad como lenguaje descriptivo para especificar componentes, identificar diversos tipos de restricciones sobre ellos y proveer una forma uniforme de representar objetos de diseño para un análisis/síntesis posterior. Las normas PDES/STEP y EXPRESS promueven el compartimiento y la integración de los datos para el diseño de productos. Es muy probable que estos dos avances lleven a una nueva generación de sistemas y herramientas en el diseño de ingeniería y la fabricación.

También hace falta trabajar más sobre el modelado de los datos de diseño debido a sus peculiaridades. Una dirección prometedora implica la separación de la estructura y la función en sistemas físicos complejos de modo que cada una se pueda representar y analizar de manera independiente (Navathe y Cornelio 1990). Este enfoque facilita aún más el modelado de la dinámica de estos sistemas mediante el enfoque de base de datos activa (véase la Sec. 25.3.1) y ayuda en la simulación (Cornelio y Navathe 1993).

25.2.2 Sistemas de oficina y sistemas de apoyo para la toma de decisiones

La automatización del trabajo de oficina ha sido una de las áreas de aplicación de los sistemas de información con más rápido crecimiento (Ellis y Nutt 1980). El creciente interés y actividad en los **sistemas de información de oficina** (OIS: *office information systems*) se hizo evidente cuando la Association for Computing Machinery sacó a la luz una publicación periódica llamada *ACM Transactions on Office Information Systems* (Transacciones de la ACM sobre sistemas de información de oficina) en 1983. Tiempo después, al observar que los sistemas de información de oficina son sólo un tipo específico de sistemas de información, la publicación cambió su título a *Transactions on Information Systems* en 1990. La tecnología de bases de datos tiene un impacto importante sobre el trabajo de oficina porque gran parte de él puede clasificarse como *trabajo programable*, pues los acontecimientos son predecibles y las respuestas son conocidas. Los computadores y, en particular, los sistemas de base de datos pueden influir considerablemente sobre este tipo de labores. El otro extremo es el *trabajo creativo emergente*, según la taxonomía de Ciborra *et al.* (1984), en el que los acontecimientos son impredecibles y las respuestas desconocidas. El trabajo de oficina de la alta gerencia, los comerciantes de valores internacionales y similares pertenece a esta categoría y requiere la ayuda de sistemas de apoyo a la toma de decisiones. En primer término señalaremos las diferencias entre los sistemas de información y los OIS; luego veremos brevemente un modelo basado en datos de un OIS, y por último haremos algunos comentarios sobre los sistemas de apoyo para la toma de decisiones.

Sistemas de información de oficina vs. aplicaciones convencionales de bases de datos.[†] Al igual que las aplicaciones de CAD, las aplicaciones de OIS exigen más de un SGBD y centralizan la información en un entorno de oficina dado. Esto se debe a las siguientes características de las aplicaciones de OIS:

- **Riqueza semántica:** Los datos de oficina tienden a ser ricos semánticamente y requieren apoyo para mensajes no estructurados, cartas, textos, anotaciones, cadenas de direcciones de envío, comunicaciones orales, etc. En una oficina hay *agrupaciones de información estereotípica*, como las cartas de negocios o los informes de avances trimestrales con formatos estandarizados.
- **El factor tiempo:** El factor tiempo y la restricción temporal se deben modelar para capturar los siguientes tipos de información: el tiempo total de recorrido de un documento a lo largo de un camino; la duración de las actividades, calendarios y planes; el tiempo de respuesta permisible para contestar un mensaje; generación automática de recordatorios, etcétera.

[†]Esta sección se deriva en gran parte de Bracchi y Pernici (1984, 1987).

- **Falta de estructura:** Las actividades de oficina tienden a estar mucho menos estructuradas que otros sistemas de información. Las instrucciones para realizar tareas son incompletas e irregulares; se requieren una comunicación y diálogo constantes.
- **Alta interconectividad:** En comparación con una celda de trabajo de fabricación que utilice herramientas mecánicas, una oficina por lo regular representa un grupo mucho más complejo de elementos, cada uno de los cuales realiza diversas funciones y está interconectado en múltiples direcciones con elementos de diferentes tipos. Por tanto, es difícil modelar con precisión el flujo de trabajo en una oficina, y la automatización y las mejoras en la productividad son difíciles de lograr a través de una metodología de diseño de oficinas bien definida.
- **Restricciones y evolución de las oficinas:** Dado que una oficina es un grupo de seres humanos, está sujeta a evolución constante en la que las autoridades y obligaciones de trabajo de los individuos suelen cambiar. Por tanto, resulta difícil modelar de manera definitiva todas las restricciones.
- **Interfaz interactiva:** Los OIS son altamente interactivos. Incluso el empleado de más bajo nivel de una oficina debe poder comunicarse con un OIS. Los grupos de usuarios son diversos y abarcan toda la gama desde simples hasta avanzados. Esto hace que el diseño de interfaces constituya un desafío.
- **Filtrado de la información:** La mayor parte de las oficinas tiene una jerarquía inherente. Es necesario filtrar y resumir la información en un estilo piramidal dentro del sistema. Las transacciones de bajo nivel deben resumirse antes de pasar al siguiente nivel de la gerencia. La agregación de los elementos de información debe efectuarse automáticamente en diferentes intervalos previamente especificados y controlados por el usuario.
- **Prioridades, planificación, recordatorios:** Todas estas son características importantes de diferentes elementos del trabajo; en una oficina se generan constantemente diversas interrupciones que deben manejarse.

Queda claro que la automatización de una oficina no es tarea sencilla. Una base de datos que sea el corazón de un OIS debe ser capaz de satisfacer todos los requerimientos precedentes. Debido a estos factores, vemos "islas de mecanización" en las oficinas actuales, en vez de una oficina totalmente automatizada. Se han propuesto muchas metodologías de modelado y diseño conceptual para los OIS; como ejemplos están OFFIS (Konsynski *et al.* 1982), OAM (Sirbu *et al.* 1981) y MOBILE-Burotique (Dumas *et al.* 1982). Las investigaciones recientes sobre sistemas de información de oficina se han concentrado en modelar el flujo de trabajo en la oficina, definiendo los requerimientos de bases de datos y estableciendo el procesamiento de consultas específicas para el entorno de la oficina. Un ejemplo es el proyecto TODOS (Pernici 1989), que significa TOols for Designing Office Systems (herramientas para diseñar sistemas de oficina). Se espera que la tecnología de bases de datos de multimedia (véase la Sec. 25.3.2) desempeñará un papel importante en los sistemas de oficina (Masunaga 1987, Woelk 1987, Bertino 1988).

Modelo basado en datos para un sistema de información de oficina. Los modelos basados en datos agrupan los datos en maneras que se parecen a las formas impresas de una oficina. El proyecto Office-By-Example de IBM Research (Zloof 1982) creó un lenguaje llamado

Office By Example (OBE: Oficina por ejemplos) que es una extensión del lenguaje QBE descrito en el capítulo 8. OBE modela diversos objetos con el paradigma básico de las tablas relacionales: formas, informes, documentos, listas de direcciones, listas de mensajes, menús, etc. El sistema contempla tipos de datos como imágenes, texto y tiempo. Las funciones especificadas sobre estos objetos incluyen el procesamiento de textos, las consultas y el disparo automático de actividades (por ejemplo, enviar una carta cuando un pago está vencido más de n días, donde n es un parámetro introducido durante la ejecución o se toma de alguna tabla). La autorización, la comunicación y el envío hacia adelante de mensajes y la manipulación de documentos son ejemplos de otras funciones que se manejan a través de la misma interfaz. El aspecto principal de los modelos de oficina basados en datos es que representan la oficina desde el punto de vista de los objetos que los empleados de la oficina manipulan. Una vez más, el enfoque orientado a objetos parece ser prometedor. Están apareciendo productos específicos para procesamiento de documentos, correo electrónico de multimedia, comunicación de audio/vídeo y cosas similares para entornos de usuario tipo PC-Windows. Un ejemplo es el sistema NEWWAVE de Hewlett-Packard.

Sistemas de apoyo a la toma de decisiones (DSS: decision support systems). Los sistemas de oficina y de procesamiento de datos de negocios en general ofrecen varios tipos de apoyo a la toma de decisiones de los gerentes de alto nivel:

- **Análisis de datos de la oficina:** Los datos deben presentarse al usuario en el nivel adecuado de detalle, mediante la presentación correcta después de un filtrado apropiado.
- **Control del estado del sistema:** Un OIS debe contar con funciones para determinar su propio estado y evaluarlo. Cualesquier discrepancias o condiciones excepcionales que requieran revisión deberán comunicarse a los individuos correctos.
- **Manejo de herramientas analíticas para la toma de decisiones:** Además de tomar decisiones *ad hoc*, los gerentes deben tomar decisiones estratégicas y de planificación de negocios a largo plazo. Deben considerar y analizar opciones alternativas. Un DSS debe conocer las alternativas así como los modelos requeridos para efectuar una optimización o llegar a opciones "inteligentes".
- **Manejo de cambios en el diseño de la organización y del sistema de oficina:** El DSS deberá contar con algunas funciones que permitan reestructurar la organización o modificar el flujo de información sin perturbar las operaciones normales.

En la actualidad, los DSS se están diseñando con un SGBD como componente central. En general, un DSS permite al usuario realizar funciones de control y vigilancia; también debe permitirle imponer decisiones de política (restricciones) y acciones (procedimientos). Para hacer posible una automatización parcial de la toma de decisiones en una oficina, se puede recurrir a un sistema de base de datos "activa" (véase la Sec. 25.3.1), donde las acciones pueden implicar la generación de mensajes, avisos, cartas, etc., o ser más completas en términos de imponer políticas que abarquen el conjunto completo de funciones y bases de datos en la oficina. Se han propuesto varios sistemas, pero la bibliografía es demasiado amplia como para resumirla aquí.

25.2.3 La iniciativa del genoma humano

Introducción y objetivos del proyecto. La iniciativa del genoma humano (HGI: Human Genome Initiative) es un buen ejemplo de una tendencia en biología que le está dando

un nuevo aspecto, la de una ciencia analítica computacional. La iniciativa del genoma humano es un esfuerzo internacional encaminado a determinar la ubicación de los 100 000 genes que, según se estima, componen el genoma humano completo. HGI es una monumental aventura científica comparable con los esfuerzos realizados en el siglo XIV para dibujar el mapa más completo y detallado posible del planeta Tierra, dadas las herramientas de aquellos tiempos, cuando se pensaba que la Tierra era plana. Las implicaciones científicas, médicas y económicas de la HGI son profundas. En última instancia, este proyecto prepararía el camino para una decodificación rutinaria de los genes que hacen posible la vida, además de ofrecer una mejor comprensión de las enfermedades que afligen a la humanidad.

El objetivo más ambicioso del proyecto del genoma es completar el mapa más detallado: una secuencia de referencia del genoma humano completo. La HGI es un proyecto a largo plazo, con una duración esperada de 25 a 40 años. Implica tres retos computacionales clave: análisis de secuencias, almacenamiento y obtención de información, y predicción de la estructura de las proteínas. Estos elementos se clasifican de la siguiente manera:

- *Problema de análisis de secuencias:* Dado un conjunto de secuencias de DNA, se requieren algoritmos eficientes de comparación de alineaciones capaces de manejar de manera elegante inserciones, eliminaciones, sustituciones e incluso huecos en la serie de elementos de una secuencia.
- *Problema de almacenamiento y obtención:* Almacenar y obtener de manera eficaz los datos, y hacer a la información fácilmente accesible para los usuarios distribuidos al tiempo que se manejan con eficacia los errores, conflictos y actualizaciones implica un reto considerable; tanto más porque la velocidad a la que se estará generando información al comenzar el nuevo siglo será de aproximadamente 1600 millones de pares de bases al año.
- *Problema de predicción de la estructura del DNA y de las proteínas:* Dentro de las células, la mayor parte del DNA está asociado a proteínas que, entre otras cosas, influyen en la tensión torsional de la doble hélice del DNA. Si la tensión torsional es negativa, la hélice de DNA puede asumir estructuras alternativas. Secuencias específicas favorecen tales estructuras, y la predicción de las probabilidades de estructuras alternativas para una secuencia dada es un problema nada trivial. Una vez que la célula sintetiza una proteína, la cadena proteínica se pliega de acuerdo con las leyes de la física para adoptar una forma especializada, con base en las propiedades particulares y el orden de los aminoácidos. Se espera que el problema del plegado de las proteínas presentará un reto para nuestros más potentes supercomputadores en el futuro previsible.

En los organismos multicelulares, el DNA generalmente se encuentra en forma de dos hilos lineales enrollados en forma de una doble hélice. Un hilo de DNA es una cadena polimérica formada por nucleótidos, cada uno de los cuales consiste en una base nitrogenada, un azúcar (desoxirribosa) y una molécula de fosfato. El acomodo de los nucleótidos a lo largo del esqueleto del DNA se denomina secuencia de DNA. En estas secuencias entran cuatro bases nitrogenadas: adenina (A), guanina (G), citosina (C) y timina (T). En la naturaleza, sólo se forman pares de bases entre A y T y entre C y G. El tamaño del genoma se suele dar como su número total de pares de bases. Un **gene** es una región de un cromosoma cuya secuencia de DNA se puede transcribir para producir una molécula de RNA biológicamente activa. Con estos antecedentes, los objetivos de la HGI relacionados con la computación, son las siguientes:

- Establecer, mantener y mejorar bases de datos que contengan información acerca de las secuencias de DNA, la ubicación de los marcadores del DNA y de los genes, la función de los genes identificados y otra información relacionada.
- Crear mapas de los cromosomas humanos, consistentes en marcadores de DNA que permitan a los científicos localizar genes con rapidez.
- Crear repositorios de material de investigación, incluidos conjuntos ordenados de fragmentos de DNA que representen totalmente el DNA de los cromosomas humanos.
- Determinar la secuencia de DNA de una fracción considerable del genoma humano y del de otros organismos.

Requerimientos de una base de datos de genoma. Es crucial que la comunidad científica pueda tener acceso a información sobre un tema a partir de diversas bases de datos que puedan manejar diferentes aspectos del problema. Las bases de datos deben utilizar formatos estandarizados o de fácil traducción y han de estar interconectadas. La base de datos del genoma debe ser heterogénea e interoperable. Los principales problemas para alcanzar estos objetivos son:

- Las diferentes (y al parecer conflictivas) vistas de una secuencia se deben representar de manera adecuada.
- Una secuencia de nucleótidos, tal como se representa en una base de datos, es una imagen imperfecta e incompleta de una molécula idealizada: imperfecta, porque los datos experimentales siempre contienen errores; incompleta, porque siempre es posible aprender más acerca de ella, e idealizada, porque normalmente no toma en cuenta variaciones.
- Los conceptos son fluidos, y sus significados cambian constantemente. En la actualidad, ninguna tecnología satisfactoria de bases de datos maneja un entorno fluido de este tipo.
- El “proceso de construcción” del modelo del universo produce inconsistencias, ambigüedades y hasta claras contradicciones. Es importante que la base de datos del genoma maneje la evolución dinámica de esquemas, mientras que el proceso de construcción continúe sobre la base de los nuevos datos experimentales y las nuevas inferencias.

Puesto que los conceptos y el esquema están cambiando continuamente, el lenguaje de consulta deberá ser lo bastante potente como para satisfacer las necesidades de los usuarios, cada uno de los cuales tiene un grado diferente de conocimientos y experiencia. La base de datos del genoma deberá efectuar comparaciones de patrones para identificar las secuencias pertinentes, huecos, etc., lo que requerirá un recurso de comparación de patrones muy potente.

Por sí mismos, la elección y el diseño del modelo de datos que reflejará la biología del problema son aspectos muy importantes. El modelo relacional es poco adecuado para una base de datos de genoma porque no captura el orden de una secuencia. Los actuales modelos orientados a objetos son útiles porque pueden capturar objetos complejos y luego realizar verificaciones de consistencia incrementales. Sin embargo, su capacidad de consulta es débil. En la actualidad, como señalamos en la sección 24.7, el sistema LDL, con su capacidad deductiva, se está aplicando a este problema para el genoma de *E. coli*.

Trabajos actuales. Las bases de datos y los repositorios actuales que reúnen, mantienen, analizan y distribuyen datos y materiales ya tienen problemas para mantenerse al día con el crecimiento exponencial de la biología molecular. Algunas bases de datos se especializan en información de mapas y secuencias de un genoma específico; por ejemplo, hay bases de datos dedicadas exclusivamente a genomas de ratón, de bacterias *E. coli*, de *Drosophila* y de nemátodos. Otros contienen clases específicas de información de todos los genomas pertinentes. A continuación mencionamos algunas bases de datos y repositorios de los Estados Unidos. En la tabla 25.2 se resume más información acerca de estas bases de datos.

On-line Mendelian Inheritance in Man (OMIM): Herencia mendeliana humana en línea). Éste es un atlas computarizado de rasgos humanos de los que se sabe que son hereditarios; esto es, los genes expresados.

Human Gene Mapping Library (HGML): Biblioteca de mapas de genes humanos). Esta biblioteca consta de cinco bases de datos vinculadas, una para información de mapas, otra para la literatura pertinente, otra para mapas RFLP, otra para sondas DNA y una más para contactos (investigadores que poseen información sobre datos y materiales). **GenBank.** Ésta es la principal base de datos estadounidense para información de secuencias de ácidos nucleicos de seres humanos y otros organismos.

Protein Identification Resource (PIR): Recurso para identificación de proteínas). Este recurso contiene datos de secuencia para proteínas y aminoácidos, con anotaciones que indican regiones funcionales conocidas.

Protein Data Bank (PDB): Banco de datos de proteínas). Esta biblioteca reúne información sobre la estructura atómica de ácidos nucleicos, de RNA mensajero, de aminoácidos, proteínas y carbohidratos, que se ha derivado de estudios cristalográficos.

25.3 La próxima generación de bases de datos y de sistemas de gestión de bases de datos

En esta sección examinaremos a grandes rasgos los principales tipos de bases de datos, y su tecnología asociada, que están en proceso de creación. Esto incluye las siguientes bases de datos: activas (Sec. 25.3.1), de multimedia (Sec. 25.3.2), estadísticas y científicas (Sec. 25.3.3) y espaciales y temporales (Sec. 25.3.4). Después vendrá una sección sobre la gestión unificada y extensible de las bases de datos (Sec. 25.3.5), donde veremos dos sistemas extensibles —EXODUS y GENESIS— y un producto comercial llamado UnisQL, que unifica los enfoques de gestión de bases de datos orientados a objetos y relacionales.

25.3.1 Bases de datos activas[†]

Tradicionalmente, los SGDB han sido pasivos; ejecutan consultas o transacciones sólo cuando un usuario o un programa de aplicación les pide explícitamente que lo hagan. Sin embargo, muchas aplicaciones, como el control de procesos, las redes de generación/distribución de energía eléctrica, el control automatizado del flujo de trabajo de una oficina, el intercambio

[†]Esta sección se basa en gran medida en Chakravorthy 1990a.

Tabla 25.2 Algunas bases de datos de genoma existentes

Nombre	Institución	Contenido	Formato	Acceso en línea
GenBank	Los Alamos National Laboratory	DNA	MT/CD-ROM	en línea/FTP anonymous
EMBL	European Molecular Biology Laboratory	DNA	MT/CD-ROM	N/D
PIR	National Biomedical Research Foundation	proteína	MT/CD-ROM	en línea
SWISS-PROT	Universidad de Ginebra	proteína	MT/CD-ROM	FTP anonymous
PDB	Brookhaven National Laboratory	proteína	MT	FTP anonymous
GDB/OMIM	Johns-Hopkins University	mapa genérico	CD-ROM	en línea
MEDLINE	National Library of Medicine	bibliografía	CD-ROM	en línea

de programas, la gestión de batallas y la vigilancia de pacientes hospitalarios —que requieren una respuesta oportuna en situaciones críticas— no reciben un servicio adecuado de estos SGDB “pasivos”. En estas aplicaciones **restringidas por el tiempo**, es preciso vigilar la ocurrencia de condiciones definidas sobre estados de la base de datos y, en caso de ocurrir, invocar acciones específicas, quizá sujetas a ciertas restricciones de tiempo. Una posible situación en la fabricación automatizada consistiría en vigilar la ocurrencia de un suceso (salida de productos de una máquina de línea de ensamble M1 durante un intervalo de tiempo), evaluar una condición (10% de los productos están saliendo defectuosos durante ese intervalo) y emprender una o más acciones (poner en línea la máquina M2 y avisar al personal apropiado). En todo esto puede caber el acceso a bases de datos compartidas que varios usuarios estén actualizando constantemente y que deban mantenerse en un estado consistente.

Enfoques. Tradicionalmente, se han adoptado dos enfoques para satisfacer los requerimientos de las aplicaciones restringidas por el tiempo (véase la Fig. 25.2). El primero consiste en escribir un programa de aplicación especial que sondee (consulte periódicamente) la base de datos para determinar si ha ocurrido la situación que se espera. El segundo consiste en aumentar cada uno de los programas que actualizan la base de datos de modo que verifiquen si se ha presentado la situación que se vigila. El primero es difícil de implementar porque no es fácil determinar la frecuencia de sondeo óptima; el segundo pone en peligro la modularidad y la reutilización del código. Las bases de datos activas manejan la vigilancia de condiciones (con disparadores y alertadores) en un nivel de abstracción que tiene tres

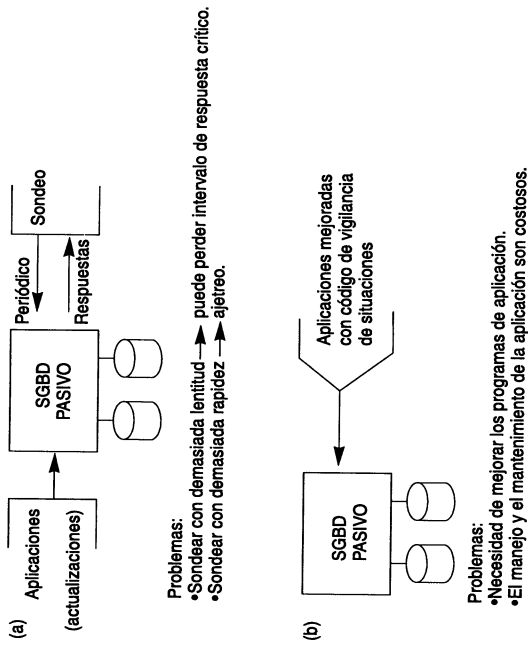


Figura 25.2 SGBD pasivo convencional que imita un comportamiento activo: (a) el enfoque de sondeo; (b) el enfoque de aplicación mejorada.

características: su semántica está bien definida; satisface los requerimientos de modelado y eficiencia para las aplicaciones de bases de datos no tradicionales, y se integra sin discontinuidad a un SGBD.

Un SGBD activo vigila continuamente el estado de la base de datos (incluso el reloj del sistema) y reacciona espontáneamente cuando ocurren sucesos predefinidos. Desde el punto de vista funcional, un sistema de gestión de bases de datos activas vigila condiciones disparadas por sucesos que representan acciones de base de datos (por ejemplo, actualizaciones) o de otra índole (por ejemplo, fallos de hardware detectados por un programa de diagnóstico); si la evaluación de la condición resulta verdadera, se ejecuta la acción. Un SGBD activo ofrece tanto modularidad como respuesta oportuna. El enfoque integrado de un SGBD activo ilustrado en la figura 25.3 puede considerarse como el siguiente paso lógico más allá de los sistemas de bases de datos deductivos. En un SGBD activo, la noción de suceso se generaliza y se hace explícita. Las reglas incluyen sucesos, condiciones y acciones. La figura 25.4 muestra una base de datos activa para reordenar automáticamente los artículos en una aplicación de control de inventarios.

Aspectos de las bases de datos activas. Seis aspectos importantes distinguen los SGBD activos de los pasivos:

- **Eficiencia:** El conjunto de todas las reglas suceso-condición-acción probablemente formará un conjunto grande de consultas predefinidas que será preciso manejar y evaluar con eficiencia cuando ocurran los sucesos especificados. La evaluación de reglas complejas bajo restricciones de tiempo es un reto cuando los conjuntos de reglas que representan un entorno altamente dinámico se vuelven muy grandes. Esto abre nuevas posibilidades de optimización si se utilizan los resultados de la optimización de múltiples consultas (Chakravarthy 1991; Sellis 1993).

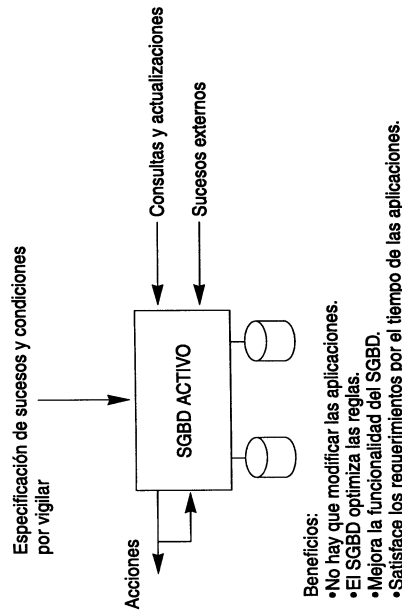


Figura 25.3 El enfoque integrado de las bases de datos activas.

- **Modos de ejecución de las reglas:** Las reglas pueden dispararse y ejecutarse en diferentes modos: inmediato, diferido o separado. En el modo de ejecución inmediato, el procesamiento de los pasos restantes de la transacción original (esto es, la transacción disparadora que provocó la ocurrencia del suceso) se suspende hasta haberse procesado por completo la regla disparada. El tiempo de respuesta y la concurrencia pueden mejorarse si la evaluación de la condición o la ejecución de la acción se separan de la transacción original (es decir, si se ejecutan en una transacción aparte). Si se permite que una regla dispare otra regla como parte de su ejecución, el modelo de transacciones se hará anidado, aumentando aún más la complejidad.

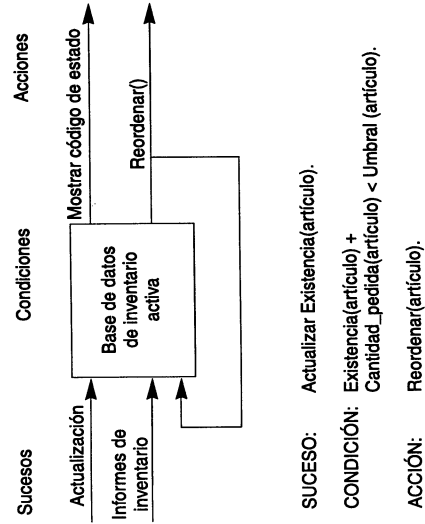


Figura 25.4 Otro ejemplo de SGBD activo (de Chakravarthy 1989).

- **Extensiones del modelo de datos:** Los SGBD activos requieren mejoras en el modelo de datos por lo menos en dos sentidos: para especificar sucesos y para especificar condiciones y acciones. Además de los sucesos correspondientes a las operaciones de la base de datos (como insertar, eliminar y modificar), es preciso manejar los sucesos temporales o periódicos, o ambas cosas (por ejemplo, que el saldo de todas las cuentas bancarias se revise a las 5 P.M. todos los días), y los sucesos generados por el usuario o la aplicación (como una señal de fallo a partir de una rutina de diagnóstico para un componente de hardware). Puede hacerse que el lenguaje maneje la especificación de sucesos complejos con base en un álgebra de sucesos (Jagadish y Gehani 1992).
- **Gestión de reglas:** En primer lugar, es esencial la capacidad de manipular reglas (añadir/eliminar/modificar) como si fueran cualquier otro objeto de datos del sistema. En segundo lugar, a menudo se requieren mecanismos para habilitar e inhabilitar reglas individuales o conjuntos de reglas. Por ejemplo, el conjunto de reglas que se activa mientras un avión avanza por la pista de un aeropuerto se debe inhabilitar una vez que despegue el avión, y es preciso habilitar un conjunto diferente de reglas para el nuevo contexto. Por último, también son importantes la habilitación y la inhabilitación selectivas de reglas.
- **Manejo de las funciones de SGBD:** El SGBD activo puede efectuar las funciones de SGBD. La gestión de restricciones (donde caben la imposición de la integridad y la seguridad), el mantenimiento de datos derivados (como las vistas) y las inferencias con base en reglas son unos cuantos ejemplos.
- **Interacción con partes del SGBD:** A diferencia de un optimizador de consultas convencional, en un SGBD activo no se pueden optimizar las reglas aisladas. La optimización de reglas requiere interactuar con varios componentes del SGBD (como el gestor de transacciones, el gestor de objetos y el planificador).

Grado de desarrollo de las bases de datos activos. A continuación hacemos un breve repaso de las investigaciones y trabajos comerciales en el área de las bases de datos activas.

HIPAC (Chakravarthy 1989, 1990), que significa High Performance ACtive database system (sistema de base de datos activo de alto rendimiento), fue un proyecto de investigación que se concentró en la gestión de datos activos restringidos por el tiempo. El modelo de conocimientos de HIPAC incluía primitivas para sucesos, condiciones y acciones, y formulaba la noción de "reglas como objetos de primera clase". En el modelo de ejecución, se investigó como interactuaban la ejecución de reglas y la ejecución convencional de transacciones; se crearon modos de acoplamiento y algoritmos para manejar reglas con varios modos de acoplamiento. Se exploraron varias cuestiones más como parte del proyecto HIPAC: una arquitectura para un SGBD activo; la combinación de procesamiento en tiempo real con bases de datos; la optimización de múltiples consultas, y el modelado y la evaluación del rendimiento de diversos modos de acoplamiento. Para la aplicación de gestión de batallas, PROBE (Dayal 1985) se extendió para incluir la detección de sucesos y el procesamiento de condiciones consistentes en consultas espaciales.

Se diseñó un mecanismo de sucesos/disparador (ETM: Event/Trigger Mechanism) en la University of Karlsruhe (Dittrich 1986; Korf 1988) para imponer restricciones de consistencia

complejas sobre bases de datos de diseño. Este trabajo atacó el problema general de imponer restricciones y se implementó encima del sistema de bases de datos de diseño DAMAS-CUS. Se mantienen tablas de sucesos, acciones y disparadores para tener acceso eficiente a los disparadores y determinarlos cuando ocurra un suceso específico. La capacidad activa es un añadido al sistema subyacente, por lo que son limitados los modos en que se puede ejecutar un disparador respecto a una transacción. En POSTGRES (Stonebraker 1987), llamado Miro en el mercado, las operaciones de base de datos y otros sucesos predefinidos (como una fecha) disparan la evaluación de reglas. Las reglas se expresan en QUEL mediante palabras reservadas (como *always* (siempre) y *demand* (demanda)), y el sistema utiliza tipos especiales de bloques para los elementos de base de datos mencionados en la condición para la implementación de reglas. La implementación de disparadores (también llamada *procesamiento a nivel de tuplas*) está incorporada en el ejecutor y utiliza tres tipos de candados. La reescritura de consultas se vale del conocido enfoque de modificación de consultas que bosquejamos en el capítulo 17. Sólo se maneja el equivalente del modo de acoplamiento inmediato.

El proyecto Starburst de IBM ha abordado el problema de las reglas orientadas a conjuntos (Widom 1990) disparadas por transiciones de estado en el contexto de las bases de datos relacionales. Los sucesos correspondientes a operaciones de bases de datos (insertar/eliminar/actualizado) se pueden asociar a una condición y a una acción con la sintaxis de SQL. La interacción entre la ejecución de transacciones y la ejecución de reglas se maneja detalladamente.

Sybase (1992) cuenta con disparadores simples, en los que las condiciones de disparo implican una sola relación, y en principio puede considerarse como un SGBD activo. Sybase reconoce sucesos que corresponden sólo a operaciones de la base de datos —insertar, eliminar y modificar—. Los disparadores no pueden devolver datos al usuario y no se pueden especificar sobre vistas ni objetos temporales. Un disparador no puede disparar otro, y siempre se ejecuta en modo inmediato.

InterBase (Interbase 90), en cambio, no impone casi ninguna de las restricciones con que cuenta Sybase. En Interbase, una especificación de disparador consiste en un suceso, un número secuencial asociado al disparador y una acción disparadora. Un suceso puede ser una de las operaciones de base de datos (almacenar, borrar o modificar). Se puede definir cualquier cantidad de disparadores sobre una relación. Además, el usuario puede especificar si la acción del disparador ha de ejecutarse antes o después de que se efectúe la operación correspondiente al suceso. Para la ejecución de múltiples reglas se emplea la información de prioridad (un entero) para resolver conflictos. El acceso a valores nuevos y viejos de las tuplas se logra por medio de las palabras reservadas *new* y *old*.

La versión 7 de ORACLE, INGRES, INFORMIX y otros proveen algún grado de manejo de reglas y disparadores. En términos de productos comerciales de SGBD, es probable que en los próximos años se ofrezcan recursos basados en reglas con una frecuencia cada vez mayor.

25.3.2 Bases de datos de multimedia

Las bases de datos de multimedia almacenan información que proviene de diferentes medios, incluidos datos numéricos, texto, imágenes de mapa de bits, imágenes de gráficos de barrido, audio y vídeo. También utilizan dispositivos de salida apropiados para presentar la información. Podemos esperar que en los próximos años haya aplicaciones a gran escala de las bases de datos de multimedia en las siguientes disciplinas:

- **Gestión de documentos y registros:** Un gran número de industrias y negocios mantienen registros muy detallados y diversos documentos. La información puede incluir datos de diseño para la ingeniería y la fabricación, expedientes médicos de pacientes, material de publicación y registros de reclamaciones de seguros, dependiendo de la organización.
- **Difusión de conocimientos:** El modo multimedia es muy efectivo para la propagación de conocimientos. Por ello, es de esperar que pronto veamos libros electrónicos y repositorios de información sobre muchos temas para el consumo del público en general.
- **Educación y capacitación:** Es posible diseñar materiales de enseñanza para diferentes públicos, desde el jardín de niños hasta los operadores de equipo profesional, a partir de fuentes de multimedia.
- **Mercadotecnia, publicidad, venta al detalle y viajes:** Prácticamente son limitadas las posibilidades que ofrece el empleo de información de multimedia en estas aplicaciones para lograr presentaciones eficaces.
- **Control y vigilancia en tiempo real:** Acoplada a la tecnología de bases de datos activas (véase la Sec. 25.3.1), la presentación de información de multimedia puede ser un método muy eficaz para vigilar y controlar operaciones de fabricación, plantas de energía nuclear, pacientes en unidades de cuidado intensivo, sistemas de transporte, etcétera.

Aspectos de las bases de datos de multimedia. Las aplicaciones de multimedia que manejan miles de imágenes, documentos, segmentos de audio y vídeo, y texto libre dependen de manera crucial de un modelado apropiado de la estructura y el contenido de los datos. Los sistemas de información multimedia son muy complejos y abarcan un gran número de aspectos, entre ellos los siguientes:

- **Modelado:** En esta área hay diferencias de opinión entre aplicar al problema técnicas de bases de datos o bien de extracción de información. No es fácil manejar objetos complejos (véase el Cap. 22) formados por tipos de datos que abarcan toda la gama de datos numéricos, textuales, de gráficos (imágenes generadas por computador), imágenes gráficas animadas, flujos de audio y secuencias de vídeo. Los documentos constituyen un área especializada y merecen consideración especial.
- **Diseño:** El diseño conceptual, lógico y físico de bases de datos de multimedia todavía no ha sido objeto de investigación. Se puede apoyar en la metodología general descrita en el capítulo 14, pero los problemas en cada nivel son mucho más complejos.
- **Almacenamiento:** Almacenar datos de multimedia en dispositivos estándar como los discos presenta problemas de representación, compresión, correspondencia con jerarquías del dispositivo, archivado y almacenamiento intermedio durante la operación de E/S. Para resolver este problema es probable que los proveedores de productos de multimedia se adhieran a normas como JPEG o MPEG. En los SCBD, un recurso de "BLOB" (*Binary Large Object*: objeto binario extenso) permite almacenar y leer mapas de bits sin tipo. Se requerirá software estandarizado para manejar la sincronización, la compresión/descompresión, etc., que todavía está en la etapa de investigación (Vijn 1992).

- **Obtención de datos:** El modo peculiar como se obtiene la información almacenada en las bases de datos depende de lenguajes de consulta y estructuras de índice mantenidas internamente. En cambio, el enfoque de "extracción de información" depende estrictamente de palabras clave o términos de índice predefinidos. En el caso de imágenes, datos de vídeo y datos de audio, esto implica muchas cuestiones, algunas de las cuales trataremos más adelante.
- **Rendimiento:** En las aplicaciones de multimedia que manejan sólo documentos y texto, las restricciones de rendimiento las determina subjetivamente la población de usuarios. En cambio, en las aplicaciones que implican la reproducción de vídeo o la sincronización de audio y vídeo predominan las limitaciones físicas. Por ejemplo, el vídeo debe presentarse a una velocidad uniforme de 60 cuadros por segundo. Las técnicas para la optimización de consultas pueden calcular el tiempo de respuesta esperado antes de evaluar la consulta. El empleo de procesamiento de datos en paralelo puede aliviar algunos problemas, pero en la actualidad casi todos estos intentos se encuentran en su fase experimental.

Las cuestiones anteriores han originado diversos problemas de investigación por resolver. En esta sección sólo trataremos unos cuantos de ellos como problemas representativos.

Bases de datos vs. perspectivas de extracción de información. En los modelos y sistemas de bases de datos, el modelado del contenido de los datos no ha sido problemático porque éstos tienen una estructura rígida y el significado de un ejemplar de un dato se puede inferir del esquema. En cambio, la extracción de información (*IR: information retrieval*) tiene que ver sobre todo con modelar el contenido de documentos de texto (mediante palabras clave, índices de frases, redes semánticas, etc.) para los cuales generalmente no se tiene en cuenta la estructura. Al modelar el contenido, el sistema examina los descriptores de contenido del documento para determinar si éste es pertinente para la consulta de un usuario. Consideremos, por ejemplo, un informe de reclamación de accidente de una compañía aseguradora como objeto de multimedia: contiene imágenes del accidente, formas de seguros estructuradas, grabaciones de audio de las partes implicadas en el accidente, el informe escrito por el representante de la aseguradora y otra información: qué modelos de datos deben usarse para representar información de multimedia es todavía una cuestión por definir.

Requerimientos de modelado y obtención de datos de multimedia/hipermedia. A fin de aprovechar todo el poder expresivo de los multimedia, el sistema debe contar con un elemento de modelado que permita al usuario especificar enlaces entre dos nodos cualesquiera. Los **enlaces de hipermedia** pueden adoptar varias formas distintas, entre ellas las siguientes:

- Se puede especificar enlaces con o sin información asociada. Un enlace puede asociarse a descripciones largas.
- Los enlaces pueden partir de un punto específico dentro de un nodo, o partir de todo el nodo.
- Los enlaces pueden ser direccionales o no direccionales.

La capacidad de enlace del modelo de datos debe dar cuenta de todas estas variaciones. Cuando se requiere una extracción de datos de multimedia con base en el contenido, el mecanismo de consulta deberá tener acceso a los enlaces y a la información asociada a ellos.

El sistema deberá contar con recursos para definir vistas sobre los enlaces, así como enlaces públicos y privados. Es posible obtener valiosa información contextual a partir de la información estructural. Un enlace de hipermmedia generado automáticamente no revela nada nuevo acerca de los dos nodos, porque el enlace se creó automáticamente por algún método. Por otro lado, los enlaces de hipermmedia generados manualmente y la información de enlace puede servir para saber más sobre los nodos que se están conectando. Los recursos para crear y utilizar tales enlaces, los lenguajes de consulta por recorrido (navegación) para aprovecharlos, etc., son características importantes de cualquier sistema que permita la utilización eficaz de la información de multimedia.

Indización de imágenes. Hay dos estrategias para enfrentar el problema de la indización de imágenes. Una utiliza técnicas de procesamiento de imágenes para identificar los diferentes objetos de manera automática. El otro se vale de técnicas de indización manual para asignar términos y frases del índice. Un problema importante con las técnicas de procesamiento de imágenes para indizar representaciones gráficas tiene que ver con la escalabilidad. Los sistemas más modernos permiten sólo patrones sencillos en las imágenes; la complejidad se incrementa conforme es mayor el número de rasgos reconocibles. Otro problema importante se relaciona con la complejidad de la consulta. Las reglas y los mecanismos de inferencia, como se vio en el capítulo 24, pueden servir para derivar hechos de más alto nivel a partir de características simples de las imágenes. Con ello se logran consultas de alto nivel, como "buscar los diseños de apartamentos que permitan entrar un máximo de luz de sol en la sala" en una aplicación para arquitectos.

La estrategia de la extracción de información para indizar imágenes se basa en uno de estos tres esquemas de indización:

- Los *sistemas clasificatorios* (Gordon 1988) ordenan jerárquicamente las imágenes en ciertas categorías predefinidas. En este enfoque, el indizador y el usuario deben conocer bien las categorías disponibles. No es posible capturar los detalles más finos de una imagen compleja ni las interrelaciones de los objetos de una imagen.
- Los *sistemas basados en teclado* usan un vocabulario no controlado (o controlado) como en la indización de documentos textuales. Es posible capturar hechos simples representados en la imagen (como "región cubierta de hielo") y hechos derivados de la interpretación de alto nivel realizada por seres humanos (como hielo permanente, nevada reciente y hielo polar).
- En el *esquema de entidad-atributo-relación* (Leung 1992) se identifican todos los objetos de la imagen y los vínculos entre los objetos y los atributos de éstos.

En el caso de los documentos de texto, un indizador humano puede elegir las palabras clave de la reserva de palabras disponibles en el documento que ha de indizarse. Esto no es posible en el caso de datos de imágenes o de vídeo.

Problemas no resueltos en la extracción de texto. Aunque por largo tiempo la extracción de texto se ha utilizado en las aplicaciones de negocios y en los sistemas bibliotecarios, los diseñadores de sistemas de extracción de texto se siguen enfrentando a estos problemas:

- *Indización de frases:* Es posible lograr mejoras sustanciales si se asignan descripciones de frases (en contraposición a los términos de índice de una sola palabra) a los

documentos y se les utiliza en las consultas, siempre y cuando dichos descriptores sean buenos indicadores del contenido del documento y de las necesidades de información (Fagan 1987).

- *Empleo de diccionarios de sinónimos:* Una razón del deficiente rendimiento que muestran los sistemas actuales es que el vocabulario del usuario difiere del vocabulario empleado para indizar los documentos. Una solución sería usar un diccionario de sinónimos (tesauro) para ampliar la consulta del usuario con términos relacionados. El problema entonces consistirá en localizar un tesauro adecuado para el dominio de interés.

- *Resolución de ambigüedad:* Una de las razones de la baja precisión (la razón entre el número de elementos pertinentes extraídos y el número total de elementos extraídos) en los sistemas de extracción de información textual es que las palabras tienen múltiples significados. Una manera de resolver la ambigüedad semántica sería emplear un diccionario en línea (Krovetz 1992); la otra sería comparar los contextos en que ocurren las dos palabras (Salton 1991).

Con los promisorios sistemas de información de multimedia es de esperar que converjan las disciplinas de la extracción de información y de la gestión de bases de datos, que hasta ahora han tendido a divergir.

25.3.3 Bases de datos científicas y estadísticas

No es nuevo el empleo de computadores para manejar información científica y estadística. Sin embargo, la tecnología de bases de datos tardó bastante en llegar a los estadísticos y científicos. En marzo de 1990 la National Science Foundation patrocinó un taller, cuyo anfitrión fue la University of Virginia, en el que se reunieron representantes de las ciencias de la Tierra, de la vida y del espacio con científicos de la computación para analizar los problemas que enfrenta la comunidad científica en el área de la gestión de bases de datos. Es probable que estos problemas crezcan exponencialmente durante la próxima década. En esta sección resumiremos los principales problemas expuestos en este informe (French *et al.* 1990).

Los datos científicos son únicos debido a su naturaleza relativamente estática y a su retención indefinida. El entorno típico de procesamiento de transacciones en las bases de datos convencionales está ausente en las bases de datos científicas; tienen una baja frecuencia de actualización, y rara vez se desechan los datos viejos. Ya antes nos referimos al gran volumen de datos científicos en la iniciativa del genoma humano: implica 3000 millones de bases de nucleótidos. La sonda planetaria "Magallanes" generará alrededor de un billón de bytes de datos en un lapso de cinco años.

Los sistemas de bases de datos científicas comprenden tres "dimensiones": nivel de interpretación, análisis propuesto y fuentes de información. En el análisis que sigue, con *conjunto de datos* nos referiremos a datos relacionados con un solo experimento o misión; *base de datos* se aplicará a cualquier agregado de datos que se resume o compile a partir de múltiples experimentos.

La dimensión de la interpretación. Esta dimensión se refiere al continuo que refleja la naturaleza bruta (no procesada) de la información. Conciérnele a lo que se espera del conjunto de

datos y a la forma como se emplea éste. Con base en esta dimensión, los datos se pueden dividir en las siguientes categorías:

- **Datos brutos/de sensor:** Éstos son valores brutos (que casi nunca se conservan) obtenidos directamente del dispositivo de medición.
- **Datos calibrados:** Éstos consisten en valores físicos brutos (que normalmente se conservan), corregidos por operadores de calibración.
- **Datos validados:** Éstos consisten en datos calibrados que se han filtrado a través de procedimientos de control de calidad; es el tipo de datos más utilizados para fines científicos.
- **Datos derivados:** Éstos son datos de agregación frecuente, como los datos reticulados o promediados, de los cuales se han perdido los detalles de las mediciones subyacentes.
- **Datos interpretados:** Éstos son datos derivados que se relacionan con otros conjuntos de datos o con la literatura del campo.

Por lo regular se debe retener y propagar la información sobre el procesamiento junto con un conjunto de datos. Es posible que los datos brutos y su resumen tengan que estar interrelacionados.

Análisis científico propuesto. En su mayoría, los datos científicos almacenados se someten a algún tipo de análisis. Los datos de ciencias de la Tierra se someten al análisis estadístico de series de tiempo. Los datos de genoma se analizan para buscar patrones en el almacenamiento de caracteres. Los datos espaciales en matrices grandes se transforman (por ejemplo, con la transformada de Fourier) para un análisis espectral. El modelo de datos relacional no es adecuado por su incapacidad para representar secuencias y porque no puede representar las características de los datos empleados durante el análisis. Algunas veces, es preciso “corregir” los datos en vez de “actualizarlos”, pero los SGBD relacionales no cuentan con recursos para efectuar esto.

Fuentes de los datos. Ésta es una característica discriminatoria fundamental en las bases de datos científicas. En un entorno de base de datos con una sola fuente, los datos brutos se acopian y procesan con técnicas de filtrado o de calibración. Los datos validados resultantes se utilizan a la luz de la misión científica. La complejidad sintáctica y semántica de los datos interpretados es mucho más alta que la de los datos originales. Las bases de datos de múltiples fuentes agregan una nueva dimensión de complejidad, con múltiples metodologías de recolección de datos, instrumentos de registro de datos y diferentes protocolos, todo lo cual tiene efectos en la codificación y la precisión. Es posible que estas bases de datos las usen y controlen de manera independiente diversos sistemas de computador y SGBD. Sin embargo, la meta casi siempre es crear un archivo de datos común, como GENBANK para la iniciativa del genoma humano (véase la Sec. 25.2.3), que utilizarán los equipos de investigadores originales que están generando y procesando los datos brutos. Esto hereda los problemas de las bases de datos distribuidas y las multibases de datos que vimos en el capítulo 23.

Los SGBD tradicionales han resultado inadecuados para satisfacer los requerimientos antes mencionados, sobre todo por las siguientes razones:

- Carecen de funciones para manejar las redundancias e inexactitudes inherentes en los datos durante la experimentación científica y su registro. El avance de los datos a lo largo de la dimensión de interpretación no puede manejarse de manera elegante.
- No permiten un análisis de los datos con funciones estadísticas multivariadas ni agregaciones.
- Los SGBD implican el gasto extra de procesamiento de transacciones, lo cual es innecesario en la mayor parte de las aplicaciones científicas.
- La distinción inherente en los datos estadísticos entre un valor y lo que representa (como un valor numérico que represente el número de hogares en el municipio de Ancona que tienen ingresos en el intervalo de \$75 000-\$100 000 anuales y por lo menos un hijo en una escuela privada) no se puede manejar en los SGBD tradicionales. Las abstracciones como el producto cruzado propuesto en el modelo de datos OSAM* (Su *et al.* 1988) resultan útiles.

Es preciso resolver problemas importantes en las áreas de la definición y la manipulación de datos y de las normas tecnológicas para poder satisfacer los requerimientos de las bases de datos científicas y estadísticas. En el área de la definición de datos surgen las siguientes cuestiones:

- Es preciso explorar nuevos modelos de datos, como los que se trataron en el capítulo 21. Estos modelos deberán manejar tipos de datos especiales como series de tiempo y secuencias de DNA.
- La gestión de metadatos es crucial, y se requiere un mecanismo para manejar citas, diferentes tipos de documentación y detalles experimentales.
- La evolución de los datos debe estar apoyada por un registro de intervenciones apropiado.
- Se requieren normas entre las disciplinas científicas para asentar resultados experimentales e incluso para compartir citas.

En el área de manipulación de datos son pertinentes los siguientes puntos:

- No basta con realizar consultas convencionales y generar informes. Las funciones estadísticas y analíticas deben ser parte integral del sistema. También se observa una tendencia hacia la adición de una interfaz de lenguaje de consulta como SQL a los paquetes estadísticos existentes como SAS.
- Es preciso contar con diversas operaciones de resumen y agregación, junto con un recurso para describir la estadística resumida.
- Debe haber funciones de exportación e importación para transferir datos hacia afuera y hacia adentro de la base de datos científica, y a utilerías estadísticas.

- Como inicialmente los estadísticos o científicos son usuarios que no conocen bien las bases de datos, habría que incluir una gama de interfaces con el usuario, además de los recursos de inspección y de recorrido. Habría que tomar medidas para que los usuarios puedan “aprender” cómo gestionar los datos del sistema mediante una graduación de dichas interfaces y recursos.

En el área de las normas tecnológicas tendríamos que destacar los siguientes aspectos:

- La codificación física de los datos es importante, así como las técnicas de comprensión e indexación necesarias, para un almacenamiento y extracción eficientes. El cifrado de datos resulta esencial para que la información sea confidencial cuando intervienen sujetos humanos.
- En experimentos a largo plazo (como una sonda espacial o la vigilancia permanente de pacientes con una cierta enfermedad), es posible que los esquemas tengan que ser modificados conforme evolucionan las técnicas experimentales.
- Se requieren normas, como son los formatos de datos y las herramientas de análisis asociadas, interdisciplinarias e intradisciplinarias. Un ejemplo es la norma FITS que utiliza la comunidad de astrofísica.
- La transmisión de conjuntos de datos completos a través de redes de alta velocidad se está haciendo factible. Para ello se requieren catálogos extensos con nombres, seudónimos, códigos, procedimientos de derivación de datos, medidas de control de calidad de los datos e información de uso. Es necesario archivar correctamente los conjuntos de datos.

El análisis anterior no deja duda de que la comunidad de usuarios de las bases de datos científicas tiene un conjunto de necesidades muy especial. French *et al.* (1990) contiene una serie de recomendaciones para resolver problemas sociológicos y técnicos; se hace un llamado a las sociedades profesionales para fomentar la cooperación entre los científicos a fin de facilitar el intercambio de información.

25.3.4 Gestión de bases de datos espaciales y temporales

En esta subsección cubriremos dos importantes tipos de datos que, apenas en los últimos años, comienzan a atraer la atención de los investigadores del modelado de bases de datos. En la sección 25.2.1, al hablar de las aplicaciones CAD/CAM, hicimos alusión al modelado geométrico. Ya sea en el contexto de CAD/CAM, del diseño arquitectónico y de ingeniería civil, o de la cartografía y los estudios geológicos, es importante modelar en la dimensión espacial. En este sentido los modelos de datos actuales son deficientes. La semántica espacial se puede capturar con tres representaciones comunes:

- **Representación sólida:** El espacio se divide en fragmentos de diversos tamaños. Las características espaciales de una entidad se representan entonces con el conjunto de estos fragmentos que está asociado a la entidad.
- **Representación de frontera (o modelos de armazón de alambre):** Las características espaciales se representan con segmentos de líneas o fronteras.
- **Representación abstracta:** Se utilizan vínculos con semántica espacial, como ARRIBA-DE, CERCA-DE, ESTÁ-JUNTO-A, Y DETRÁS-DE para asociar entidades.

El proyecto PROBE de Computer Corporation of America (Dayal *et al.* 1987) incorporó el manejo de datos espaciales al modelo de datos funcional DAPLEX (Shipman 1981). Aunque es evidente que muchas aplicaciones podrían sacar provecho de estos recursos, los diferentes dominios de aplicación tienen distintos requerimientos. Por ejemplo, en las

aplicaciones VLSI, el espacio es bidimensional y discreto; los objetos básicos por almacenar son puntos, segmentos de líneas, rectángulos y polígonos. En el modelado de cuerpos para la mayor parte de las aplicaciones de fabricación, el espacio es tridimensional y continuo, y los objetos básicos son curvas y superficies definidas por parámetros. Las operaciones pertinentes para cada aplicación también son distintas.

Si suponemos que un espacio de dos o tres dimensiones se representa con segmentos de líneas y polígonos, podrían hacerse las siguientes consultas representativas:

- Consultas sobre segmentos de líneas:
 - Todos los segmentos que pasan por un punto o conjunto de puntos dado.
 - Todos los segmentos que tienen un conjunto de extremos dado.
 - Todos los segmentos que intersecan un segmento de línea dado.
- Consultas de proximidad:
 - El segmento de línea más cercano a un punto dado.
 - Todos los segmentos dentro de una distancia dada respecto a un punto dado (también denominada consulta de intervalo o de ventana).
- Consultas respecto a los atributos de segmentos de líneas:
 - Dado un punto, el polígono circunscrito mínimo cuyos segmentos de líneas constituyentes sean todos de un cierto tipo.
 - Dado un punto, todos los polígonos que inciden en él.

Se han propuesto diferentes formas para almacenar objetos en el espacio descomponiendo el espacio. Como ejemplos podríamos citar los siguientes:

- **Rectángulos de acotación mínimos:** Los objetos se agrupan en jerarquías, que a su vez se organizan en estructuras similares a árboles B. Ejemplos de ello son el árbol R (Gutman 1984) y el árbol R*.
- **Celdas disjuntas:** Los objetos se descomponen en subobjetos. Cada subobjeto está en una celda distinta.
- **Retícula uniforme:** El espacio se divide superponiéndole una estructura de retícula uniforme. Los objetos aparecen en una celda específica de esta estructura.
- **Árboles cuadráticos:** Ésta es una estructura de datos jerárquica para resolución de variables; puede haber muchas variaciones.

La representación de objetos en el espacio y el procesamiento de consultas sobre esos objetos son áreas de investigación específicas que los creadores de sistemas de información geográfica (GIS: *Geographic Information Systems*) estudian con ahínco. Si el lector desea un repaso exhaustivo de las estructuras de datos espaciales y sus aplicaciones, deberá consultar Samet (1990a, 1990b).

Gestión de datos temporales. La información temporal es un caso unidimensional de la información espacial. Los aspectos temporales incorporados en las bases de datos deben incluir tres tipos de apoyo para el tiempo: puntos de tiempo, intervalos de tiempo y vínculos abstractos en los que participa el tiempo (antes de, después de, durante, simultáneamente,

concurrentemente, etc.). El aspecto **histórico** de las bases de datos es muy importante en aplicaciones como la gestión de proyectos, las historias clínicas de los pacientes en un hospital, las historias de mantenimiento de equipos y el control administrativo y de operación en los sistemas de oficina.

Una limitación de las bases de datos actuales es que la información se hace efectiva en el momento mismo en que se asienta en la base de datos. No se contempla una distinción entre el *tiempo de registro* (o *tiempo de transacción*) en el que se introducen ciertos datos y el periodo de *tiempo lógico* (o *tiempo válido*), durante el cual los valores específicos de los datos son válidos. De hecho, en muchos casos hay que registrar un tiempo de transacción para un suceso. Muchas actualizaciones de base de datos en las aplicaciones reales son *retroactivas* (se hacen efectivas en algún momento previo) o bien *proactivas* (se hacen efectivas en algún momento futuro). Otra limitación es que los sistemas actuales no mantienen ninguna historia de las modificaciones. Cada actualización destruye hechos antiguos. Así, la base de datos representa sólo el estado actual de algún dominio y no la historia de ese dominio. Para satisfacer los requerimientos anteriores, se requieren modelos de datos que incorporen explícitamente el tiempo, separando la información que varía con el tiempo de la que no lo hace, y representándolas por separado. Un modelo que se ha propuesto es el modelo temporal relacional (TRM: *Temporal Relational Model*) de Navathe y Ahmed (1989). En él, los atributos se dividen en variantes con el tiempo y no variantes con el tiempo, y lo mismo se hace con las relaciones. En las relaciones variantes con el tiempo se anexan dos atributos de marca de tiempo que representan el tiempo de inicio y el tiempo final del intervalo durante el cual la tupla es válida. Este enfoque se denomina *marcado de tiempo de tuplas*. La figura 25.5 muestra una relación EMPLEADO variante con el tiempo en TRM.

Se ha propuesto un procedimiento de normalización del tiempo para este modelo a fin de evitar anomalías temporales; el lenguaje SQL se extiende al SQL temporal (TSQL) con la adición de diversas características, como la cláusula WHEN (cuando), el ordenamiento temporal, segmentos de tiempo, funciones agregadas, agrupación y una ventana de tiempo en movimiento. Otra escuela de pensamiento propone el *marcado de tiempo de atributos*, que anexa una marca de tiempo a cada nuevo valor de atributo en el momento de la actualización (Ben-Zvi 1982; Clifford y Tansel 1985; Gadia 1988). Esto produce relaciones no normalizadas con un álgebra y un lenguaje de consulta mucho más complicados; por añadidura, no es posible representar vínculos con claves que no varían con el tiempo.

NúmEmp	Salario	Puesto	T _i	T _f
33	20K	Mecanógrafo	12	24
33	25K	Secretario	25	35
45	27K	Ingeniero Jr.	28	37
45	30K	Ingeniero Sr.	38	42

Notas: T_i: Tiempo inicial
T_f: Tiempo final

Figura 25.5 Relación variante con el tiempo en el modelo relacional temporal.

Un hito reciente en las investigaciones sobre bases de datos temporales fue la publicación de una colección muy completa de trabajos que representan lo último en este campo, editada por Tansel *et al.* (1993). La organización de esta colección destaca los temas en los que han aparecido trabajos interesantes sobre gestión de bases de datos temporales en fechas recientes:

- **Extensiones del modelo relacional:** Estos trabajos han hecho hincapié en las diferentes maneras de agregar información de marcas de tiempo a un modelo relacional estático que contenga relaciones “instantáneas”. Es posible añadir diferentes tipos de tiempos empleando dos enfoques primordiales: marca de tiempo triple, que ya se ilustró, y marca de tiempo de atributos, que origina relaciones desnormalizadas (no 1FN). Se han propuesto extensiones de SQL y QUEL para manejar operaciones temporales. Se han propuesto álgebras, cálculos y funciones agregadas relacionales temporales. Se ha emprendido un esfuerzo organizado para “estandarizar” las mejoras temporales hechas a SQL.
- **Otros modelos de datos:** Estos trabajos incluyen extensiones del modelo ER y del modelo de datos funcional. También se han propuesto un modelo histórico de hoja de cálculo y un modelo de base de datos deductiva temporal.
- **Cuestiones de implementación:** El procesamiento y la optimización de consultas adquieren nuevas formas, dependiendo del modelo de datos y del lenguaje empleado. Se ha investigado la indexación de información temporal de diversas maneras (Elmasri *et al.* 1990) y su actualización, así como el control de concurrencia y la recuperación de bases de datos de *tiempo de transacción* (o *reversión*). El operador de reunión temporal puede asumir muchas formas, y se ha estudiado su optimización. Por último, se ha investigado el cómputo incremental y decremental de bases de datos de tiempo de transacción.

Entre los problemas no resueltos están el razonamiento con información temporal, el procesamiento de transacciones con bases de datos de tiempo válido y tiempo de transacción, así como la integración de información temporal entre entornos heterogéneos.

Con la aparición de los discos ópticos de escritura única que pueden almacenar 10¹³ bytes de datos, la anexión de actualizaciones a los datos históricos se ha hecho viable. Sin embargo, es preciso trabajar más sobre los aspectos de implementación y de factibilidad comercial de estos modelos. Además, la implementación de estos modelos exige a los sistemas operativos poseer capacidades de gestión de tiempo, grabación y sincronización de cuadros de tiempo multiusuario.

25.3.5 Gestión de bases de datos extensible y unificada

Como vimos en una sección anterior de este capítulo, es necesario construir nuevos SCBD para enfrentar los retos de las nuevas aplicaciones. Los SCBD son paquetes de software complejos con algoritmos e interrelaciones entre componentes bastante complicados. En vista del tiempo tan largo que se requiere para crear un SCBD, se ha propuesto un nuevo enfoque modular que construye un SCBD a partir de “componentes de SCBD”, o bloques de construcción.

Un **sistema de gestión de bases de datos extensible** como éste ensamblaría módulos (algoritmos) preescritos para crear nuevos “SCBD hechos a la medida”. Este enfoque tiene varias ventajas obvias:

- La creación de nuevos SGBD es rápida y económica.
- Las mejoras tecnológicas o algorítmicas se pueden incorporar rápidamente en los módulos reutilizables, con lo que el sistema siempre estará al día.
- Las nuevas técnicas y algoritmos propuestos pueden evaluarse sin hacer modificaciones importantes al sistema.

Este enfoque es uno de los objetivos del proyecto GENESIS en la University of Texas (Batory *et al.* 1986) y de EXODUS en la University of Wisconsin (Graefe y Dewitt 1987; Carey *et al.* 1986, 1986a, 1988). GENESIS utiliza la estrategia de definir los componentes de un SGBD y las interfaces entre ellos de tal manera que sea posible configurar un nuevo SGBD en unos cuantos minutos a través de una interfaz de usuario controlada por menús, con sólo seleccionar las opciones apropiadas. Se designan módulos de conexión compatible para los métodos de acceso, la optimización de consultas, el control de concurrencia, la recuperación y otras funciones preimplementando un conjunto de algoritmos alternativos. Las interfaces comunes permiten ensamblarlos como bloques de construcción.

EXODUS tiene una arquitectura distinta. Ofrece ciertos recursos básicos, incluidos un gestor de almacenamiento versátil y un gestor de tipos. Este último permite definir jerarquías de clases con herencia múltiple. El objeto de almacenamiento es una secuencia de bytes sin tipo, no interpretada, de longitud variable y de tamaño arbitrario. Los mecanismos de gestión de almacenamiento intermedio, control de concurrencia y recuperación se basan en el gestor de almacenamiento y se pueden modificar. Es posible seleccionar estructuras de índice independientes de los tipos, como son árboles B*, archivos de retícula y dispersión lineal, de una biblioteca de métodos de acceso. Se provee el lenguaje E para el implementador de bases de datos (DBI: *database implementor*). Este lenguaje es una extensión de C que incorpora la noción de objetos persistentes, y hace innecesario que el implementador se ocupe de la estructura interna de estos objetos. La capa de "métodos de operador" de la arquitectura contiene una mezcla de código proporcionado por el DBI y por EXODUS para operar sobre los objetos almacenados con tipo. El procesamiento de consultas se divide en dos fases: optimización y evaluación. El DBI provee la descripción de los operadores (en un lenguaje de consulta destino), métodos para implementar los operadores y fórmulas de costo para dichos métodos. El generador de optimizadores basado en reglas transforma estas descripciones en código fuente en C, creándose así un optimizador hecho a la medida para el sistema destino. Varios proveedores han utilizado el gestor de almacenamiento de EXODUS (incluido O2, que vimos en el capítulo 22) para implementar sus propios SGBD.

El enfoque de configuración anterior, que implica generar un SGBD hecho a la medida, contrasta con otro enfoque propuesto para crear nuevas aplicaciones: el de construir un SGBD con funcionalidad extensible proporcionando un conjunto amplio de funciones. Ejemplos de este enfoque de **funcionalidad completa** son los proyectos PROBE (Dayal *et al.* 1987) y STARBURST (Haas *et al.* 1988).

Ya nos hemos referido a PROBE y STARBURST, cuando estudiamos las bases de datos activas, y mencionamos el procesamiento de consultas espaciales en PROBE. El SGBD POSTGRES, ahora llamado Miro (Stonebraker *et al.* 1992), también combina capacidades de bases de datos orientadas a objetos y activas con el modelo relacional.

Una tercera tendencia en esta categoría es hacia la gestión unificada de bases de datos. Un ejemplo de sistema de esta categoría es UniSQL, que, según se asegura, combina "el grado de avance, la potencia y la facilidad de uso de las herramientas más utilizadas para

la creación de aplicaciones" al tiempo que "cumple la promesa de integrar el desarrollo orientado a objetos y las bases de datos de multimedia". El producto UniSQL está organizado como sigue:

- **UniSQL/x:** Este componente provee la plataforma de SGBD cliente-servidor. Permite a las aplicaciones relacionales tradicionales coexistir con la nueva aplicación de base de datos orientada a objetos. SQL/s es un lenguaje de consulta orientado a objetos que se ajusta al ANSI SQL.
- **UniSQL/M:** Este componente es un gestor de bases de datos heterogéneas que permite acceso a bases de datos de otros SGBD relacionales y prerrelacionales. Maneja un lenguaje de consulta orientado a objetos, SQL/M, con el que es posible definir y procesar un esquema de multibases de datos.
- **Herramientas UniSQL/4GE:** ObjectMaster es una herramienta para generar dinámicamente aplicaciones para la creación y gestión de objetos dentro de los entornos SQL/x y SQL/M. Otras herramientas, llamadas Visual Editor y Media Master, permiten visualizar y editar los esquemas, así como generar informes complejos.

Es probable que la próxima generación de SGBD siga el patrón del SGBD UniSQL. Estos sistemas combinarán características del modelo de datos relacional con las de otros modelos y ofrecerán una combinación de los recursos activos, de multimedia, científicos, espaciales y temporales que vimos aquí.

25.4 Interfaces con otras tecnologías e investigaciones futuras

La tecnología de bases de datos no es un área que se desarrolle aisladamente. Está íntimamente ligada a varias otras disciplinas, algunas de las cuales ya se señalaron aquí. Por ejemplo, el área de las bases de datos distribuidas y múltiples (Cap. 23) está muy relacionada con las telecomunicaciones y las redes. Las bases de datos de multimedia están íntimamente vinculadas con la extracción de información. En esta sección comentaremos algunas de estas interfaces y áreas que probablemente se desarrollarán en el futuro.

25.4.1 Interfaz con la tecnología de ingeniería de software

El objetivo primordial de la ingeniería de software es hallar formas de hacer más fácil, eficaz y eficiente el proceso de creación de software. Un producto final importante de la ingeniería de software es el software de aplicación. Ahora que los SGBD se han vuelto tan ubicuos, la creación de software de aplicación se debe considerar en el contexto de los SGBD. Es probable que haya una fusión de los conceptos de ambas disciplinas en las siguientes áreas:

- **Bases de datos de diseño:** Una actividad de diseño de gran envergadura, como la creación de un sistema de software grande, es similar a los proyectos de diseño de gran magnitud en otras áreas, como la construcción de un rascacielos, de una estación de energía eléctrica o de una fábrica. El diseño de un sistema grande se puede descomponer jerárquicamente en diseños de componentes cada vez más pequeños. Cada

componente evolucionaria, las especificaciones de diseño cambian y aparecen nuevas alternativas de diseño. Una forma apropiada de consolidar y controlar la información de diseño es crear una base de datos para ella.

- *Generación de aplicaciones a partir de especificaciones de alto nivel:* Esta área tiene gran interés para la creación de aplicaciones comerciales. Ya hay en el mercado varios generadores de aplicaciones comerciales, como parte de herramientas CASE (*computer-assisted software engineering*: ingeniería de software asistida por computador). El incentivo principal es que estos generadores reducen el esfuerzo y el costo de la creación de aplicaciones. Puesto que los SCBD comparten este objetivo, es probable que en el futuro veamos generadores de aplicaciones íntimamente acoplados a los SCBD subyacentes.
- *Herramientas de software:* Ésta es un área importante para el desarrollo futuro. Las herramientas de software tienen dos fines: aliviar el pesado trabajo que ciertas tareas implican para los seres humanos y mejorar el desempeño de algunas máquinas. Se requieren diversas herramientas con los SCBD. Las del primer tipo incluyen herramientas para la especificación y análisis de requerimientos, para el diseño conceptual, para la integración de vistas, para la transformación de esquemas entre modelos, para el diseño físico y la optimización, y para la división y el reparto de los datos en las bases de datos distribuidas. Entre las herramientas del segundo tipo están las de vigilancia del rendimiento, las de afinación, las de reorganización y las de reestructuración. En los últimos años se ha observado una actividad tremenda en el desarrollo de herramientas. Sin embargo, pocas herramientas actuales ofrecen buenas interfaces que se puedan comunicar sin dificultad con otras herramientas. Después de mucho hay excepciones. Es probable que la intensa actividad en el desarrollo de herramientas continúe en los próximos años.

- *Prototipos:* Éste es un tema muy popular entre los entusiastas de la ingeniería de software, pero ha recibido muy poca atención en el área de bases de datos. La creación de prototipos de bases de datos y aplicaciones puede ser de gran ayuda para validar el diseño de los esquemas, refinar su estructura y evaluar la frecuencia relativa de las consultas (transacciones). Es probable que veamos un desarrollo sustancial en esta área. La creciente disponibilidad de computadores personales y estaciones de trabajo hace aún más importante el papel de los prototipos.

25.4.2 Interfaz con la tecnología de inteligencia artificial

En el capítulo 24 analizamos la relación entre la lógica y las bases de datos y vimos cómo propició el desarrollo de la tecnología de las bases de datos deductivas. Hoy día está creciendo el interés en pasar de las bases de datos a las **bases de conocimientos**. Los conocimientos son información en un nivel más alto de abstracción. Por ejemplo, "el señor González tiene 45 años de edad" puede considerarse como un hecho en una base de datos. "El señor González es de edad madura" no es un hecho tan preciso; es una forma más amplia y elevada de conocimiento. De manera similar, "el señor González tuvo un accidente el 17 de enero de 1988 en Nueva York, en la rampa que conduce de la autopista I-495 a la I-278" es un hecho; "el señor González es un conductor imprudente" es conocimiento. "Todos los hombres de edad madura

son imprudentes" es otro elemento de conocimiento de más alto nivel, aunque puede representar una inferencia errónea porque en general no es cierto.

Es muy difícil definir y cuantificar los conocimientos. Estos los generan por lo regular expertos en algún dominio del saber. Así, los conocimientos sirven para definir, controlar e interpretar los datos de una base de datos.

Los conocimientos adoptan diversas formas (Wiederhold 1984):

- *Conocimientos estructurales:* Éstos son conocimientos acerca de las dependencias y relaciones entre los datos (por ejemplo, "la inscripción en el plan de prestaciones x está sujeta a la inscripción previa en el plan de prestaciones y").
- *Conocimientos generales por procedimientos:* Éstos son conocimientos que se pueden describir sólo con un procedimiento o método (por ejemplo, un procedimiento para determinar si un cliente es "sujeto de crédito").
- *Conocimientos específicos para una aplicación:* Éstos son conocimientos determinados por las reglas y pautas aplicables en un dominio específico (por ejemplo, el cálculo de la tarifa aérea más económica entre dos ciudades).
- *Conocimientos para dirección de empresas:* Ésta es una forma más alta de conocimientos que ayuda a una empresa en la toma de decisiones. Por ejemplo, en el ámbito de toda una compañía, estos conocimientos incluyen el costo de relocalizar y capacitar empleados, así como alguna medida del beneficio en términos del ánimo y la lealtad de los empleados que les hará permanecer en el trabajo más de n años.

Los **conocimientos intensionales** se definen como conocimientos que subyacen y preceden al contenido de hechos de la base de datos. Estos conocimientos pueden especificarse antes de establecerse la base de datos. Los sistemas de bases de datos sirven para manejar los datos y los metadatos, que juntos constituyen **conocimientos extensionales** o conocimientos incorporados en hechos y ejemplares. Los sistemas de conocimientos no sólo utilizan los conocimientos extensionales sino también los intensionales, tal vez en forma de las reglas de una base de reglas. Los **conocimientos derivados** son una combinación de conocimientos extensionales e intensionales. Una base de datos puede almacenar relaciones llamadas padre (Nombre_padre, Nombre_hijo), madre (Nombre_madre, Nombre_hijo) y persona (Nombre_persona, Sexo). A partir de estas tres relaciones básicas sería posible definir diversos vínculos familiares, desde primos hasta tíos y abuelos, y ésta sería la función de una base de reglas.

Casi todos los sistemas de base de conocimientos actuales almacenan los conocimientos en forma de reglas. Otras representaciones incluyen aserciones lógicas, redes semánticas y marcos. Llamamos **conocimientos estratégicos** a la estrategia de inferencias y control de una base de conocimientos.

Un área de investigación nueva y prometedora es la *extracción de conocimientos*, que implica obtener información previamente desconocida —pero útil— de una base de datos (Piatesky, Shapiro y Frawley 1991; Anwar et al. 1992). Entre las áreas de desarrollo potencial están el razonamiento basado en ejemplos (Kolodner 1993), aplicado como ayuda para el razonamiento con bases de datos; el diseño automatizado de esquemas por medio de agrupamiento conceptual (Beck et al. 1993), y el aprendizaje basado en explicaciones, que se aplica para mantener la consistencia de una base de datos (como en el sistema XCON; Schimmler et al. 1991). Podemos usar muchas estrategias de aprendizaje como nuevos paradigmas para el procesamiento de consultas en los sistemas de bases de datos.

25.4.3 Interfaces con el usuario para bases de datos

A pesar de los avances recientes en las interfaces con el usuario, hay unas cuantas áreas que siguen inexploradas, sobre todo las que podrían beneficiar al usuario de bases de datos:

- **Lenguajes personalizados:** Los usuarios esporádicos y ocasionales necesitan lenguajes que sean fáciles de usar y se ajusten bien a sus dominios de problemas. Un ejecutivo de cuentas en una casa de inversiones maneja un conjunto de términos y requerimientos de procesamiento diferentes de los de un supervisor de producción en una compañía manufacturera. Si ambos usan el mismo sistema cada uno deberá tener una interfaz personalizada, lo que en la actualidad no es imposible, aunque implica que un esfuerzo de programación considerable. Los sistemas futuros deberán facilitar la generación de un conjunto personalizado de construcciones de lenguaje para cada conjunto de usuarios.
- **Paradigmas alternativos para tener acceso a las bases de datos:** El sistema llamado Spatial Data Management System (SDMS: sistema de gestión de datos espaciales) de Computer Corporation of America (Herot 1980) es un buen ejemplo de cómo alejarse del enfoque orientado a registros y archivos de la gestión de datos. SDMS simula un entorno en el que los datos se organizan en un espacio tridimensional como podría ser una oficina. Basa la búsqueda de datos en el principio de que la gente es hábil para localizar información si sabe dónde está colocada y si conoce algunos atributos del aspecto físico del medio en el que está registrada. Un sistema así permite una "exploración espacial" en la base de datos, fomenta la inspección aleatoria y utiliza iconos como información de diccionario (metadatos). Este sistema puede almacenar y exhibir ilustraciones e imágenes de videodiscos y otros medios. La visualización de datos científicos e incluso de repositorios de documentos mediante técnicas ingeniosas está recibiendo mucha atención (Korfang 1991; Chalmers 1992). En el futuro quizá presenciemos más avances en esta dirección que incorporarán también otras formas de información, como películas, animaciones y (dando un paso más allá) ¡incluso olores y sabores!

- **Interfaz de lenguaje natural en múltiples idiomas:** Al aumentar la velocidad de los medios de comunicación, ya se está haciendo necesario poner la misma base de datos bajo el mismo SCBD a disposición de estadounidenses, italianos, japoneses y otros al mismo tiempo. Las interfaces de lenguaje natural actuales son ineficientes e inadecuadas. Las interfaces futuras exigirán el manejo de varios idiomas. Hay problemas relacionados, como exhibir información en diferentes clases de escritura, permitir la edición de texto en diversos idiomas (por ejemplo, español y árabe, ya que este último se lee de derecha a izquierda), almacenarseudónimos en diferentes lenguajes, etcétera.

25.4.4 Máquinas de bases de datos y arquitecturas

Las investigaciones sobre máquinas de bases de datos se han concentrado primordialmente en la obtención y el almacenamiento eficientes de grandes cantidades de datos. Una dirección de investigación futura consistiría en diseñar un computador que integre los componentes de resolución de problemas con las funciones de gestión de datos. Otra sería producir

hardware que maneje funciones de gestión de bases de datos, como el control de la seguridad, la integridad y la concurrencia, y la gestión de transacciones, que actualmente se implementan con software y constituyen un renglón importante de gasto extra para el rendimiento. También es necesario investigar arquitecturas paralelas para los sistemas basados en conocimientos.

25.4.5 Interfaz con la tecnología de lenguajes de programación

Se están efectuando investigaciones sobre los lenguajes de consulta —el diseño de dichos lenguajes, así como la modificación, optimización y compilación de consultas— en relación con las áreas de aplicación difíciles de CAD, de sistemas de oficina y de gestión de bases de datos estadísticas, científicas, espaciales y temporales. Se requiere trabajar más sobre la demostración de la equivalencia de modelos y lenguajes y la formalización de su semántica.

Está surgiendo un nuevo campo de investigación en el que los llamados lenguajes de programación de bases de datos (DBPL: *database programming languages*) (Bancilhon y Bunneman 1990) habrán de reducir al mínimo la "diferencia de impedancia" entre los lenguajes de programación tradicionales y los lenguajes de consulta de bases de datos. Sin embargo, ninguna de las propuestas hechas hasta ahora se ha demostrado en una escala práctica.

Muchos otros campos pueden beneficiarse con la aplicación de la tecnología de bases de datos. No podemos ni siquiera intentar nombrarlos todos aquí. Un área prometedora es el trabajo cooperativo apoyado por computador (CSCW: *computer-supported cooperative work*) (Kling 1991). Podría beneficiarse facilitando la resolución de problemas en un contexto de grupo y empleando los principios de la gestión de bases de datos activas (Chakravarthy et al. 1993).

Concluimos este capítulo con la figura 25.6 que nos muestra que, en el modelado de información y las aplicaciones de procesamiento futuro en las diversas áreas en las que se utilizarán bases de datos, esta tecnología debe colaborar con las tecnologías relacionadas de ingeniería de software, inteligencia artificial, lenguajes de programación, interfaces con el usuario y sistemas distribuidos. Los usuarios emprenderán actividades de resolución de problemas con la base de datos como foco y con estas otras tecnologías apoyando diversas facetas del proceso de resolución de problemas.

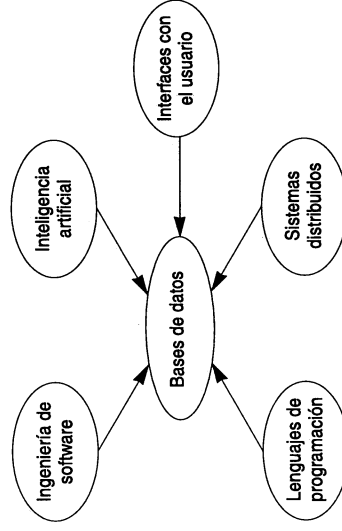


Figura 25.6 Integración futura de las tecnologías.

Bibliografía selecta

Ya mencionamos en las diversas secciones gran parte de la bibliografía pertinente para este capítulo. Aquí presentaremos unas cuantas referencias adicionales. Se puede consultar varias publicaciones periódicas y actas de conferencias para estar al tanto de los avances en este campo. Entre las más prominentes están *ACM Transactions on Database Systems*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Software Engineering* (sobre todo antes de 1990), *Information Systems y Data and Knowledge Engineering* (North Holland). En los últimos años han empezado a aparecer las siguientes publicaciones periódicas nuevas: *The VLDB Journal*, *Journal of Distributed and Parallel Databases*, *Journal of Intelligent Information Systems* y *Journal of Intelligent Cooperative Information Systems*. Varias conferencias internacionales con arbitraje riguroso publican actas de los artículos. Entre ellas destacan la conferencia anual ACM SIGMOD (desde 1974), la conferencia VLDB (desde 1975), la IEEE Data Engineering Conference (desde 1984), la Conference on Extending Database Technology (desde 1989) y la Entity-Relationship Conference (desde 1980). Debido a la rapidez con que crece el interés por las nuevas tecnologías, se inició un nuevo taller llamado Research Issues in Data Engineering en 1991. Se ha dedicado a la interoperabilidad y a las multibases de datos en 1991 y 1993, a los multimedia en 1992 y a las bases de datos activas en 1994. El IEEE publica las actas. También se ha iniciado una nueva conferencia llamada Parallel and Distributed Information Systems, a partir de 1991.

Se ha celebrado una serie de talleres internacionales llamados Statistical Database Workshops a intervalos irregulares desde 1982. La International Federation of Information Processing Societies tiene un grupo de interés especial sobre bases de datos que ha estado celebrando talleres sobre semántica de bases de datos. La serie de actas de conferencias sobre sistemas expertos de bases de datos (EDS 1984, 1986, 1988) contiene artículos relacionados con bases de datos, lógica, sistemas expertos y basados en conocimientos, y técnicas relacionadas con la integración de las tecnologías de bases de datos e inteligencia artificial.

Un número especial de *IEEE Transactions on Knowledge and Data Engineering* está dedicado a la descripción de prototipos de sistemas de base de datos, algunos de los cuales se mencionaron en este capítulo y en los anteriores.

Los informes de ciertos talleres resumen los resultados de mesas redondas organizadas por la National Science Foundation. Uno de estos informes (Silberschatz *et al.* 1991) trata del futuro de las investigaciones sobre sistemas de bases de datos. En la sección 25.3.3 se presentaron algunos puntos importantes de ese informe sobre las bases de datos científicas (French *et al.* 1990).

Entre los libros de lectura especializados, Stonebraker (1988) es una colección de artículos de investigación sobre sistemas de bases de datos. Mylopoulos y Brodie (1989) incluye contribuciones sobre la interfaz de la inteligencia artificial y las bases de datos. Han comenzado a aparecer libros especializados sobre áreas de aplicación como las bases de datos geográficas (Laurini 1992).

Otro tema de investigación actual, que no hemos mencionado antes, implica la incorporación de información incompleta e imprecisa (difusa) en las bases de datos. Entre los artículos que estudian la información incompleta están Lipski (1979), Vassiliou (1980), Imielinski y Lipski (1981) y Liu y Sunderraman (1988). En Zadeh (1983) y Zvieli (1986) se analizan técnicas para manejar información difusa.

Uno de los primeros artículos de reseña sobre máquinas de bases de datos dorsales es Maryanski (1980). Las técnicas para implementar bases de datos de memoria principal se analizan en DeWitt *et al.* (1984). Las aplicaciones de las bases de datos de multimedia se estudian en Christodoulakis y Faloutsos (1986).

Dittrich (1986) es una reseña del enfoque orientado a objetos. Maier *et al.* (1986) describe el sistema orientado a objetos GEMSTONE/OPAL. El Darmstadt Database Kernel System se describe en Paul *et al.* (1987). Navathe y Pillalamari (1988) analiza la forma de convertir el modelo ER (que vimos en los capítulos 3 y 21) en un modelo orientado a objetos.

Muchos proyectos tratan las bases de datos para diseño de ingeniería. Haskin y Lorie (1982) y Lorie y Plouffe (1983) analizan los objetos de diseño complejos en las bases de datos. Eastman (1987) examina los recursos que requieren las bases de datos de ingeniería. Kemper *et al.* (1987) describe un SCBD orientado a objetos para aplicaciones de ingeniería. El modelado geométrico y el procesamiento de datos espaciales se tratan en Kemper y Wallrath (1987) y en Orenstein (1986). La estructura de árbol R para indizar datos espaciales se presenta en Guttman (1984). Banerjee *et al.* (1987a) analizan la evolución de esquemas, y Du y Chanta (1987) hace lo propio con las bases de datos CAD para VLSI.

A últimas fechas ha crecido el interés por las bases de datos en tiempo real, como se aprecia en los trabajos de Abbott y García-Molina (1988). En Gadia (1988) se describe un modelo de datos temporales y un lenguaje de consulta para bases de datos relacionales. Tsichritzis (1982) trata la gestión de formas.

A P É N D I C E A

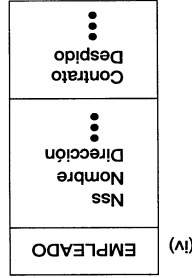
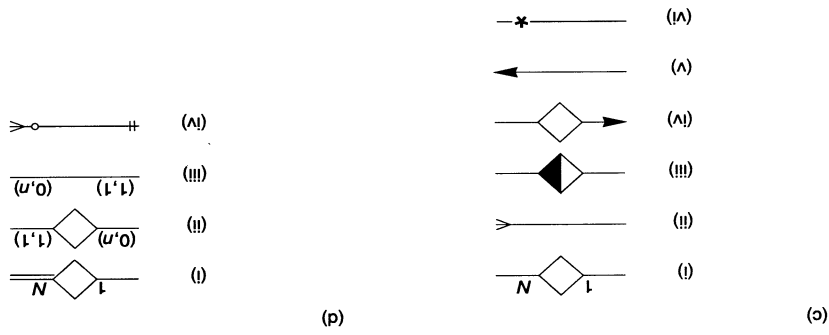
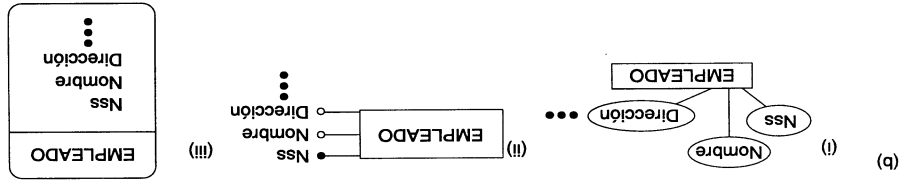
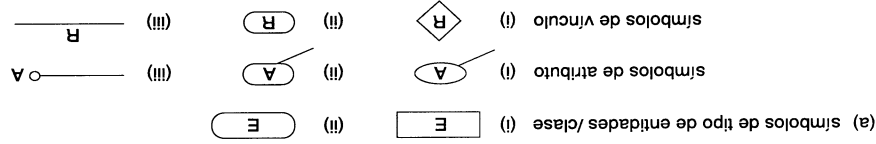
Notaciones diagramáticas alternativas

La figura A.1 muestra varias notaciones diagramáticas distintas para representar conceptos de los modelos ER y EER. Desafortunadamente, no existe una notación estándar: los diferentes diseñadores de bases de datos prefieren distintas notaciones. De manera similar, las diversas metodologías CASE (ingeniería de software asistida por computador) y OOA (análisis orientado a objetos) utilizan diversas notaciones. Algunas notaciones están asociadas a modelos que tienen conceptos y restricciones adicionales a los de los modelos ER y EER descritos en los capítulos 3 y 15, mientras que otros modelos cuentan con menos conceptos y restricciones. La notación que usamos en el capítulo 3 es muy cercana a la notación original de los diagramas ER, que todavía se usa ampliamente. Aquí veremos algunas notaciones alternativas.

La figura A.1(a) muestra diferentes notaciones para representar tipos de entidades/clases, atributos y vínculos. en los capítulos 3 y 15 usamos los símbolos marcados con (i) en dicha figura; a saber, rectángulo, óvalo y rombo. Observe que el símbolo (ii) para tipos de entidades/clases, el símbolo (ii) para atributos y el símbolo (ii) para vínculos son similares, pero los utilizan diferentes metodologías para representar tres conceptos distintos. El símbolo de línea recta (iii) para representar vínculos lo usan varias herramientas y metodologías.

La figura A.1(b) muestra algunas notaciones para conectar atributos a los tipos de entidades. Aquí utilizamos la notación (i). La notación (ii) emplea la tercera notación (iii) para atributos de la figura A.1(a). Las últimas dos notaciones de la figura A.1(b) —(iii) y (iv)— son populares en las metodologías OOA y en algunas herramientas CASE. En particular, la última notación representa tanto los atributos como los métodos de una clase, separados por una línea horizontal.

La figura A.1(c) muestra diversas notaciones para representar la razón de cardinalidad de los vínculos binarios. Usamos la notación (i) en los capítulos 3 y 21. La notación (ii) —conocida como notación de *patas de gallina*— es muy popular. La notación (iv) utiliza la flecha como referencia funcional (del lado N al lado 1) y se parece a nuestra notación



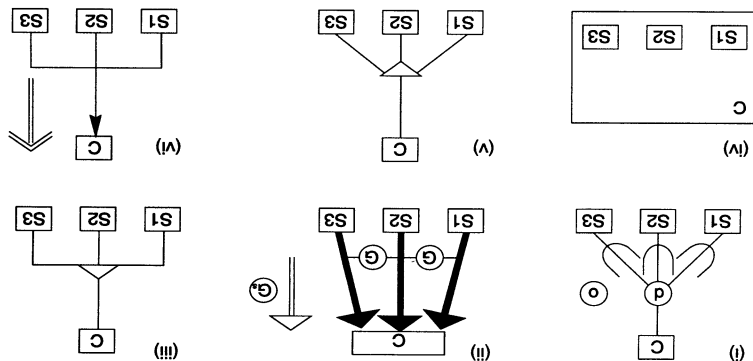
para las claves externas en el modelo relacional (véase la Fig. 6.7); la notación (v) —empleada en los *diagramas de Bachman*— usa la flecha en la *dirección contraria* (del lado 1 al lado N). En el caso de un vínculo 1:1, (ii) utiliza una línea recta sin patas de gallina; (iii) pone en blanco ambas mitades del rombo, y (iv) coloca puntas de flecha en ambos extremos. En el caso de un vínculo M:N, (ii) usa patas de gallina en ambos extremos de la línea; (iii) pone en negro ambas mitades del rombo, y (iv) no coloca puntas de flecha.

La figura A.1(d) muestra algunas variaciones para representar restricciones (*mín*, *máx*), que sirven para indicar tanto la razón de cardinalidad como la participación total/parcial. (ii) es la notación alternativa que usamos en la figura 3.14 y que analizamos en la sección 3.4. Recuerde que nuestra notación especifica la restricción de que cada entidad debe participar en por lo menos *mín* y cuando más *máx* casos de un vínculo. Así, en el caso de un vínculo 1:1, ambos valores de *máx* son 1, y para un M:N ambos valores de *máx* son *n*. Un valor de *mín* mayor que cero especifica participación total (dependencia de existencia). En las metodologías que usan la línea recta para representar vínculos, es común *invertir la colocación* de las restricciones (*mín*, *máx*), como se muestra en (iii). Otra técnica muy utilizada —con la misma colocación que (iii)— es representar *mín* como 0 (la letra “o” o un círculo, que significa cero) o como 1 (raya vertical, que significa 1), y representar *máx* como | (raya vertical, que significa 1) o como pata de gallina (que significa *n*), como puede verse en (iv).

La figura A.1(e) muestra algunas notaciones para representar especialización/generalización. Usamos la notación (i) en el capítulo 15, donde una *d* en el círculo especifica que las subclases (S1, S2 y S3) son disjuntas y una *o* especifica que se traslapan. La notación (ii) usa *G* (de generalización) para especificar clases disjuntas y *Os* para especificar traslapeo; algunas notaciones usan la flecha rellena de negro, mientras que otras usan la flecha hueca (que se muestra a un lado). La notación (iii) usa un triángulo que apunta hacia la superclase, y la notación (v) emplea un triángulo que apunta hacia las subclases; también es posible utilizar ambas notaciones en la misma metodología: (iii) para indicar generalización y (v) para indicar especialización. La notación (iv) coloca los cuadros que representan subclases dentro del cuadro que representa la superclase. De las notaciones basadas en (vi), algunas emplean una flecha de una sola línea y otras una de doble línea (que se muestra a un lado).

Las notaciones de la figura A.1 muestran sólo algunos de los símbolos diagramáticos que se han utilizado o sugerido para representar esquemas conceptuales de bases de datos. También se han empleado otras notaciones, así como varias combinaciones de las anteriores. Conveniría establecer un estándar al que todo el mundo se ajustara, a fin de evitar malentendidos y reducir la confusión.

Figura A.1 Notaciones alternativas. (a) Símbolos para tipo de entidades/clase. (b) Representación de atributos. (c) Representación de razones de cardinalidad. (d) Diversas notaciones (*mín*, *máx*). (e) Notaciones para representar especialización/generalización.



A P É N D I C E B

Parámetros de discos

El parámetro de disco más importante es el tiempo requerido para localizar un bloque arbitrario en el disco, dada su dirección de bloque, y luego transferirlo del disco a un almacenamiento intermedio (*buffer*) de la memoria principal. Éste es el **tiempo de acceso aleatorio** para tener acceso a un bloque de disco. Debemos considerar tres componentes de tiempo:

1. **Tiempo de búsqueda (s):** Éste es el tiempo requerido para colocar mecánicamente la cabeza de lectura/escritura sobre la pista correcta en los discos de cabeza móvil. (En los de cabeza fija es el tiempo requerido para conmutar electrónicamente a la cabeza de lectura/escritura apropiada.) En los discos de cabeza móvil este tiempo varía, dependiendo de la distancia entre la pista que está actualmente bajo la cabeza de lectura/escritura y la pista especificada en la dirección de bloque. Por lo regular, el fabricante del disco cita un tiempo de búsqueda medio en milisegundos. El intervalo usual de tiempos de búsqueda medios es de 10 a 60 mseg. Éste es el principal "culpable" del retardo que implica transferir bloques entre el disco y la memoria.

2. **Retardo rotacional (rd : *rotational delay*):** Una vez que la cabeza de lectura/escritura está sobre la pista correcta, el usuario debe esperar hasta que el principio del bloque deseado pase por debajo de la cabeza. En promedio, esto corresponde a la mitad del tiempo que tarda media revolución del disco, pero en realidad va desde el acceso inmediato (si el principio del bloque deseado ya está colocado bajo la cabeza de lectura/escritura justo al terminar la búsqueda) hasta una revolución completa del disco (si el principio del bloque requerido acababa de pasar debajo de la cabeza al terminar la búsqueda). Si la velocidad de rotación del disco es de p revoluciones por minuto (rpm), el retardo rotacional medio rd está dado por

$$rd = (1/2) * (1/p) \text{ min} = (60 * 1000) / (2 * p) \text{ mseg}$$

Un valor representativo de p es 3600 rpm, lo que da un retardo rotacional de $rd = 8.33$ mseg. En el caso de los discos de cabeza fija, en los que el tiempo de búsqueda es despreciable, este componente provoca el mayor retardo al transferir un bloque de disco.

3. **Tiempo de transferencia de bloque (btr : *block transfer time*):** Una vez que la cabeza de lectura/escritura está al principio del bloque deseado, se requiere cierto tiempo para transferir los datos del bloque. Este tiempo de transferencia de bloque depende del tamaño del bloque, el tamaño de las pistas y la velocidad rotacional. Si la **velocidad de transferencia (tr : *transfer rate*)** del disco es tr bytes/mseg y el tamaño del bloque es B bytes, entonces

$$btr = B/tr \text{ mseg}$$

Si tenemos pistas de 50 Kbytes y p es 3600 rpm, la velocidad de transferencia en bytes/mseg es

$$tr = (50 * 1000) / (60 * 1000 / 3600) = 3000 \text{ bytes/mseg}$$

En este caso, $btr = B/3000$ mseg, donde B es el tamaño del bloque en bytes.

El tiempo medio requerido para encontrar y transferir un bloque, dada su dirección, se estima con

$$(s + rd + btr) \text{ mseg}$$

Esto se cumple tanto para la lectura como para la escritura de un bloque. El principal método para reducir este tiempo consiste en transferir varios bloques que estén almacenados en una o más pistas del mismo cilindro; así, la búsqueda sólo se requiere para el primer bloque. Si queremos transferir consecutivamente k bloques *no contiguos* que están en el mismo cilindro, necesitaremos aproximadamente

$$s + (k * (rd + btr)) \text{ mseg}$$

En este caso, necesitaremos dos o más *buffers* en la memoria principal, porque estaremos leyendo o escribiendo continuamente los k bloques, como explicamos en la sección 4.3. El tiempo de transferencia por bloque se reduce aún más cuando se transfieren *bloques consecutivos* de la misma pista o cilindro. Esto elimina el retardo rotacional para todos los bloques, excepto el primero, de modo que la estimación para transferir k bloques consecutivos es

$$s + rd + (k * btr) \text{ mseg}$$

Una estimación más precisa para la transferencia de bloques consecutivos tiene en cuenta la separación entre bloques (véase la Sec. 4.2.1), que incluye la información que permite a la cabeza de lectura/escritura determinar cuál bloque está a punto de leer. Por lo regular, el fabricante del disco cita una **velocidad de transferencia masiva (btr : *bulk transfer rate*)** que considera el tamaño de dicha separación al leer bloques almacenados consecutivamente. Si el tamaño de la separación es G bytes, entonces

$$btr = (B/(B + G)) * tr \text{ bytes/mseg}$$

La velocidad de transferencia masiva es la velocidad con que se transfieren *bytes útiles* de los bloques de datos. La cabeza de lectura/escritura del disco debe pasar por todos los bytes de una pista conforme el disco gira, incluidos los bytes de las separaciones entre bloques, que contienen información de control pero no datos reales. Cuando se usa la velocidad de transferencia masiva, el tiempo requerido para transferir los datos útiles de un bloque, de entre varios bloques consecutivos, es B/btr . Así, el tiempo estimado para leer k bloques almacenados consecutivamente en el mismo cilindro se convierte en

$$s + rd + (k * (B/btr)) \text{ mseg}$$

Otro parámetro de disco es el **tiempo de reescritura**. Éste nos sirve en casos en los que leemos un bloque del disco y lo colocamos en un almacenamiento intermedio de la memoria principal, actualizamos el almacenamiento intermedio y luego lo volvemos a escribir en el mismo bloque de disco en el que estaba almacenado antes. En muchos casos, el tiempo requerido para almacenar el almacenamiento intermedio en la memoria es menor que el requerido para que el disco efectúe una revolución completa. Si sabemos que el almacenamiento intermedio está listo para reescritura, el sistema puede mantener las cabezas del disco en la misma pista y reescribir el *buffer* actualizado en el bloque del disco durante la siguiente revolución. Debido a esto, el tiempo de reescritura T_{rw} casi siempre se toma como el tiempo requerido para una revolución del disco:

$$T_{rw} = 2 * rd \text{ mseg}$$

En síntesis, los parámetros que hemos analizado y los símbolos que usamos para denotarlos son:

tiempo de búsqueda: s mseg
 retardo rotacional: rd mseg
 tiempo de transferencia de bloque: bt mseg
 tiempo de reescritura: T_{rw} mseg
 velocidad de transferencia: tr bytes/mseg
 velocidad de transferencia masiva: btr bytes/mseg
 tamaño de bloque: B bytes
 tamaño de separación entre bloques: G bytes

A P É N D I C E C

Comparación de modelos de datos y sistemas

En este apéndice compararemos de forma somera los diversos modelos de datos descritos en este libro y los sistemas de bases de datos que se fundan en dichos modelos. Compararemos las capacidades de representación de los modelos, sus lenguajes, sus estructuras de almacenamiento representativas y sus restricciones de integridad.

C.1 Comparación de las representaciones en los modelos de datos

Una diferencia importante entre las representaciones en los modelos de datos se refiere a la forma de representar los vínculos. En el *modelo relacional*, las conexiones entre dos relaciones se representan mediante atributos de clave externa de una relación que hacen referencia a la clave primaria de otra relación. Las tuplas individuales que tienen el mismo valor en los atributos de clave externa y clave primaria tienen un vínculo lógico, aunque no están conectadas físicamente. En el *modelo de red*, las conexiones 1:N entre dos tipos de registros se representan explícitamente con la construcción de tipo de conjuntos, y el SGBD conecta físicamente los registros vinculados en un ejemplar de conjunto.

Podemos decir que el modelo relacional emplea referencias *lógicas* o *simbólicas*, en tanto que el modelo de red usa *referencias físicas*. Podemos mantener las referencias lógicas en una base de datos de red —además de las referencias físicas— duplicando el campo clave del registro propietario en los registros miembro y usando los valores duplicados para especificar restricciones estructurales o para seleccionar automáticamente los conjuntos. Si esta duplicación se efectúa con todos los tipos de conjuntos, los tipos de registros del esquema de red se asemejarán a las relaciones correspondientes del esquema relacional. Por otro lado, si no duplicamos la clave propietaria en el registro miembro, el sistema no podrá verificar la validez de la pertenencia al conjunto; será obligación del usuario vincular físicamente los registros miembro a los ejemplares de conjunto, empleando las órdenes

CONNECT, DISCONNECT, RECONNECT o STORE. Esta última orden efectúa la vinculación cuando se especifica pertenencia a conjuntos AUTOMATIC.

El *modelo jerárquico* también representa los vínculos explícitamente, pero tiene limitaciones graves en comparación con el modelo de red. En tanto que en este último modelo un tipo de registros puede ser miembro de *cualquier cantidad* de tipos de conjuntos, en el modelo jerárquico sólo puede tener un padre real y un padre virtual. Esto crea problemas al modelar tipos de vínculos M:N y n-arios. Si un esquema contiene principalmente tipos de vínculos 1:N en la misma dirección, se le puede modelar naturalmente como una jerarquía; sin embargo, si hay muchos tipos de vínculos es difícil lograr una buena representación jerárquica sin duplicar algunos registros ni emplear apuntadores. Por esto, el modelo jerárquico suele considerarse inferior a los modelos relacional y de red en cuanto a su capacidad de modelado.

En los modelos *orientados a objetos* (OO), los vínculos suelen representarse mediante referencias a través del identificador de objeto (OID). Esto se parece un poco a las claves externas, excepto que se usan identificadores internos del sistema en vez de atributos definidos por el usuario. Los modelos OO manejan estructuras de objetos complejos empleando constructores de tupla, de lista, de conjunto, etc. Por añadidura, manejan la especificación de métodos y los mecanismos de herencia que permiten crear nuevas definiciones de clases a partir de las que ya existen. El *modelo relacional anidado* maneja la creación de relaciones estructuradas jerárquicamente además de las relaciones planas. En cierto sentido, las relaciones anidadas se asemejan al empleo de los constructores de tuplas y de conjuntos de manera anidada, por lo que podemos considerar las capacidades de estructuración de las relaciones anidadas como un subconjunto de los constructores tipo OO.

En la tabla C.1 se compara la terminología empleada en los modelos ER, relacional, de red, jerárquico y OO. En la tabla C.2 se resume la transformación de los elementos ER a los de estos modelos.

C.2 Comparación de lenguajes de manipulación de datos

Hemos distinguido entre dos tipos de lenguajes de base de datos: lenguajes de alto nivel que operan sobre conjuntos de registros y lenguajes de bajo nivel que operan sobre un registro a la vez. Casi todos los lenguajes de base de datos de alto nivel están asociados al modelo relacional, en tanto que los modelos de red y jerárquico están asociados a lenguajes de bajo nivel de registro por registro. Por lo regular, los sistemas OO apoyan la programación objeto por objeto en un lenguaje de programación OO, además de contar con un lenguaje de consulta de alto nivel para seleccionar un subconjunto de objetos de una colección.

El *modelo relacional* cuenta con varios lenguajes de alto nivel. Las operaciones formales del álgebra relacional se aplican a conjuntos de tuplas; una consulta se especifica mediante una *secuencia* de operaciones sobre relaciones. En el cálculo relacional, una sola expresión (y no una secuencia de operaciones) especifica una consulta; especificamos *qué* queremos obtener sin indicar *cómo* y en qué orden debe obtenerse. Por esto, se considera que el cálculo relacional está en un nivel declarativo más alto que el álgebra relacional. Los lenguajes comerciales para el modelo relacional —como SQL, QUEL y QBE— se basan primordialmente en el cálculo relacional. También incluyen recursos para manejar funciones agregadas, agrupar, ordenar, mantener tuplas repetidas y efectuar aritmética, que no pertenecen al ámbito del álgebra o el cálculo relacionales básicos. Casi todos los sistemas relacionales permiten a los usuarios introducir consultas de manera directa e interactiva o incorporar las consultas en un lenguaje de programación.

Tabla C.1 Terminología comparativa de los modelos de datos

Modelo de entidad-vínculo	Modelo relacional Formal	Modelo relacional Informal	Modelo de red	Modelo jerárquico	Modelo OO
Esquema de tipo de entidades	Esquema de relación	Descripción de tabla	Descripción de tipo de registros	Descripción de tipo de registros	Descripción de clase
Conjunto de entidades	Estado de relación	Tabla	Ejemplares de registros	Tipo de registros de tipo (segmento en IMS)	Colección de objetos
Ejemplar de entidad	Tupla	Fila	Ocurrencia de registro	Ocurrencia de registro	Objeto
Tipo de vínculo 1:N	a	a	Tipo de conjuntos	Tipo de vínculo padre-hijo (VPH)	c
Ejemplar de vínculo 1:N	a	a	Ocurrencia de conjunto (o ejemplar de VPH)	Ocurrencia de VPH	c
Atributo	Atributo	Columna	Campo o elemento de información	Campo o elemento de información	Atributo
Conjunto de valores	Domnio	Tipo de datos	Tipo de datos	Tipo de datos	Tipo de datos atómico
Clave	Clave candidata	Clave candidata	b	b	b
Atributo multivaluado	b	b	Vector o grupo repetitivo	b	Constructor de conjunto
Atributo compuesto	b	b	Grupo repetitivo	b	Constructor de tupla

^aNo hay un concepto correspondiente; el vínculo se establece empleando claves externas.
^bNo hay un concepto o término equivalente.
^cNo hay un concepto correspondiente; el vínculo se establece empleando referencias.

Tabla C.2 Resumen de la transformación de los conceptos del modelo ER a los del relacional, de red, jerárquico y OO.

Concepto del modelo ER	Modelo relacional	Modelo de red	Modelo jerárquico	Modelo OO
Tipo de entidades	Como relación	Como tipo de registros	Como tipo de registros	Como clase
Tipo de entidades débiles	Como relación, pero con la clave primaria de la relación identificadora	Como tipo de registros miembro de un tipo de conjuntos que tiene al tipo de registros identificador como propietario (o como grupo repetitivo)	Como tipo de registros hijo del tipo de registros identificador	Como clase o como conjunto de tuplas dentro de una clase propietaria
Tipo de vínculos 1:1	Incluye la clave primaria de una relación como clave externa de la otra, o las combina en una sola relación	Usa un tipo de conjuntos cuyos ejemplares estén restringidos a tener un registro miembro, o los combina en un solo tipo de registros	Usa un tipo de VPH cuyos ejemplares estén restringidos a tener un solo registro hijo, o los combina en un solo tipo de registros	Dos atributos de referencia que sean inversos; ambos son simples
Tipo de vínculos 1:N	Incluye la clave primaria de la relación del "lado 1" como clave externa en la relación del "lado N"	Emplea un tipo de conjuntos	Emplea un tipo de vínculos padre-hijo	Dos atributos de referencia que sean inversos; uno tiene valor de conjunto
Tipo de vínculos M:N	Establece una nueva relación que incluya como claves externas a las claves primarias de las relaciones participantes	Establece un tipo de registros de enlace y lo hace miembro de los tipos de conjuntos propiedad de los tipos de registros participantes	(a) Usa una sola jerarquía y registros repetidos (b) Usa múltiples jerarquías y tipos de VPHV	Ambos atributos de referencia tienen valor de conjunto; o como en el renglón que sigue
Tipo de vínculos n-ario	Igual que el M:N	Igual que el M:N	(a) Igual que el M:N (b) Hace al vínculo padre de una sola jerarquía y a los tipos de entidades participantes sus hijos	Crea una nueva clase con n referencias

Las órdenes de los DML de red y jerárquico que vimos son de bajo nivel porque buscan y obtienen registros individuales. Debemos usar un lenguaje de programación de aplicación general e incorporar las órdenes de bases de datos en el programa. En ambos lenguajes, el concepto de registro actual es crucial para interpretar el significado de las órdenes de DML, pues el efecto de la orden depende del registro actual. El DML del modelo de red utiliza indicadores de actualidad adicionales como el actual del tipo de conjuntos y el actual del tipo de registros, que también afectan el resultado de las órdenes de DML. Aunque estos conceptos de actualidad facilitan el acceso a los registros uno por uno, el programador debe conocer perfectamente la forma como las diferentes órdenes afectan los indicadores de actualidad para poder escribir programas correctos. Estas órdenes de registro por registro tienen su origen en las órdenes del procesamiento de archivos tradicional.

De lo anterior se desprende que el modelo relacional posee una ventaja clara en lo tocante a los lenguajes. Tanto los lenguajes formales como los comerciales asociados al modelo relacional son muy potentes y de alto nivel. La existencia de una norma SQL es otra gran ventaja, aunque dicha norma está en continua evolución. Los trabajos actuales sobre una norma SQL3 extienden el SQL2 con características y estructuras orientadas a objetos. Varios SGBD de red y jerárquicos han implementado lenguajes de consulta de alto nivel que se asemejan a los lenguajes relacionales y que pueden usarse con sus sistemas junto con las órdenes de DML tradicionales.

En los sistemas OO, el DML suele estar incorporado en algún lenguaje de programación OO, como C++. Por ello, las estructuras de los objetos persistentes almacenados y de los objetos transitorios del lenguaje de programación casi siempre son compatibles. Este rasgo suele considerarse como una gran ventaja para integrar servicios de base de datos en sistemas de software complejos. Se han desarrollado lenguajes de consulta para bases de datos OO. Los trabajos sobre un modelo y lenguaje OO estándar están avanzando, pero todavía no ha surgido una norma completa y detallada.

C.3 Comparación de estructuras de almacenamiento

En los capítulos 4 y 5 estudiamos algunas estructuras de almacenamiento que emplean los SGBD para almacenar físicamente la base de datos. Por mucho, la técnica actual más importante es la indexación. Como vimos, hay una gran variedad de opciones para implementar índices, desde los árboles B* hasta los índices densos. También se están usando cada vez más técnicas de dispersión.

Para el modelo relacional, la técnica general consiste en implementar cada relación base como un archivo individual. Si el usuario no especifica ninguna estructura de almacenamiento, la mayor parte de los SGBD almacenarán las tuplas como registros del archivo sin ningún orden. Muchos SGBD relacionales permiten al usuario especificar dinámicamente para cada archivo un solo índice primario o de agrupamiento y cualquier cantidad de índices secundarios. El usuario se encarga de escoger los atributos sobre los cuales se crean los índices. Un número creciente de SGBD permiten especificar técnicas de dispersión como organización primaria para almacenar registros y también como estructuras de indexación. En general, se recomienda crear índices (o atributos clave) para todos los atributos que se usarán con frecuencia en las condiciones de selección o de reunión. Los índices y claves de dispersión únicos también constituyen una técnica eficiente para imponer restricciones de clave. Además, la mayor parte de los SGBD relacionales permiten al usuario desechar índices dinámicamente.

Algunos SGBD relacionales ofrecen al usuario la opción de mezclar registros de varias relaciones base, cosa que resulta útil cuando lo usual es tener acceso a registros de más de una relación juntos. Este agrupamiento de registros coloca físicamente un registro de una relación seguido de los registros relacionados de otra relación de modo que éstos se puedan obtener de la manera más eficiente posible. Esto también suele hacerse en los sistemas jerárquicos y de red, que fueron los primeros en usar este tipo de estructuras.

El *modelo de red* casi siempre se implementa con apuntadores y listas circulares enlazadas (archivos de anillo). Casi todos los SGBD de red ofrecen además la opción de implementar algunos conjuntos con agrupamiento; esto es, con el registro propietario seguido físicamente de los registros miembro en cada ejemplar de conjunto. Este agrupamiento de los registros miembro junto a su registro propietario puede hacerse sólo para un tipo de conjuntos en el que un tipo de registros dado participe como miembro, porque sólo podemos agrupar físicamente los registros miembro con base en un solo tipo de vínculos lógicos 1:N. El tipo de conjuntos que se utilice con mayor frecuencia para tener acceso a los registros es el que se deberá usar para el agrupamiento físico. En muchos casos, también puede implementarse la indicación o la dispersión según ciertos atributos de un tipo de registros sobre el archivo de anillo para tener acceso rápido a registros individuales de un cierto tipo.

El *modelo jerárquico* por lo regular se implementa con archivos jerárquicos, que conservan la secuencia jerárquica de la base de datos. Además están disponibles diversas opciones, como dispersión, indicación y apuntadores, para tener acceso eficiente a registros individuales y a los registros relacionados con ellos. Casi todos los sistemas jerárquicos ofrecen muchas de estas opciones para "afinar" el rendimiento de un sistema de bases de datos.

Los *sistemas OO* proveen almacenamiento persistente para objetos de estructura compleja. Por lo regular emplean técnicas de indicación para localizar las páginas de disco donde están almacenados los objetos. Éstos suelen almacenarse como cadenas de bytes, y su estructura se reconstruye después de copiar en almacenamiento intermedio del sistema las páginas de disco que contienen el objeto. Varios sistemas OO —y relacionales— están usando alguna forma de arquitectura cliente-servidor.

C.4 Comparación de restricciones de integridad

Las restricciones del *modelo relacional* incluyen las de clave, las de integridad de entidades y las de integridad referencial. En la norma SQL2 están disponibles diferentes *opciones de comportamiento* para las claves externas, como PROPAGATE (propagar) y SET NULL (hacer nulo). También se puede utilizar aserciones generales para especificar otros tipos de restricciones. Así, el modelo relacional ha avanzado mucho respecto a sus versiones iniciales en cuanto a incorporar mecanismos útiles para imponer restricciones de integridad.

El *modelo jerárquico* incluye la restricción jerárquica inherente de que un tipo de registros sólo puede tener un padre real en una jerarquía. En los SGBD individuales existen otras restricciones; por ejemplo, IMS sólo permite un padre virtual para cada tipo de registros. No *hay un mecanismo* para imponer la consistencia entre registros repetidos; los programas de aplicación que actualizan la base de datos deben encargarse de imponerla. Es posible especificar restricciones de clave. La restricción implícita de que un registro hijo debe estar relacionado con un registro padre sí se hace cumplir; es más, los registros hijo se eliminan automáticamente cuando se elimina su padre o un antepasado.

Las restricciones del *modelo de red* incluyen opciones de retención en conjuntos, que especifican si todo registro miembro debe tener un propietario (MANDATORY o FIXED) o no (OPTIONAL). Los tipos de conjuntos automáticos para los que se especifica SET SELECTION IS STRUCTURAL (la selección de conjuntos es estructural) igualan el campo clave de un propietario con un campo del registro miembro. Puede usarse la opción CHECK para especificar una restricción similar que se aplique a tipos de conjuntos no automáticos (MANUAL). Las restricciones de clave se especifican con la cláusula DUPLICATES NOT ALLOWED (no se permiten duplicados). De este modo, es posible especificar muchas restricciones estructurales sobre los tipos de vínculos en un SGBD de red. Desafortunadamente, no todos los SGBD de red actuales cuentan con estas características.

Las restricciones que se manejan en los *sistemas OO* varían de un sistema a otro. La filosofía de que las restricciones se deben especificar por procedimientos en los métodos, si bien es bastante general, dificulta determinar cuáles restricciones se especificaron y comprobar la consistencia de las restricciones. El mecanismo de vínculo inverso que manejan algunos sistemas OO constituye un buen principio para proveer algunas restricciones declarativas. Es posible que, conforme los modelos y sistemas de bases de datos OO maduren, incorporen más de las restricciones usuales, como sucedió con el modelo relacional.

Formato de las citas bibliográficas

Los títulos de libros aparecen en negritas; por ejemplo, **Database Computers**. Los nombres de actas de conferencias aparecen en cursivas; por ejemplo, *NCC* o *Proceedings of the ACM Pacific Conference*. Los nombres de publicaciones periódicas aparecen en negritas; por ejemplo, *TODS* o **Information Systems**. En el caso de citas de publicaciones periódicas, damos el volumen y el número (dentro del volumen, si lo hay) así: volumen:número, y la fecha de publicación. Por ejemplo "TODS, 3:4, diciembre de 1978" se refiere al número de diciembre de 1978 de ACM Transactions on Database Systems, que es el volumen 3, número 4. Los artículos que aparecen en libros o actas de conferencias y que se citan individualmente en la bibliografía se indican con la palabra "en" seguida de la referencia; por ejemplo, "en VLDB [1978]" o "en Rustin [1974]". En el caso de citas con más de cuatro autores daremos sólo el primer autor seguido de *et al.* En las secciones de bibliografía selecta al final de cada capítulo usamos *et al.* si se trataba de más de dos autores. Todos los nombres de autores se dan con una sola inicial (por ejemplo, Codd, E. en vez de Codd, E.F.).

Referencias bibliográficas

- Abott, R. y García-Molina, H. [1989] "Scheduling Real-Time Transactions with Disk Resident Data", en VLDB [1989].
- Abiteboul, S. y Kanellakis, P. [1989] "Object Identity as a Query Language Primitive", en SIGMOD [1989].
- Abrial, J. [1974] "Data Semantics", en Klimbie y Koffeman [1974].
- Adam, N. y Gongopadhyay, A. [1993] "Integrating Functional and Data Modeling in a Computer Integrated Manufacturing System", en ICDE [1993].
- Afsarmanesh, H., McLeod, D., Knapp, D. y Parker, A. [1985] "An Extensible Object-Oriented Approach to Databases for VLSI/CAD", en VLDB [1985].
- Agrawal, D. y Elabbadi, A. [1990] "Storage Efficient Replicated Databases", TKDE, 2:3, septiembre de 1990.
- Agrawal, R. y Gehani, N. [1989] "ODE: The Language and the Data Model", en SIGMOD [1989].
- Agrawal, R., Gehani, H. y Srinivasan, J. [1990] "OdeView: The Graphical Interface to Ode", en SIGMOD [1990].
- Ahad, R. y Basu, A. [1991] "ESQL: A Query Language for the Relational Model Supporting Image Domains", en ICDE [1991].
- Ahmed, R. y Navathe, S. [1991] "Version Management of Composite Objects in CAD Databases", en SIGMOD [1991].
- Aho, A., Beeri, C. y Ullman, J. [1979] "The Theory of Joins in Relational Databases", TODS, 4:3, septiembre de 1979.
- Aho, A., Sagiv, Y. y Ullman, J. [1979a] "Efficient Optimization of a Class of Relational Expressions", TODS, 4:4, diciembre de 1979.
- Aho, A. y Ullman, J. [1979] "Universality of Data Retrieval Languages", *Proceedings of the POPL Conference*, San Antonio, Texas, ACM, 1979.
- Akl, S. [1983] "Digital Signatures: A Tutorial Survey", *IEEE Computer*, 16:2, febrero de 1983.
- Alashkur, A., Su, S. y Lam, H. [1989] "OQL: A Query Language for Manipulating Object-Oriented Databases", en VLDB [1989].

Bibliografía selecta

Abreviaturas empleadas en la bibliografía

- ACM: Association for Computing Machinery.
- AFIPS: American Federation of Information Processing Societies.
- CACM: Communications of the ACM (publicación periódica).
- CIKM: Actas de la Conferencia internacional sobre gestión de información y conocimientos (International Conference on Information and Knowledge Management).
- EDS: Actas de la Conferencia internacional sobre sistemas de bases de datos expertos (International Conference on Expert Database Systems).
- ER Conference: Actas de la Conferencia internacional sobre el enfoque de entidad-vínculo (International Conference on Entity-Relationship Approach).
- ICDE: Actas de la Conferencia internacional del IEEE sobre ingeniería de datos (IEEE International Conference on Data Engineering).
- IEEE: Institute of Electrical and Electronics Engineers.
- IEEE Computer: Computer magazine (publicación periódica) de la IEEE CS.
- IEEE CS: IEEE Computer Society.
- IFIP: International Federation for Information Processing.
- JACM: Journal of the ACM.
- NCC: Actas de la National Computer Conference (publicadas por AFIPS).
- OOPSLA: Actas de la Conferencia sobre sistemas, lenguajes y aplicaciones de programación orientada a objetos de la ACM (ACM Conference on Object-Oriented Programming Systems, Languages and Applications).
- PODS: Actas del Simposio sobre principios de los sistemas de bases de datos de la ACM (ACM Symposium on Principles of Database Systems).
- SIGMOD: Actas de la Conferencia internacional ACM SIGMOD sobre gestión de datos (ACM SIGMOD International Conference on Management of Data).
- TKDE: IEEE Transactions on Knowledge and Data Engineering (publicación periódica).
- TOCS: ACM Transactions on Computer Systems (publicación periódica).
- TODS: ACM Transactions on Database Systems (publicación periódica).
- TOOIS: ACM Transactions on Office Information Systems (publicación periódica).
- TSE: IEEE Transactions on Software Engineering (publicación periódica).
- VLDB: Actas de la Conferencia internacional sobre bases de datos muy grandes (International Conference on Very Large Data Bases). (Los números posteriores a 1981 se pueden obtener de Morgan Kaufmann, Menlo Park, California).

- Albano, A., Cardelli, L. y Orsini, R. [1985] "GALILEO: A Strongly-Typed Interactive Conceptual Language", TODS, 10:2, junio de 1985.
- Albano, A., de Antonellis, V. y di Leva, A. (editores) [1985a] **Computer-Aided Database Design: The DATAID Project**, North-Holland, 1985.
- Allen, F., Loomis, M. y Mannino, M. [1982] "The Integrated Dictionary/Directory System", ACM Computing Surveys, 14:2, junio de 1982.
- Andrews, T. y Harris, C. [1987] "Combining Language and Database Advances in an Object-Oriented Development Environment", OOPSLA, 1987.
- ANSI [1975] American National Standards Institute Study Group on Data Base Management Systems: Interim Report, FDT, 7:2, ACM, 1975.
- ANSI [1986] American National Standards Institute: The Database Language SQL, documento ANSI X3.135, 1986.
- ANSI [1986a] American National Standards Institute: The Database Language NDL, documento ANSI X3.133, 1986.
- ANSI [1989] American National Standards Institute: Information Resource Dictionary Systems, documento ANSI X3.138, 1989.
- Anwar, T., Beck, H. y Navathe, S. [1992] "Knowledge Mining by Imprecise Querying: A Classification Based Approach", en ICDE [1992].
- Apers, P., Hevner, A. y Yao, S. [1983] "Optimization Algorithms for Distributed Queries", TSE, 9:1, enero de 1983.
- Armstrong, W. [1974] "Dependency Structures of Data Base Relationships", *Proceedings of the IFIP Congress*, 1974.
- Astrahan, M. et al. [1976] "System R: A Relational Approach to Data Base Management", TODS, 1:2, junio de 1976.
- Atkinson, M. y Buneman, P. [1987] "Types and Persistence in Database Programming Languages", en ACM Computing Surveys, 19:2, junio de 1987.
- Atre, S. [1980] **Structured Techniques for Design, Performance and Management**, Wiley, 1980.
- Atzeni, P. y de Antonellis, V. [1993] **Relational Database Theory**, Benjamin/Cummings, 1993.
- Bachman, C. [1969] "Data Structure Diagrams", **Data Base** (Bulletin of ACM SIGFIDET), 1:2, marzo de 1969.
- Bachman, C. [1973] "The Programmer as a Navigator", CACM, 16:1, noviembre de 1973.
- Bachman, C. [1974] "The Data Structure Set Model", en Rustin [1974].
- Bachman, C. y Williams, S. [1964] "A General Purpose Programming System for Random Access Memories", *Proceedings of the Fall Joint Computer Conference*, AFIPS, 26, 1964.
- Badal, D. y Popek, G. [1979] "Cost and Performance Analysis of Semantic Integrity Validation Methods", en SIGMOD [1979].
- Badrinath, B. y Ramamirtham, K. [1992] "Semantics-Based Concurrency Control: Beyond Commutativity", TODS, 17:1, marzo de 1992.
- Baeza-Yates, R. y Larson, P. [1989] "Performance of B+-trees With Partial Expansions", TKDE, 1:2, junio de 1989.
- Bancilhon, F. y Buneman, P. (editores) [1990] **Advances in Database Programming Languages**, ACM Press, 1990.
- Bancilhon, F., Maier, D., Sagiv, Y. y Ullman, J. [1986] "Magic sets and other strange ways to implement logic programs", PODS [1986].
- Bancilhon, F. y Ramakrishnan, R. [1986] "An Amateur's Introduction to Recursive Query Processing Strategies", en SIGMOD [1986].
- Banerjee, J. et al. [1987] "Data Model Issues for Object-Oriented Applications", TODS, 5:1, enero de 1987.

- Banerjee, J., Kim, W., Kim, H. y Korth, H. [1987a] "Semantics and Implementation of Schema Evolution in Object-Oriented Databases", en SIGMOD [1987].
- Baroody, A. y DeWitt, D. [1981] "An Object-Oriented Approach to Database System Implementation", TODS, 6:4, diciembre de 1981.
- Barsalou, T., Siamela, N., Keller, A. y Wiederhold, G. [1991] "Updating Relational Databases Through Object-Based Views", en SIGMOD [1991].
- Bassiouni, M. [1988] "Single-Site and Distributed Optimistic Protocols for Concurrency Control", TSE, 14:8, agosto de 1988.
- Batini, C., Ceri, S. y Navathe, S. [1992] **Database Design: An Entity-Relationship Approach**, Benjamin/Cummings, 1992. (Existe versión en español por Addison-Wesley Iberoamericana.)
- Batini, C., Lenzerini, M. y Navathe, S. [1986] "A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computing Surveys, 18:4, diciembre de 1986.
- Batory, D. y Buchmann, A. [1984] "Molecular Objects, Abstract Data Types, and Data Models: A Framework", en VLDB [1984].
- Batory, D. et al. [1988] "GENESIS: An Extensible Database Management System", TSE, 14:11, noviembre de 1988.
- Bayer, R., Graham, M. y Seegmuller, G. (editores) [1978] **Operating Systems: An Advanced Course**, Springer-Verlag, 1978.
- Bayer, R. y McCreight, E. [1972] "Organization and Maintenance of Large Ordered Indexes", Acta Informatica, 1:3, febrero de 1972.
- Beck, H., Anwar, T. y Navathe, S. [1993] "A Conceptual Clustering Algorithm for Database Schema Design", TKDE, en prensa.
- Beck, H., Gala, S. y Navathe, S. [1989] "Classification as a Query Processing Technique in the CANDIDE Semantic Data Model", en ICDE [1989].
- Beeri, C., Fagin, R. y Howard, J. [1977] "A Complete Axiomatization for Functional and Multivalued Dependencies", en SIGMOD [1977].
- Beeri, C. y Ramakrishnan, R. [1987] "On the Power of Magic", en PODS [1987].
- Ben-Zvi, J. [1982] "The Time Relational Model", tesis de doctorado, University of California, Los Angeles, 1982.
- Berg, B. y Rooth, J. [1989] **Software for Optical Disk**, Meckler, 1989.
- Bernstein, P. [1976] "Synthesizing Third Normal Form Relations from Functional Dependencies", TODS, 1:4, diciembre de 1976.
- Bernstein, P., Blaustein, B. y Clarke, E. [1980] "Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data", en VLDB [1980].
- Bernstein, P. y Goodman, N. [1980] "Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems", en VLDB [1980].
- Bernstein, P. y Goodman, N. [1981] "The Power of Natural Semijoins", SIAM Journal of Computing, 10:4, diciembre de 1981.
- Bernstein, P. y Goodman, N. [1981a] "Concurrency Control in Distributed Database Systems", ACM Computing Surveys, 13:2, junio de 1981.
- Bernstein, P. y Goodman, N. [1984] "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases", TODS, 9:4, diciembre de 1984.
- Bernstein, P., Hadzilacos, V. y Goodman, N. [1988] **Concurrency Control and Recovery in Database Systems**, Addison-Wesley, 1988.
- Bertino, E. [1992] "Data Hiding and Security in Object-Oriented Databases", en ICDE [1992].
- Bertino, E., Negri, M., Pelagatti, G. y Shatella, L. [1992] "Object-Oriented Query Languages: The Notion and the Issues", TKDE, 4:3, junio de 1992.
- Bertino, E. y Kim, W. [1989] "Indexing Techniques for Queries on Nested Objects", TKDE, 1:2, junio de 1989.

- Bertino, E., Rabbitti, F. y Gibbs, S. [1988] "Query Processing in a Multimedia Environment", *TOIS*, 6, 1988.
- Bhargava, B. (editor) [1987] **Concurrency and Reliability in Distributed Systems**, Van Nostrand Reinhold, 1987.
- Bhargava, B. y Helal, A. [1993] "Efficient Reliability Mechanisms in Distributed Database Systems", *CIKM*, noviembre de 1993.
- Bhargava, B. y Reidl, J. [1988] "A Model for Adaptable Systems for Transaction Processing", en *ICDE* [1988].
- Biliris, A. [1992] "The Performance of Three Database Storage Structures for Managing Large Objects", en *SIGMOD* [1992].
- Billir, H. [1979] "On the Equivalence of Data Base Schemas—A Semantic Approach to Data Translation", *Information Systems*, 4:1, 1979.
- Bjork, A. [1973] "Recovery Scenario for a DB/DC System", *Proceedings of the ACM National Conference*, 1973.
- Björner, D. y Lovengren, H. [1982] "Formalization of Database Systems and a Formal Definition of IMS", en *Vldb* [1982].
- Blakeley, J., Coburn, N. y Larson, P. [1989] "Updated Derived Relations: Detecting Irrelevant and Autonomously Computable Updates", *rods*, 14:3, septiembre de 1989.
- Blakeley, J. y Martin, N. [1990] "Join Index, Materialized View, and Hybrid-Hash Join: A Performance Analysis", en *ICDE* [1990].
- Blankinship, R., Hevner, A. y Yao, S. [1991] "An Iterative Method for Distributed Database Design", en *Vldb* [1991].
- Blasgen, M. y Eswaran, K. [1976] "On the Evaluation of Queries in a Relational Database System", *IBM Systems Journal*, 16:1, enero de 1976.
- Blasgen, M. et al. [1981] "System R: An Architectural Overview", *IBM Systems Journal*, 20:1, enero de 1981.
- Bleier, R. y Vorhaus, A. [1968] "File Organization in the SDC TDMS", *Proceedings of the IFIP Congress*, 1968.
- Bocca, J. [1986] "EDUCE—A Marriage of Convenience: Prolog and a Relational DBMS", *Proceedings of the Third International Conference on Logic Programming*, Springer-Verlag, 1986.
- Bocca, J. [1986a] "On the Evaluation Strategy of EDUCE", en *SIGMOD* [1986].
- Bodnick, P., Riordon, J. y Pyra, J. [1992] "Deciding on Correct Distributed Query Processing", *TKDE*, 4:3, junio de 1992.
- Borgida, A., Brachman, R., McGuiness, D. y Resnick, L. [1989] "CLASSIC: A Structural Data Model for Objects", en *SIGMOD* [1989].
- Borkin, S. [1978] "Data Model Equivalence", en *Vldb* [1978].
- Bouzghoub, M. y Merais, E. [1991] "Semantic Modelling of Object-Oriented Databases", en *Vldb* [1991].
- Boyce, R., Chamberlin, D., King, W. y Hammer, M. [1975] "Specifying Queries as Relational Expressions", *CACM*, 18:11, noviembre de 1975.
- Bracchi, G., Paolini, P. y Pelagatti, G. [1976] "Binary Logical Associations in Data Modelling", en *Niessen* [1976].
- Bracchi, G. y Pernici, B. [1984] "The Design Requirements of Office Systems", *roods*, 2:2, abril de 1984.
- Bracchi, G. y Pernici, B. [1987] "Decision Support in Office Information Systems", en *Holsapple y Whinston* [1987].
- Brachman, R. y Levesque, H. [1984] "What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level", en *EDS* [1984].
- Bradley, J. [1978] "An Extended Owner-Coupled Set Data Model and Predicate Calculus for Database Management", *roods*, 3:4, diciembre de 1978.
- Bray, O. [1988] **Computer Integrated Manufacturing—The Data Management Strategy**, Digital Press, 1988.
- Breitbart, Y., Silberschatz, A. y Thompson, G. [1990] "Reliable Transaction Management in a Multibase System", en *SIGMOD* [1990].
- Brodie, M. y Mylopoulos, J. (editores) [1985] **On Knowledge Base Management Systems**, Springer-Verlag, 1985.
- Brodie, M., Mylopoulos, J. y Schmidt, J. (editores) [1984] **On Conceptual Modeling**, Springer-Verlag, 1984.
- Brossey, M. y Schneiderman, B. [1978] "Two Experimental Comparisons of Relational and Hierarchical Database Models", *International Journal of Man-Machine Studies*, 1978.
- Bry, F. [1990] "Query Evaluation in Recursive Databases: Bottom-up and Top-down Reconciled", *TKDE*, 2, 1990.
- Bubenko, J., Berild, S., Lindercrona-Olin, E. y Nachmens, S. [1976] "From Information Requirements to DTG Data Structures", *Proceedings of the ACM SIGMOD/SIGPLAN Conference on Data Abstraction*, 1976.
- Bubenko, J., Langefors, B. y Solberg, A. (editores) [1971] **Computer-Aided Information Systems Analysis and Design**, Studentlitteratur, Lund, Suecia, 1971.
- Bukhres, O. [1992] "Performance Comparison of Distributed Deadlock Detection Algorithms", en *ICDE* [1992].
- Buneman, P. y Frankel, R. [1979] "FQL: A Functional Query Language", en *SIGMOD* [1979].
- Bush, V. [1945] "As We May Think", *The Atlantic Monthly*, 176:1, enero de 1945 (reimpreso en Kochen, M. (editor) **The Growth of Knowledge**, Wiley, 1967).
- Cammarata, S., Ramachandra, P. y Shane, D. [1989] "Extending a Relational Database with Deferred Referential Integrity Checking and Intelligent Joins", en *SIGMOD* [1989].
- Campbell, D., Embley, D. y Czejdo, B. [1985] "A Relationally Complete Query Language for the Entry-Relationship Model", en *ER Conference* [1985].
- Cardenas, A. [1985] **Data Base Management Systems**, segunda edición, Allyn and Bacon, 1985.
- Carey, M. et al. [1986] "The Architecture of the EXODUS extensible DBMS", en *Dirtrich y Dayal* [1986].
- Carey, M., De Witt, D., Richardson, J. y Shekita, E. [1986a] "Object and File Management in the EXODUS Extensible Database System", en *Vldb* [1986].
- Carey, M., DeWitt, D. y Vandenberg, S. [1988] "A Data Model and Query Language for Exodus", en *SIGMOD* [1988].
- Carey, M., Franklin, M., Livny, M. y Shekita, E. [1991] "Data Caching Tradeoffs in Client-Server DBMS Architectures", en *SIGMOD* [1991].
- Carlis, J. [1986] "HAS, a Relational Algebra Operator or Divide Is Not Enough to Conquer", en *ICDE* [1986].
- Carlis, J. y March, S. [1984] "A Descriptive Model of Physical Database Design Problems and Solutions", en *ICDE* [1984].
- Casanova, M., Fagin, R. y Papadimitriou, C. [1981] "Inclusion Dependencies and Their Interaction with Functional Dependencies", *roods*, 1981.
- Casanova, M., Furtado, A. y Tucherann, L. [1991] "A Software Tool for Modular Database Design", *roods*, 16:2, junio de 1991.
- Casanova, M., Tucherann, L., Furtado, A. y Braga, A. [1989] "Optimization of Relational Schemas Containing Inclusion Dependencies", en *Vldb* [1989].
- Casanova, M. y Vidal, V. [1982] "Toward a Sound View Integration Method", *roods*, 1982.
- Cattell, R. y Skeen, J. [1992] "Object Operations Benchmark", *roods*, 17:1, marzo de 1992.

- Ceri, S. (editor) [1983] *Methodology and Tools for Database Design*, North-Holland, 1983.
- Ceri, S., Gottlob, G. y Tanca, L. [1990] *Logic Programming and Databases*, Springer-Verlag, 1990.
- Ceri, S., Navathe, S. y Wiederhold, G. [1983] "Distribution Design of Logical Database Schemas", *TSE*, 9:4, julio de 1983.
- Ceri, S., Negri, M. y Pelagatti, G. [1982] "Horizontal Data Partitioning in Database Design", en *SIGMOD* [1982].
- Ceri, S. y Owicki, S. [1983] "On the Use of Optimistic Methods for Concurrency Control in Distributed Databases", *Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks*, febrero de 1983.
- Ceri, S. y Pelagatti, G. [1984] "Correctness of Query Execution Strategies in Distributed Databases", *TODS*, 8:4, diciembre de 1984.
- Ceri, S. y Pelagatti, G. [1984a] *Distributed Databases: Principles and Systems*, McGraw-Hill, 1984.
- Ceri, S. y Tanaka, L. [1987] "Optimization of Systems of Algebraic Equations for Evaluating Data log Queries", en *Vldb* [1987].
- Cesarini, F. y Soda, G. [1991] "A Dynamic Hash Method With Signature", *TODS*, 16:2, junio de 1991.
- Chakravathy, S. [1990] "Active Database Management Systems: Requirements, State-of-the-Art, and an Evaluation", en *ER Conference* [1990].
- Chakravathy, S. [1991] "Divide and Conquer: A Basis for Augmenting a Conventional Query Optimizer with Multiple Query Processing Capabilities", en *ICDE* [1991].
- Chakravathy, S. et al. [1989] "HiPAC: A Research Project in Active, Time Constrained Database Management", Final Technical Report, XAIT-89-02, Xerox Advanced Information Technology, agosto de 1989.
- Chakravathy, S., Karlapalem, K., Navathe, S. y Tanaka, A. [1993] "Database Supported Cooperative Problem Solving", en *International Journal of Intelligent Co-operative Information Systems*, 2:3, septiembre de 1993.
- Chakravathy, U., Grant, J. y Minker, J. [1990] "Logic-Based Approach to Semantic Query Optimization", *TODS*, 15:2, junio de 1990.
- Chalmers, M. y Chitson, P. [1992] "Bead: Explorations in Information Visualization", *Proceedings of the ACM SIGIR International Conference*, junio de 1992.
- Chamberlin, D. y Boyce, R. [1974] "SEQUEL: A Structured English Query Language", en *SIGMOD* [1974].
- Chamberlin, D. et al. [1976] "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control", *IBM Journal of Research and Development*, 20:6, noviembre de 1976.
- Chamberlin, D. et al. [1981] "A History and Evaluation of System R", *CACM*, 24:10, octubre de 1981.
- Chan, C., Ooi, B. y Lu, H. [1992] "Extensible Buffer Management of Indexes", en *Vldb* [1992].
- Chandy, K., Browne, J., Disley, C. y Uhrig, W. [1975] "Analytical Models for Rollback and Recovery Strategies in Database Systems", *TSE* 1:1, marzo de 1975.
- Chang, C. [1981] "On the Evaluation of Queries Containing Derived Relations in a Relational Data Base", en Gallaire et al. [1981].
- Chang, C. y Walker, A. [1984] "PROSQL: A Prolog Programming Interface with SQL/DS", en *EDS* [1984].
- Chang, E. y Karz, R. [1989] "Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in Object-Oriented Databases", en *SIGMOD* [1989].
- Chang, N. y Fu, K. [1981] "Picture Query Languages for Pictorial Databases", *IEEE Computer*, 14:11, noviembre de 1981.
- Chang, P. y Myre, W. [1988] "OS/2 EE Database Manager: Overview and Technical Highlights", *IBM Systems Journal*, 27:2, 1988.
- Chang, S., Lin, B. y Walser, R. [1979] "Generalized Zooming Techniques for Pictorial Database Systems", *NCC, AFIPS*, 48, 1979.

- Chen, M. y Yu, P. [1991] "Determining Beneficial Semijoins for a Join Sequence in Distributed Query Processing", en *ICDE* [1991].
- Chen, P. [1976] "The Entry Relationship Model—Toward a Unified View of Data", *TODS*, 1:1, marzo de 1976.
- Chen, Q. y Kambayashi, Y. [1991] "Nested Relation Based Database Knowledge Representation", en *SIGMOD* [1991].
- Cheng J. [1991] "Effective Clustering of Complex Objects in Object-Oriented Databases", en *SIGMOD* [1991].
- Childs, D. [1968] "Feasibility of a Set Theoretical Data Structure—A General Structure Based on a Reconstituted Definition of Relation", *Proceedings of the IFIP Congress*, 1968.
- Chimenti, D. et al. [1987] "An Overview of the LDL System", MCC Technical Report #ACA-ST-370-87, Austin, Texas, noviembre de 1987.
- Chimenti, D. et al. [1990] "The LDL System Prototype", *TKDE*, 2:1, marzo de 1990.
- Chin, F. [1978] "Security in Statistical Databases for Queries with Small Counts", *TODS*, 3:1, marzo de 1978.
- Chin, F. y Osoyoglu, G. [1981] "Statistical Database Design", *TODS*, 6:1, marzo de 1981.
- Chou, H. y Kim, W. [1986] "A Unifying Framework for Version Control in a CAD Environment", en *Vldb* [1986].
- Christodoulakis, S. et al. [1984] "Development of a Multimedia Information System for an Office Environment", en *Vldb* [1984].
- Christodoulakis, S. y Faloutsos, C. [1986] "Design and Performance Considerations for an Optical Disk-Based Multimedia Object Server", *IEEE Computer*, 19:12, diciembre de 1986.
- Chu, W. y Hurely, P. [1982] "Optimal Query Processing for Distributed Database Systems", *IEEE Transactions on Computers*, 31:9, septiembre de 1982.
- Ciborra, C., Migliarese, P. y Romano, P. [1984] "A Methodological Inquiry of Organizational Noise in Socio Technical Systems", *Human Relations*, 37:8, 1984.
- Claybrook, B. [1983] *File Management Techniques*, Wiley, 1983.
- Claybrook, B. [1992] *OLRP: OnLine Transaction Processing Systems*, Wiley, 1992.
- Clifford, J. y Tansel, A. [1985] "On an Algebra for Historical Relational Databases: Two Views", en *SIGMOD* [1985].
- CODASYL [1978] *Data Description Language Journal of Development*, Canadian Government Publishing Centre, 1978.
- Codd, E. [1970] "A Relational Model for Large Shared Data Banks", *CACM*, 13:6, junio de 1970.
- Codd, E. [1971] "A Data Base Sublanguage Founded on the Relational Calculus", *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access and Control*, noviembre de 1971.
- Codd, E. [1972] "Relational Completeness of Data Base Sublanguages", en Rustin [1972].
- Codd, E. [1972a] "Further Normalization of the Data Base Relational Model", en Rustin [1972].
- Codd, E. [1974] "Recent Investigations In Relational Database Systems", *Proceedings of the IFIP Congress*, 1974.
- Codd, E. [1978] "How About Recently? (English Dialog with Relational Data Bases Using Rendevous Version 1)", en Shneiderman [1978].
- Codd, E. [1979] "Extending the Database Relational Model to Capture More Meaning", *TODS*, 4:4, diciembre de 1979.
- Codd, E. [1982] "Relational Database: A Practical Foundation for Productivity", *CACM*, 25:2, diciembre de 1982.
- Codd, E. [1985] "Is Your DBMS Really Relational?" y "Does Your DBMS Run By the Rules", *COMPUTER WORLD*, 14 de octubre y 21 de octubre de 1985.
- Codd, E. [1986] "An Evaluation Scheme for Database Management Systems That Are Claimed to be Relational", en *ICDE* [1986].

- Codd, E. [1990] **Relational Model for Data Management-Version 2**, Addison-Wesley, 1990.
- Comer, D. [1979] "The Ubiquitous B-tree", *ACM Computing Surveys*, 11:2, junio de 1979.
- Cornelio, A. y Navathe, S. [1993] "Applying Active Database Models for Simulation", en *Proceedings of 1993 Winter Simulation Conference*, Los Angeles, IEEE, diciembre de 1993.
- Cruz, J. [1992] "Doodle: A Visual Language for Object-Oriented Databases", en SIGMOD [1992].
- Curtice, R. [1981] "Data Dictionaries: An Assessment of Current Practice and Problems", en VLDB [1981].
- Czejdo, B., Elmarsi, R., Rusinkiewicz, M. y Embley, D. [1987] "An Algebraic Language for Graphical Query Formulation Using an Extended Entity-Relationship Model", *Proceedings of the ACM Computer Science Conference*, 1987.
- Dahl, R. y Bubenko, J. [1982] "IDBD: An Interactive Design Tool for CODASYL DBTG Type Databases", en VLDB [1982].
- Dahl, V. [1984] "Logic Programming for Constructive Database Systems", en EDS [1984].
- Date, C. [1983] **An Introduction to Database Systems**, vol. 2, Addison-Wesley, 1983.
- Date, C. [1983a] "The Outer Join", *Proceedings of the Second International Conference on Databases (ICOD-2)*, 1983.
- Date, C. [1984] "A Critique of the SQL Database Language", *ACM SIGMOD Record*, 14:3, noviembre de 1984.
- Date, C. [1990] **An Introduction to Database Systems**, vol. 1, 5a. ed., Addison-Wesley, 1990. (Existe versión en español por Addison-Wesley Iberoamericana.)
- Date, C. y White, C. [1988] **A Guide to SQL/DS**, Addison-Wesley, 1988.
- Date, C. y White, C. [1989] **A Guide to DB2**, 3a. ed., Addison-Wesley, 1989.
- Davies, C. [1973] "Recovery Semantics for a DB/DC System", *Proceedings of the ACM National Conference*, 1973.
- Dayal, U. y Bernstein, P. [1978] "On the Updatability of Relational Views", en VLDB [1978].
- Dayal, U., Hsu, M. y Ladin, R. [1991] "A Transaction Model for Long-Running Activities", en VLDB [1991].
- Daval, U. et al. [1987] "PROBE Final Report", Technical Report CCA-87-02, Computer Corporation of America, diciembre de 1987.
- DBTG [1971] Report of the CODASYL Data Base Task Group, ACM, abril de 1971.
- Delcambre, L., Lim, B. y Urban, S. [1991] "Object-Centered Constraints", en ICDE [1991].
- DeMarco, T. [1979] **Structured Analysis and System Specification**, Prentice-Hall Yourdon Inc., 1979.
- DeMichiel, L. [1989] "Performing Operations Over Mismatched Domains", en ICDE [1989].
- Denning, D. [1980] "Secure Statistical Databases with Random Sample Queries", *TODS*, 5:3, septiembre de 1980.
- Denning, D. y Denning, P. [1979] "Data Security", *ACM Computing Surveys*, 11:3, septiembre de 1979.
- Devor, C. y Weeldreyer, J. [1980] "DBTS: A Testbed for Distributed Database Research", *Proceedings of the ACM Pacific Conference*, 1980.
- DeWitt, D. et al. [1984] "Implementation Techniques for Main Memory Databases", en SIGMOD [1984].
- DeWitt, D. et al. [1990] "The Gamma Database Machine Project", *TKDE*, 2:1, marzo de 1990.
- DeWitt, D., Fattersack, P., Maier, D. y Velez, F. [1990] "A Study of Three Alternative Workstation Server Architectures for Object-Oriented Database Systems", en VLDB [1990].
- Diffie, W. y Hellman, M. [1979] "Privacy and Authentication", *Proceedings of the IEEE*, 67:3, marzo de 1979.

- Dittrich, K. [1986] "Object-Oriented Database Systems: The Notion and the Issues", en Dittrich y Dayal [1986].
- Dittrich, K. y Dayal, U. (editores) [1986] *Proceedings of the International Workshop on Object-Oriented Database Systems*, IEEE CS, Pacific Grove, California, septiembre de 1986.
- Dittrich, K., Kotz, A. y Mülle, J. [1986] "An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases", en SIGMOD Record, 15:3, 1986.
- Dodd, G. [1969] "API—A Language for Associative Data Handling in PL/I", *Proceedings of the Fall Joint Computer Conference*, AFIPS, 29, 1969.
- Dodd, G. [1969a] "Elements of Data Management Systems", *ACM Computing Surveys*, 1:2, junio de 1969.
- Dos Santos, C., Neuhold, E. y Furtado, A. [1979] "A Data Type Approach to the Entity-Relationship Model", en ER Conference [1979].
- Du, D. y Tong, S. [1991] "Multilevel Extendible Hashing: A File Structure for Very Large Databases", *TKDE*, 3:3, septiembre de 1991.
- Du, H. y Ghanta, S. [1987] "A Framework for Efficient IC/LSI CAD Databases", en ICDE [1987].
- Dumas, P. et al. [1982] "MOBILE-Burotique: Prospects for the Future", en Naffah [1982].
- Dumpala, S. y Arora, S. [1983] "Schema Translation Using the Entity-Relationship Approach", en ER Conference [1983].
- Dwyer, S. et al. [1982] "A Diagnostic Digital Imaging System", *Proceedings of the IEEE CS Conference on Pattern Recognition and Image Processing*, junio de 1982.
- Eastman, C. [1981] "Database Facilities for Engineering Design", *Proceedings of the IEEE*, 69:10, octubre de 1981.
- EDS [1984] **Expert Database Systems**, Kerschberg, L. (editor), (*Proceedings of the First International Workshop on Expert Database Systems*, Kiawah Island, Carolina del Sur, octubre de 1984), Benjamin/Cummings, 1986.
- EDS [1986] **Expert Database Systems**, Kerschberg, L. (editor), (*Proceedings of the First International Conference on Expert Database Systems*, Charleston, Carolina del Sur, abril de 1986), Benjamin/Cummings, 1987.
- EDS [1988] **Expert Database Systems**, Kerschberg, L. (editor), (*Proceedings of the Second International Conference on Expert Database Systems*, Tysons Corner, Virginia, abril de 1988), Benjamin/Cummings, (en prensa).
- Eick, C. y Lockemann, P. [1985] "Acquisition of Terminological Knowledge Using Database Design Techniques", en SIGMOD [1985].
- Eick, C. [1991] "A Methodology for the Design and Transformation of Conceptual Schemas", en VLDB [1991].
- ElAbbadi, A. y Toueg, S. [1988] "The Group Paradigm for Concurrency Control", en SIGMOD [1988].
- ElAbbadi, A. y Toueg, S. [1989] "Maintaining Availability in Partitioned Replicated Databases", *TODS*, 14:2, junio de 1989.
- Ellis, C. y Nutt, G. [1980] "Office Information Systems and Computer Science", *ACM Computing Surveys*, 12:1, marzo de 1980.
- Elmagarmid, A. y Helal, A. [1988] "Supporting Updates in Heterogeneous Distributed Databases Systems", en ICDE [1988].
- Elmagarmid, A., Leu, Y., Litwin, W. y Rusinkiewicz, M. [1990] "A Multidatabase Transaction Model for Interbase", en VLDB [1990].
- Elmasri, R., James, S. y Kouramajian, V. [1993] "Automatic Class and Method Generation for Object-Oriented Databases", *Proceedings of the Third International Conference on Deductive and Object-Oriented Databases (DOOD-93)*, Phoenix, Arizona, diciembre de 1993.

- Elmasri, R., Kouramajian, V. y Fernando, S. [1993] "Temporal Database Modeling: An Object-Oriented Approach", *CIKM*, noviembre de 1993.
- Elmasri, R. y Larson, J. [1985] "A Graphical Query Facility for ER Databases", en *ER Conference* [1985].
- Elmasri, R., Larson, J. y Navathe, S. [1986] "Schema Integration Algorithms for Federated Databases and Logical Database Design", Honeywell CSDD, Technical Report CSC-86-9-8212, enero de 1986.
- Elmasri, R. y Navathe, S. [1984] "Object Integration in Logical Database Design", en *ICDE* [1984].
- Elmasri, R., Srinivas, P. y Thomas, G. [1987] "Fragmentation and Query Decomposition in the ECR Model", en *ICDE* [1987].
- Elmasri, R., Weeldreyer, J. y Hevner, A. [1985] "The Category Concept: An Extension to the Entity-Relationship Model", *International Journal on Data and Knowledge Engineering*, 1:1, mayo de 1985.
- Elmasri, R. y Wiederhold, G. [1979] "Data Model Integration Using the Structural Model", en *SIGMOD* [1979].
- Elmasri, R. y Wiederhold, G. [1980] "Structural Properties of Relationships and Their Representation", *NCC, AFPS*, 49, 1980.
- Elmasri, R. y Wiederhold, G. [1981] "GORDAS: A Formal, High-Level Query Language for the Entity-Relationship Model", en *ER Conference* [1981].
- Elmasri, R. y Wu, G. [1990] "A Temporal Model and Query Language for ER Databases", en *ICDE* [1990], en *Vldb* [1990].
- Engelbart, D. y English, W. [1968] "A Research Center for Augmenting Human Intellect", *Proceedings of the Fall Joint Computer Conference, AFPS*, diciembre de 1968.
- Epstein, R., Stonebraker, M. y Wong, E. [1978] "Distributed Query Processing in a Relational Database System", en *SIGMOD* [1978].
- ER Conference [1979] **Entity-Relationship Approach to Systems Analysis and Design**, Chen, P. (editor), (*Proceedings of the First International Conference on Entity-Relationship Approach*, Los Angeles, California, diciembre de 1979), North-Holland, 1980.
- ER Conference [1981] **Entity-Relationship Approach to Information Modeling and Analysis**, Chen, P. (editor), (*Proceedings of the Second International Conference on Entity-Relationship Approach*, Washington, D.C., octubre de 1981), Elsevier Science, 1981.
- ER Conference [1983] **Entity-Relationship Approach to Software Engineering**, Davis, C., Iajodia, S., Ng, P. y Yeh, R. (editores), (*Proceedings of the Third International Conference on Entity-Relationship Approach*, Anaheim, California, octubre de 1983), North-Holland, 1983.
- ER Conference [1985] *Proceedings of the Fourth International Conference on Entity-Relationship Approach*, Liu, J. (editor), Chicago, Illinois, octubre de 1985, IEEE CS.
- ER Conference [1986] *Proceedings of the Fifth International Conference on Entity-Relationship Approach*, Spaccapietra, S. (editor), Dijon, Francia, noviembre de 1986, Express-Trages.
- ER Conference [1987] *Proceedings of the Sixth International Conference on Entity-Relationship Approach*, March, S. (editor), Nueva York, Nueva York, noviembre de 1987.
- ER Conference [1988] *Proceedings of the Seventh International Conference on Entity-Relationship Approach*, Batini, C. (editor), Roma, Italia, noviembre de 1988.
- ER Conference [1989] *Proceedings of the Eighth International Conference on Entity-Relationship Approach*, Lochovsky, F. (editor), Toronto, Canada, octubre de 1989.
- ER Conference [1990] *Proceedings of the Ninth International Conference on Entity-Relationship Approach*, Kangassalo, H. (editor), Lausana, Suiza, septiembre de 1990.
- ER Conference [1991] *Proceedings of the Tenth International Conference on Entity-Relationship Approach*, Teorey, T. (editor), San Mateo, California, octubre de 1991.

- ER Conference [1992] *Proceedings of the Eleventh International Conference on Entity-Relationship Approach*, Pernul, G. y Tjoa, A. (editores), Karlsruhe, Alemania, octubre de 1992.
- ER Conference [1993] *Proceedings of the Twelfth International Conference on Entity-Relationship Approach*, Elmasri, R. y Kouramajian, V. (editores), Arlington, Texas, diciembre de 1993.
- Eswaran, K. y Chamberlin, D. [1975] "Functional Specifications of a Subsystem for Database Integrity", en *Vldb* [1975].
- Eswaran, K., Gray, J., Lorie, I. y Traiger, I. [1976] "The Notions of Consistency and Predicate Locks in a Data Base System", *CACM*, 19:11, noviembre de 1976.
- Everett, G., Dissly, C. y Hardgrave, W. [1981] *RIMS User Manual*, TRM-16, Computing Center, University of Texas at Austin, 1981.
- Fagin, J. [1987] "Experiments in Automatic Phrase Indexing for Document Retrieval: A Comparison of Syntactic and Non-Syntactic Methods", disertación de doctorado, Departamento de ciencias de la computación, Cornell University, 1987.
- Fagin, R. [1977] "Multivalued Dependencies and a New Normal Form for Relational Databases", *TODS*, 2:3, septiembre de 1977.
- Fagin, R. [1979] "Normal Forms and Relational Database Operators", en *SIGMOD* [1979].
- Fagin, R. [1981] "A Normal Form for Relational Databases That is Based on Domains and Keys", *TODS*, 6:3, septiembre de 1981.
- Fagin, R., Nievergelt, J., Pippenger, N. y Strong, H. [1979] "Extendible Hashing—A Fast Access Method for Dynamic Files", *TODS*, 4:3, septiembre de 1979.
- Faloutsos, G. y Jagadish, H. [1992] "On B-Tree Indices for Skewed Distributions", en *Vldb* [1992].
- Faraq, W. y Teorey, T. [1993] "FunBase: A Function-based Information Management System", *CIKM*, noviembre de 1993.
- Fernandez, E., Summers, R. y Wood, C. [1981] **Database Security and Integrity**, Addison-Wesley, 1981.
- Ferrier, A. y Stangret, C. [1982] "Heterogeneity in the Distributed Database Management System SIRIUS-DELTA", en *Vldb* [1982].
- Fishman, D. et al. [1986] "IRIS: An Object-Oriented DBMS", *TOOLS*, 4:2, abril de 1986.
- Ford, D., Blakeley, J. y Bannon, T. [1993] "Open OODB: A Modular Object-Oriented DBMS", en *SIGMOD* [1993].
- Ford, D. y Christodoulakis, S. [1991] "Optimizing Random Retrievals from CLV Format Optical Disks", en *Vldb* [1991].
- Franszek, P., Robinson, J. y Thomasian, A. [1992] "Concurrency Control for High Contention Environments", *TODS*, 17:2, junio de 1992.
- Franklin, F. et al. [1992] "Crash Recovery in Client-Server EXODUS", en *SIGMOD* [1992].
- French, C., Jones, K. y Pfaltz, J.L. (editores) [1990] "Scientific Database Management (Final Report)", Complete Science Report núm. TR-90-2-1, Departamento de ciencias de la computación, University of Virginia, agosto de 1990.
- Frenkel, K. [1991] "The Human Genome Project and Informatics", *CACM*, noviembre de 1991.
- Fry, J. y Sibley, E. [1976] "Evolution of Data-Base Management Systems", *ACM Computing Surveys*, 8:1, marzo de 1976.
- Furrado, A. [1978] "Formal Aspects of the Relational Model", *Information Systems*, 3:2, 1978.
- Gadua, S. [1988] "A Homogeneous Relational Model and Query Language for Temporal Databases", *TODS*, 13:4, diciembre de 1988.
- Gait, J. [1988] "The Optical File Cabinet: A Random Access File System for Write-Once Optical Disks", *IEEE Computer*, 21:6, junio de 1988.
- Gallaire, H. y Minker, J. (editores) [1978] **Logic and Databases**, Plenum Press, 1978.
- Gallaire, H., Minker, J. y Nicolas, J. [1984] "Logic and Databases: A Deductive Approach", *ACM Computing Surveys*, 16:2, junio de 1984.

- Gallaire, H., Minker, J. y Nicolas, J. (editores) [1981] *Advances in Database Theory*, vol. 1, Plenum Press, 1981.
- Gamal-Eldin, M., Thomas, G. y Elmarsi, R. [1988] "Integrating Relational Databases with Support for Updates", *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems*, IEEE CS, diciembre de 1988.
- Gane, C. y Sarson, T. [1977] *Structured Systems Analysis: Tools and Techniques*, Improved Systems Technologies Inc., 1977.
- García-Molina, H. [1978] "Performance Comparison of Two Update Algorithms for Distributed Databases", *Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks*, IEEE CS, febrero de 1978.
- García-Molina, H. [1982] "Elections in Distributed Computing Systems", *IEEE Transactions on Computers*, 31:1, enero de 1982.
- García-Molina, H. [1983] "Using Semantic Knowledge for Transaction Processing in a Distributed Database", *TODS*, 8:2, junio de 1983.
- Gehani, N., Jagadish, H. y Shmueli, O. [1992] "Composite Event Specification in Active Databases: Model and Implementation", en *Vldb* [1992].
- Georgakopoulos, D., Rusinkiewicz, M. y Sheth, A. [1991] "On Serializability of Multidatabase Transactions Through Forced Local Conflicts", en *ICDE* [1991].
- Gerritsen, R. [1975] "A Preliminary System for the Design of DRTG Data Structures", *CACM*, 18:10, octubre de 1975.
- Ghosh, S. [1984] "An Application of Statistical Databases in Manufacturing Testing", en *ICDE* [1984].
- Ghosh, S. [1986] "Statistical Data Reduction for Manufacturing Testing", en *ICDE* [1986].
- Gifford, D. [1979] "Weighted Voting for Replicated Data", *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, 1979.
- Gladney, H. [1989] "Data Replicas in Distributed Information Services", *TODS*, 14:1, marzo de 1989.
- Gogolla, M. y Hohenstein, U. [1991] "Towards a Semantic View of an Extended Entity-Relationship Model", *TODS*, 16:3, septiembre de 1991.
- Goldberg, A. y Robson, D. [1983] *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, 1983.
- Goldfine, A. y König, P. [1988] A Technical Overview of the Information Resource Dictionary System (IRDS), 2da. ed., NBS IR 88-3700, National Bureau of Standards.
- Gordon, C. [1988] "Report of Icon class Workshop, November 1987", en *Visual Resources: an International Journal of Documentation*, 5, 1988.
- Gottlieb, L. [1975] "Computing Joins of Relations", en *SIGMOD* [1975].
- Graefe, G. y DeWitt, D. [1987] "The EXODUS Optimizer Generator", en *SIGMOD* [1987].
- Gray, J. [1978] "Nores on Data Base Operating Systems", en Bayer, Graham y Seegmüller [1978].
- Gray, J. [1981] "The Transaction Concept: Virtues and Limitations", en *Vldb* [1981].
- Gray, J., Lorie, R. y Putzulo, G. [1975] "Granularity of Locks and Degrees of Consistency in a Shared Data Base", en Nijssen [1976].
- Gray, J., McJones, P. y Blasgen, M. [1981] "The Recovery Manager of the System R Database Manager", *ACM Computing Surveys*, 13:2, junio de 1981.
- Gray, J. y Reuter, A. [1993] *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.
- Griffiths, P. y Wade, B. [1976] "An Authorization Mechanism for a Relational Database System", *TODS*, 1:3, septiembre de 1976.
- Guttman, A. [1984] "R-Trees: A Dynamic Index Structure for Spatial Searching", en *SIGMOD* [1984].
- Gyssens, M., Paredaens, J. y Van Gucht, D. [1989] "A Grammar-Based Approach Towards Unifying Hierarchical Data Models", en *SIGMOD* [1989].

- Gyssens, M., Paredaens, J. y Van Gucht, D. [1990] "A Graph-Oriented Object Model for Database End-User Interfaces", en *SIGMOD* [1990].
- Haas, L., Freytag, J., Lohman, G. y Pirahesh, H. [1989] "Extensible Query Processing in STARBURST", en *SIGMOD* [1989].
- Hachem, N. y Berra, P. [1992] "New Order Preserving Access Methods for Very Large Files Derived from Linear Hashing", *TKDE*, 4:1, febrero de 1992.
- Haerder, T. y Rothermel, K. [1987] "Concepts for Transaction Recovery in Nested Transactions", en *SIGMOD* [1987].
- Hall, P. [1976] "Optimization of a Single Relational Expression in a Relational Data Base System", *IBM Journal of Research and Development*, 20:3, mayo de 1976.
- Hammer, M. y McLeod, D. [1975] "Semantic Integrity in a Relational Database System", en *Vldb* [1975].
- Hammer, M. y McLeod, D. [1981] "Database Description with SDM: A Semantic Data Model", *TODS*, 6:3, septiembre de 1981.
- Hammer, M. y Sarin, S. [1978] "Efficient Monitoring of Database Assertions", en *SIGMOD* [1978].
- Hanson, E. [1992] "Rule Condition Testing and Action Execution in Ariel", en *SIGMOD* [1992].
- Hardgrave, W. [1974] "BOLT: A Retrieval Language for Tree-Structured Database Systems", en *TOU* [1974].
- Hardgrave, W. [1980] "Ambiguity in Processing Boolean Queries on TDMS Tree Structures: A Study of Four Different Philosophies", *TSE*, 6:4, julio de 1980.
- Hardwick, M. y Spooner, D. [1989] "ROSE Data Manager: Using Object Technology to Support Interactive Engineering Applications", *TKDE*, 1:2, 1989.
- Harrington, J. [1987] *Relational Database Management for Microcomputers: Design and Implementation*, Holt Rinehart Winston, 1987.
- Harris, L. [1978] "The ROBOT System: Natural Language Processing Applied to Data Base Query", *Proceedings of the ACM National Conference*, diciembre de 1978.
- Haskin, R. y Lorie, R. [1982] "On Extending the Functions of a Relational Database System", en *SIGMOD* [1982].
- Hasse, C. y Weikum, G. [1991] "A Performance Evaluation of Multi-Level Transaction Management", en *Vldb* [1991].
- Hayes-Roth, F., Waterman, D. y Lenat, D. (editores) [1983] *Building Expert Systems*, Addison-Wesley, 1983.
- Hayne, S. y Ram, S. [1990] "Multi-User View Integration System: An Expert System for View Integration", en *ICDE* [1990].
- Heiler, S. y Zdonick, S. [1990] "Object Views: Extending the Vision", en *ICDE* [1990].
- Helal, A., Hu, T., Elmarsi, R. y Mukherjee, S. [1993] "Adaptive Transaction Scheduling", *CIKM*, noviembre de 1993.
- Held, G. y Stonebraker, M. [1978] "B-Trees Reexamined", *CACM*, 21:2, febrero de 1978.
- Hendrix, G., Sacerdoti, D. y Slocum, J. [1978] "Developing a Natural Language Interface to Complex Data", *TODS*, 3:2, junio de 1978.
- Hernandez, H. y Chan, E. [1991] "Constraint-Time-Maintainable BCNF Database Schemes", *TODS*, 16:4, diciembre de 1991.
- Herot, C. [1980] "Spatial Management of Data", *TODS*, 5:4, diciembre de 1980.
- Hewner, A. y Yao, S. [1979] "Query Processing in Distributed Database Systems", *TSE*, 5:3, mayo de 1979.
- Hoffer, J. [1982] "An Empirical Investigation with Individual Differences in Database Models", *Proceedings of the Third International Information Systems Conference*, diciembre de 1982.
- Holsapple, C. y Whinston, A. (editores) [1987] *Decisions Support Systems Theory and Application*, Springer-Verlag, 1987.

- Hsiao, D. y Kamel, M. [1989] "Heterogeneous Databases: Proliferation, Issues, and Solutions", *TKDE*, 1:1, marzo de 1989.
- Hsu, A. e Imielinski, T. [1985] "Integrity Checking for Multiple Updates", en *SIGMOD* [1985].
- Hull, R. y King, R. [1987] "Semantic Database Modeling: Survey, Applications, and Research Issues", *ACM Computing Surveys*, 19:3, septiembre de 1987.
- IBM [1978] QBE Terminal Users Guide, forma número SH20-2078-0.
- IBM [1978a] QBE Quick Reference Card, forma número JX20-2030-0.
- IBM [1992] Systems Application Architecture Common Programming Interface Database Level 2 Reference, documento número SC26-4798-01.
- ICDE [1984] *Proceedings of the IEEE CS International Conference on Data Engineering*, Berra, P. (editor), Los Angeles, California, abril de 1984.
- ICDE [1986] *Proceedings of the IEEE CS International Conference on Data Engineering*, Wiederhold, G. (editor), Los Angeles, California, febrero de 1986.
- ICDE [1987] *Proceedings of the IEEE CS International Conference on Data Engineering*, Wah, B. (editor), Los Angeles, California, febrero de 1987.
- ICDE [1988] *Proceedings of the IEEE CS International Conference on Data Engineering*, Cartis, J. (editor), Los Angeles, California, febrero de 1988.
- ICDE [1989] *Proceedings of the IEEE CS International Conference on Data Engineering*, Shuey, R. (editor), Los Angeles, California, febrero de 1989.
- ICDE [1990] *Proceedings of the IEEE CS International Conference on Data Engineering*, Liu, M. (editor), Los Angeles, California, febrero de 1990.
- ICDE [1991] *Proceedings of the IEEE CS International Conference on Data Engineering*, Cercone, N. y Tsuchiya, M. (editores), Kobe, Japón, abril de 1991.
- ICDE [1992] *Proceedings of the IEEE CS International Conference on Data Engineering*, Golshani, F. (editor), Phoenix, Arizona, febrero de 1992.
- ICDE [1993] *Proceedings of the IEEE CS International Conference on Data Engineering*, Elmagarmid, A. y Neuhold, E. (editores), Viena, Austria, abril de 1993.
- ICES [1983] International Graphics Exchange Specification Version 2, National Bureau of Standards, U.S. Department of Commerce, enero de 1983.
- Imielinski, T. y Lipski, W. [1981] "On Representing Incomplete Information in a Relational Database", en *VLD8* [1981].
- Interbase [1990] InterBase DDL Reference Manual: Interbase Version 3.0, Interbase Software Corporation, 1990.
- Ioannidis, Y. y Kang, Y. [1990] "Randomized Algorithms for Optimizing Large Join Queries", en *SIGMOD* [1990].
- Ioannidis, Y. y Wong, E. [1988] "Transforming Non-Linear Recursion to Linear Recursion", en *EDS* [1988].
- Iossophidis, J. [1979] "A Translator to Convert the DDL of ERM to the DDL of System 2000", en *ER Conference* [1979].
- Irani, K., Purkayastha, S. y Teorey, T. [1979] "A Designer for DBMS-Processable Logical Database Structures", en *VLD8* [1979].
- Jagadish, H. [1989] "Incorporating Hierarchy in a Relational Model of Data", en *SIGMOD* [1989].
- Jajodia, S. y Murchler, D. [1990] "Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database", *TODS*, 15:2, junio de 1990.
- Jajodia, S., Ng, P. y Springsteel, F. [1983] "The Problem of Equivalence for Entity-Relationship Diagrams", *TSE*, 9:5, septiembre de 1983.
- Jajodia, S. y Sandhu, R. [1991] "Toward a Multilevel Secure Relational Data Model", en *SIGMOD* [1991].
- Jardine, D. (editor) [1977] *The ANSI/SPARC DBMS Model*, North-Holland, 1977.

- Jarke, M. y Koch, J. [1984] "Query Optimization in Database Systems", *ACM Computing Surveys*, 16:2, junio de 1984.
- Jensen, C. y Snodgrass, R. [1992] "Temporal Specialization", en *ICDE* [1992].
- Johnson, T. y Shasha, D. [1993] "The Performance of Current B-Tree Algorithms", *TODS*, 18:1, marzo de 1993.
- Kaefer, W. y Schoening, H. [1992] "Realizing a Temporal Complex-Object Data Model", en *SIGMOD* [1992].
- Kamel, I. y Faloutsos, C. [1993] "On Packing R-trees", *CIKM*, noviembre de 1993.
- Kamel, N. y King, R. [1985] "A Model of Data Distribution Based on Texture Analysis", en *SIGMOD* [1985].
- Kapp, D. y Leben, J. [1978] *IMS Programming Techniques*, Van Nostrand-Reinhold, 1978.
- Kappel, G. y Schrefl, M. [1991] "Object/Behavior Diagrams", en *ICDE* [1991].
- Katz, R. [1985] *Information Management for Engineering Design*, Surveys in Computer Science, Springer-Verlag, 1985.
- Katz, R. y Wong, E. [1982] "Decompiling CODASYL DML into Relational Queries", *TODS*, 7:1, marzo de 1982.
- Kaul, M., Drost, K. y Neuhold, E. [1990] "ViewSystem: Integrating Heterogeneous Information Bases by Object-Oriented Views", en *ICDE* [1990].
- Kedem, Z. y Silberschatz, A. [1980] "Non-Two Phase Locking Protocols with Shared and Exclusive Locks", en *VLD8* [1980].
- Keller, A. [1982] "Updates to Relational Database Through Views Involving Joins", en Scheuermann [1982].
- Kemper, A., Lockemann, P. y Wallrath, M. [1987] "An Object-Oriented Database System for Engineering Applications", en *SIGMOD* [1987].
- Kemper, A., Moerkotte, G. y Steinbrunn, M. [1992] "Optimizing Boolean Expressions in Object Bases", en *VLD8* [1992].
- Kemper, A. y Wallrath, M. [1987] "An Analysis of Geometric Modeling in Database Systems", *ACM Computing Surveys*, 19:1, marzo de 1987.
- Kent, W. [1978] *Data and Reality*, North-Holland, 1978.
- Kent, W. [1979] "Limitations of Record-Based Information Models", *TODS*, 4:1, marzo de 1979.
- Kent, W. [1991] "Object-Oriented Database Programming Languages", en *VLD8* [1991].
- Kerschberg, L., Ting, P. y Yao, S. [1982] "Query Optimization in Star Computer Networks", *TODS*, 7:4, diciembre de 1982.
- Ketafchi, M. y Bertins, V. [1986] "Component Aggregation: A Mechanism for Organizing Efficient Engineering Databases", en *ICDE* [1986].
- Kifer, M. y Lozinskii, E. [1986] "A Framework for an Efficient Implementation of Deductive Databases", *Proceedings of the Sixth Advanced Database Symposium*, Tokio, Japón, agosto de 1986.
- Kim, W. [1982] "On Optimizing an SQL-like Nested Query", *TODS*, 3:3, septiembre de 1982.
- Kim, W. [1989] "A Model of Queries for Object-Oriented Databases", en *VLD8* [1989].
- Kim, W. [1990] "Object-Oriented Databases: Definition and Research Directions", *TKDE*, 2:3, septiembre de 1990.
- Kim, W., Garza, J., Ballou, N. y Woelk, D. [1990] "Architecture of the ORION Next Generation Database System", *TKDE*, 2:1, marzo de 1990.
- Kim, W., Reiner, D. y Batory, D. (editores) [1985] *Query Processing in Database Systems*, Springer-Verlag, 1985.
- Kim, W. et al. [1987] "Features of the ORION Object-Oriented Database System", *Microelectronics and Computer Technology Corporation, Technical Report ACA-ST-308-87*, septiembre de 1987.
- King, J. [1981] "QUIST: A System for Semantic Query Optimization in Relational Databases", en *VLD8* [1981].

- Kitsuregawa, M., Nakayama, M. y Takagi, M. [1989] "The Effect of Bucket Size Tuning in the Dynamic Hybrid GRACE Hash Join Method", en *Vldb* [1989].
- Klimbie, J. y Koffeman, K. (editores) [1974] *Data Base Management*, North-Holland, 1974.
- Klug, A. [1982] "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions", *JACM*, 29:3, julio de 1982.
- Knuth, D. [1973] *The Art of Computer Programming*, vol. 3: *Sorting and Searching*, Addison-Wesley, 1973.
- Kogan, B. y Jajodia, S. [1990] "Concurrency Control in Multilevel Secure Databases Based on Replicated Architecture", en *SIGMOD* [1990].
- Kohler, W. [1981] "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems", *ACM Computing Surveys*, 13:2, junio de 1981.
- Kolodner, J. [1993] *Case Based Reasoning*, Morgan Kaufmann, 1993.
- Konsynski, B., Bracker, L. y Bracker, W. [1982] "A Model for Specification of Office Communications", *IEEE Transactions on Communications*, 30:1, enero de 1982.
- Korfhage, R. [1991] "To see, or Not to see: Is that the Query", en *Proceedings of the ACM SIGIR International Conference*, junio de 1991.
- Korth, H. [1983] "Locking Primitives in a Database System", *JACM*, 30:1, enero de 1983.
- Korth, H., Levy, E. y Silberschatz, A. [1990] "A Formal Approach to Recovery by Compensating Transactions", en *Vldb* [1990].
- Korth, H. y Silberschatz, A. [1991] *Database System Concepts*, 2da. ed., McGraw-Hill, 1991.
- Kotz, A., Dittrich, K. y Mülle, J. [1988] "Supporting Semantic Rules by a Generalized Event/Trigger Mechanism", en *Vldb* [1988].
- Krishnamurthy, R., Litwin, W. y Kent, W. [1991] "Language Features for Interoperability of Databases with Semantic Discrepancies", en *SIGMOD* [1991].
- Krishnamurthy, R. y Naqvi, S. [1988] "Database Updates in Logic Programming, Rev. 1", *MCC Technical Report #ACA-ST-010-88*, Rev. 1, septiembre de 1988.
- Krishnamurthy, R. y Naqvi, S. [1989] "Non-Deterministic Choice in Datalog", *Proceedings of the 3rd International Conference on Data and Knowledge Bases*, junio 27-30, Jerusalén, Israel.
- Kroenke, D. y Dolan, K. [1988] *Database Processing*, 3ra. ed., Science Research Associates, 1988.
- Krovetz, R. y Croft, B. [1992] "Lexical Ambiguity and Information Retrieval", en *TOIS*, 10, abril de 1992.
- Kumar, A. [1991] "Performance Measurement of Some Main Memory Recovery Algorithms", en *ICDE* [1991].
- Kumar, A. y Segev, A. [1993] "Cost and Availability Tradeoffs in Replicated Concurrency Control", *TODS*, 18:1, marzo de 1993.
- Kumar, A. y Stonebraker, M. [1987] "Semantics Based Transaction Management Techniques for Replicated Data", en *SIGMOD* [1987].
- Kung, H. y Robinson, J. [1981] "Optimistic Concurrency Control", *TODS*, 6:2, junio de 1981.
- Kunii T. y Harada, M. [1980] "SD: A System for Interactive Design", *NCC, AFIPS*, 49, junio de 1980.
- Lacroix, M. y Protte, A. [1977] "Domain-Oriented Relational Languages", en *Vldb* [1977].
- Lacroix, M. y Protte, A. [1977a] "ILL: An English Structured Query Language for Relational Data Bases", en *Nijssen* [1977].
- Lafue, G. y Smith, R. [1984] "Implementation of a Semantic Integrity Manager with a Knowledge Representation System", en *EDS* [1984].
- Lamport, L. [1978] "Time, Clocks, and the Ordering of Events in a Distributed System", *CACM*, 21:7, julio de 1978.
- Langerak, R. [1990] "View Updates in Relational Databases with an Independent Scheme", *TODS*, 15:1, marzo de 1990.
- Lanka, S. y Mays, E. [1991] "Fully Persistent B+-Trees", en *SIGMOD* [1991].

- Larson, J. [1983] "Bridging the Gap Between Network and Relational Database Management Systems", *IEEE Computer*, 16:9, septiembre de 1983.
- Larson, J., Navathe, S. y Elmasri, R. [1989] "Attribute Equivalence and its Use in Schema Integration", *TSE*, 15:2, abril de 1989.
- Larson, P. [1978] "Dynamic Hashing", *BIT*, 18, 1978.
- Larson, P. [1981] "Analysis of Index-Sequential Files with Overflow Chaining", *TODS*, 6:4, diciembre de 1981.
- Laurini, R. y Thompson, D. [1992] *Fundamentals of Spatial Information Systems*, Academic Press, 1992.
- Lehmann, P. y Yao, S. [1981] "Efficient Locking for Concurrent Operations on B-Trees", *TODS*, 6:4, diciembre de 1981.
- Lehman, T. y Lindsay, B. [1989] "The Starburst Long Field Manager", en *Vldb* [1989].
- Leiss, E. [1982] "Randomizing, A Practical Method for Protecting Statistical Databases Against Compromise", en *Vldb* [1982].
- Leiss, E. [1982a] *Principles of Data Security*, Plenum Press, 1982.
- Lenzerini, M. y Santucci, C. [1983] "Cardinality Constraints in the Entity-Relationship Model", en *ER Conference* [1983].
- Leung, C., Hibler, B. y Mwara, N. [1992] "Picture Retrieval by Content Description", en *Journal of Information Science*, 1992, págs. 111-119.
- Levesque, H. [1984] "The Logic of Incomplete Knowledge Bases", capítulo 7 de Brodie et al. [1984].
- Lien, E. y Weinberger, P. [1978] "Consistency, Concurrency and Crash Recovery", en *SIGMOD* [1978].
- Lieuwen, L. y DeWitt, D. [1992] "A Transformation-Based Approach to Optimizing Loops in Database Programming Languages", en *SIGMOD* [1992].
- Lilien, L. y Bhargava, B. [1985] "Database Integrity Block Construct: Concepts and Design Issues", *TSE*, 11:9, septiembre de 1985.
- Lindsay, B. et al. [1984] "Computation and Communication in R: A Distributed Database Manager", *TODS*, 2:1, enero de 1984.
- Lipton, R., Naughton, J. y Schneider, D. [1990] "Practical Selectivity Estimation through Adaptive Sampling", en *SIGMOD* [1990].
- Lipski, W. [1979] "On Semantic Issues Connected with Incomplete Information", *TODS*, 4:3, septiembre de 1979.
- Liskov, B. y Zilles, S. [1975] "Specification Techniques for Data Abstractions", *TSE*, 1:1, marzo de 1975.
- Liskov, B. et al. [1981] *clu Reference Manual*, Lecture Notes in Computer Science, Springer-Verlag, 1981.
- Litwin, W. [1980] "Linear Hashing: A New Tool for File and Table Addressing", en *Vldb* [1980].
- Liu, K. y Sunderraman, R. [1988] "On Representing Indefinite and Maybe Information in Relational Databases", en *ICDE* [1988].
- Liu, L. y Meersman, R. [1992] "Activity Model: A Declarative Approach for Capturing Communication Behavior in Object-Oriented Databases", en *Vldb* [1992].
- Livadas, P. [1989] *File Structures: Theory and Practice*, Prentice-Hall, 1989.
- Lockemann, P. y Knutsen, W. [1968] "Recovery of Disk Contents after System Failure", *CACM*, 11:8, agosto de 1968.
- Lorie, R. [1977] "Physical Integrity in a Large Segmented Database", *TODS*, 2:1, marzo de 1977.
- Lorie, R. y Plouffe, W. [1983] "Complex Objects and Their Use in Design Transactions", en *SIGMOD* [1983].
- Lozinskii, E. [1986] "A Problem-Oriented Inferential Database System", *TODS*, 11:3, septiembre de 1986.

- Lu, H., Mikilineni, K. y Richardson, J. [1987] "Design and Evaluation of Algorithms to Compute the Transitive Closure of a Database Relation", en ICDE [1987].
- Lucyk, B. [1993] *Advanced Topics in DB2*, Addison-Wesley, 1993.
- Maier, D. [1983] *The Theory of Relational Databases*, Computer Science Press, 1983.
- Maier, D., Stein, J., Otis, A. y Purdy, A. [1986] "Development of an Object-Oriented DBMS", OOPSLA, 1986.
- Malley, C. y Zdonick, S. [1986] "A Knowledge-Based Approach to Query Optimization", en EDS [1986].
- Mannino, M. y Effelsberg, W. [1984] "Matching Techniques in Global Schema Design", en ICDE [1984].
- March, S. y Severance, D. [1977] "The Determination of Efficient Record Segmentations and Blocking Factors for Shared Files", TODS, 2:3, septiembre de 1977.
- Mark, L., Roussopoulos, N., Newsome, T. y Loahapattana, P. [1992] "Incrementally Maintained Network to Relational Mappings", *Software Practice & Experience*, 22:12, diciembre de 1992.
- Markowitz, V. y Raz, Y. [1983] "EROL: An Entity-Relationship, Role Oriented, Query Language", en ER Conference [1983].
- Martin, J., Chapman, K. y Leben, J. [1989] *DB2 Concepts, Design and Programming*, Prentice-Hall, 1989.
- Maryanski, F. [1980] "Backend Database Machines", *ACM Computing Surveys*, 12:1, marzo de 1980.
- Masunaga, Y. [1987] "Multimedia Databases: A Formal Framework", *Proceedings of the IEEE Office Automation Symposium*, abril de 1987.
- McFadden, F. y Hoffer, J. [1988] *Database Management*, 2da. ed., Benjamin/Cummings, 1988.
- McGee, W. [1977] "The Information Management System IMS/VS, Part I: General Structure and Operation", *IBM Systems Journal*, 16:2, junio de 1977.
- McLeish, M. [1989] "Further Results on the Security of Partitioned Dynamic Statistical Databases", TODS, 14:1, marzo de 1989.
- McLeod, D. y Heimbigner, D. [1985] "A Federated Architecture for Information Systems", TODS, 3:3, julio de 1985.
- Mehrotra, S. et al. [1992] "The Concurrency Control Problem in Multidatabases: Characteristics and Solutions", en SIGMOD [1992].
- Menace, D., Popek, G. y Muntz, R. [1980] "A Locking Protocol for Resource Coordination in Distributed Databases", TODS, 5:2, junio de 1980.
- Mendelson, A. y Maier, D. [1979] "Generalized Mutual Dependencies and the Decomposition of Database Relations", en VLDB [1979].
- Mikilineni, K. y Su, S. [1988] "An Evaluation of Relational Join Algorithms in a Pipelined Query Processing Environment", TSE, 14:6, junio de 1988.
- Miller, N. [1987] *File Structures Using PASCAL*, Benjamin/Cummings, 1987.
- Minoura, T. y Wiederhold, G. [1981] "Resilient Extended True-Copy Token Scheme for a Distributed Database", TSE, 8:3, mayo de 1981.
- Missikoff, M. y Wiederhold, G. [1984] "Toward a Unified Approach for Expert and Database Systems", en EDS [1984].
- Mitschang, B. [1989] "Extending the Relational Algebra to Capture Complex Objects", en VLDB [1989].
- Mohan, C. [1993] "IBM's Relational Database Products: Features and Technologies", en SIGMOD [1993].
- Mohan, C. y Levine, F. [1992] "ARIEL/IM: An Efficient and High-Concurrency Index Management Method Using Write-Ahead Logging", en SIGMOD [1992].
- Mohan, C. y Narang, I. [1992] "Algorithms for Creating Indexes for Very Large Tables without Quiescing Updates", en SIGMOD [1992].

- Mohan, C. et al. [1992] "ARIEL: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging", TODS, 17:1, marzo de 1992.
- Morgan, H. y Levin, K. [1977] "Optimal Program and Data Locations in Computer Networks", CACM, 20:5, mayo de 1977.
- Morris, K., Ullman, J. y VanGelden, A. [1986] "Design Overview of the NAIL System", *Proceedings of the Third International Conference on Logic Programming*, Springer-Verlag, 1986.
- Morris, K. et al. [1987] "YAWN! (Yet Another Window on NAIL)", en ICDE [1987].
- Morris, R. [1968] "Scatter Storage Techniques", CACM, 11:1, enero de 1968.
- Morsi, M., Navathe, S. y Kim, H. [1992] "An Extensible Object-Oriented Database Testbed", en ICDE [1992].
- Moss, J. [1982] "Nested Transactions and Reliable Distributed Computing", *Proceedings of the Symposium on Reliability in Distributed Software and Database Systems*, IEEE CS, julio de 1982.
- Motro, A. [1987] "Supervisors: Virtual Integration of Multiple Databases", TSE, 13:7, julio de 1987.
- Mukkamala, R. [1989] "Measuring the Effect of Data Distribution and Replication Models on Performance Evaluation of Distributed Systems", en ICDE [1989].
- Mumick, I., Finkelstein, S., Pirahesh, H. y Ramakrishnan, R. [1990] "Magic is Relevant", en SIGMOD [1990].
- Mumick, I., Pirahesh, H. y Ramakrishnan, R. [1990] "The Magic of Duplicates and Aggregates", en VLDB [1990].
- Muralikrishna, M. [1992] "Improved Unnesting Algorithms for Join and Aggregate SQL Queries", en VLDB [1992].
- Mylopoulos, J., Bernstein, P. y Wong, H. [1980] "A Language Facility for Designing Database-Intensive Applications", TODS, 5:2, junio de 1980.
- Naffah, N. (editor) [1982] *Office Information Systems*, North-Holland, 1983.
- Naish, L. y Thom, J. [1983] "The MU-PROLOG Deductive Database", Technical Report 83/10, Department of Computer Science, University of Melbourne, 1983.
- Navathe, S. [1980] "An Intuitive View to Normalize Network-Structured Data", en VLDB [1980].
- Navathe, S. [1985] "Important Issues in Database Design Methodologies and Tools", en Albano et al. [1985].
- Navathe, S. y Ahmed, R. [1988] "A Temporal Relational Model and Query Language", *Information Sciences* (en prensa).
- Navathe, S., Ceri, S., Wiederhold, G. y Dou, J. [1984] "Vertical Partitioning Algorithms for Database Design", TODS, 9:4, diciembre de 1984.
- Navathe, S. y Cornelio, A. [1990] "Modeling Engineering Data by Complex Structural Objects and Complex Functional Objects", *Proceedings of the International Conference on Extending Data Base Technology*, Venecia, Italia, marzo de 1990, Springer-Verlag Notes in Computer Science, núm. 416, 1990.
- Navathe, S., Elmasri, R. y Larson, J. [1986] "Integrating User Views in Database Design", *IEEE Computer*, 19:1, enero de 1986.
- Navathe, S. y Gadgil, S. [1982] "A Methodology for View Integration in Logical Database Design", en VLDB [1982].
- Navathe, S. y Kerschberg, L. [1986] "Role of Data Dictionaries in Database Design", *Information and Management*, 10:1, enero de 1986.
- Navathe, S. y Pillalamari, M. [1988] "Toward Making the ER Approach Object-Oriented", en ER Conference [1988].
- Navathe, S., Sashidhar, T. y Elmasri, R. [1984] "Relationship Merging in Schema Integration", en VLDB [1984].
- Navathe, S. y Schkolnick, M. [1978] "View Representation in Logical Database Design", en SIGMOD [1978].

- Negri, M., Pelagatti, S. y Sbatella, L. [1991] "Formal Semantics of SQL Queries", *TODS*, 16:3, septiembre de 1991.
- Ng, P. [1981] "Further Analysis of the Entity-Relationship Approach to Database Design", *TSE*, 7:1, enero de 1981.
- Nicolas, J. [1978] "Mutual Dependencies and Some Results on Undecomposable Relations", en *Vldb* [1978].
- Nievergelt, J. [1974] "Binary Search Trees and File Organization", *ACM Computing Surveys*, 6:3, septiembre de 1974.
- Nijssen, G. (editor) [1976] *Modelling in Data Base Management Systems*, North-Holland, 1976.
- Nijssen, G. (editor) [1977] *Architecture and Models in Data Base Management Systems*, North-Holland, 1977.
- Obermark, R. [1982] "Distributed Deadlock Detection Algorithms", *TODS*, 7:2, junio de 1982.
- Ohswaga, S. [1982] "Knowledge Based Systems as a New Interactive Computer System of the Next Generation", en *Computer Science and Technologies*, North-Holland, 1982.
- Olle, T. [1978] *The CODASYL Approach to Data Base Management*, Wiley, 1978.
- Olle, T., Sol, H. y Verrijn-Stuart, A. (editores) [1982] *Information System Design Methodology*, North-Holland, 1982.
- Omicinski, E. y Scheuermann, P. [1990] "A Parallel Algorithm for Record Clustering", *TODS*, 15:4, diciembre de 1990.
- Onuegue, E., Rahimi, S. y Hevner, A. [1983] "Local Query Translation and Optimization in a Distributed System", *NCC, AFPS*, 52, 1983.
- Orenstein, J. [1986] "Spatial Query Processing in an Object-Oriented Database System", en *SIGMOD* [1986].
- Osborn, S. [1979] "Towards a Universal Relation Interface", en *Vldb* [1979].
- Osborn, S. [1989] "The Role of Polymorphism in Schema Evolution in an Object-Oriented Database", *TKDE*, 1:3, septiembre de 1989.
- Osoyoglu, G., Osoyoglu, Z. y Matos, V. [1987] "Extending Relational Algebra and Relational Calculus with Set Valued Attributes and Aggregate Functions", *TODS*, 12:4, diciembre de 1987.
- Osoyoglu, Z. y Yuan, L. [1987] "A New Normal Form for Nested Relations", *TODS*, 12:1, marzo de 1987.
- Ozsu, T. y Valduriez, P. [1991] *Principles of Distributed Database Systems*, Prentice-Hall, 1991.
- Palermo, F. [1974] "A Database Search Problem", en *Tou* [1974].
- Papadimitriou, C. [1979] "The Serializability of Concurrent Database Updates", *JACM*, 26:4, octubre de 1979.
- Papadimitriou, C. [1986] *The Theory of Database Concurrency Control*, Computer Science Press, 1986.
- Papadimitriou, C. y Kanellakis, P. [1979] "On Concurrency Control by Multiple Versions", *TODS*, 9:1, marzo de 1979.
- Papazoglou, M. y Valder, W. [1989] *Relational Database Management: A Systems Programming Approach*, Prentice-Hall, 1989.
- Paredaens, J. y Van Gucht, D. [1992] "Converting Nested Algebra Expressions into Flat Algebra Expressions", *TODS*, 17:1, marzo de 1992.
- Parent, C. y Spaccapietra, S. [1985] "An Algebra for a General Entity-Relationship Model", *TSE*, 11:7, julio de 1985.
- Paris, J. [1986] "Voting with Witnesses: A Consistency Scheme for Replicated Files", en *ICDE* [1986].
- Paul, H. et al. [1987] "Architecture and Implementation of the Darmstadt Database Kernel System", en *SIGMOD* [1987].
- PDES [1991] "A High-Lead Architecture for Implementing a PDES/STEP Data Sharing Environment", Publication Number PT 1017.03.00, PDES Inc., mayo de 1991.

- Pernici, B. et al. [1989] "C-TODS: an Automatic Tool for Office System Conceptual Design", *TODS*, 14:4, octubre de 1989.
- Perrizo, W., Rajkumar, J. y Ram, P. [1991] "HYDRO: A Heterogeneous Distributed Database System", en *SIGMOD* [1991].
- Phipps, G., Derr, M. y Ross, K. [1991] "Glue-NAIL: A Deductive Database System", en *SIGMOD* [1991].
- Platesky-Shapiro, G. y Frauley, W. (editores) [1991] *Knowledge Discovery in Databases*, AAAI Press/The MIT Press, 1991.
- Pillalamari, M., Navathe, S. y Papachristidis, A. [1988] "Understanding the Power of Semantic Data Models", artículo de trabajo, Database Systems R & D Center, University of Florida.
- Prague, C. y Hammit, J. [1985] *Programming with dBase III*, Tab Books, 1985.
- Rabitti, F., Bertino, E., Kim, W. y Woelk, D. [1991] "A Model of Authorization for Next-Generation Database Systems", *TODS*, 16:1, marzo de 1991.
- Ramakrishnan, R., Srivastava, D. y Sudarshan, S. [1992] "CORAL: Control, Relations and Logic", en *Vldb* [1992].
- Ramakrishnan, R., Srivastava, D. y Sudarshan, S. [1992a] "{CORAL}: {C}ontrol, {R}elations and {L}ogic", en *Vldb* [1992].
- Ramakrishnan, R., Srivastava, D., Sudarshan, S. y Sheshadri, P. [1993] "Implementation of the {CORAL} deductive database system", en *SIGMOD* [1993].
- Ramakrishnan, R. y Ullman, J. [1994] "Survey of Research in Deductive Database Systems", en *Journal of Logic Programming*, por publicarse en 1994.
- Ramamoorthy, C. y Wah, B. [1979] "The Placement of Relations on a Distributed Relational Database", *Proceedings of the First International Conference on Distributed Computing Systems*, IEEE CS, 1979.
- Reed, D. [1983] "Implementing Atomic Actions on Decentralized Data", *TODS*, 1:1, febrero de 1983.
- Reisner, P. [1977] "Use of Psychological Experimentation as an Aid to Development of a Query Language", *TSE*, 3:3, mayo de 1977.
- Reisner, P. [1981] "Human Factors Studies of Database Query Languages: A Survey and Assessment", *ACM Computing Surveys*, 13:1, marzo de 1981.
- Reiter, R. [1984] "Towards a Logical Reconstruction of Relational Database Theory", capítulo 8 de Brodie et al. [1984].
- Ries, D. [1979] "The Effect of Concurrency Control on the Performance of a Distributed Database Management System", *Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks*, IEEE CS, febrero de 1979.
- Ries, D. y Stonebraker, M. [1977] "Effects of Locking Granularity in a Database Management System", *TODS*, 2:3, septiembre de 1977.
- Rissanen, J. [1977] "Independent Components of Relations", *TODS*, 2:4, diciembre de 1977.
- Rivest, R., Shamir, A. y Adelman, L. [1978] "On Digital Signatures and Public Key Cryptosystems", *CACM*, 21:2, febrero de 1978.
- Ross, S. [1986] *Understanding and Using dBase III*, West Publishing, 1986.
- Roth, M. y Korth, H. [1987] "The Design of Non-1NF Relational Databases into Nested Normal Form", en *SIGMOD* [1987].
- Rothnie, J. [1975] "Evaluating Inter-Entry Retrieval Expressions in a Relational Data Base Management System", *NCC, AFPS*, 44, 1975.
- Rothnie, J. et al. [1980] "Introduction to a System for Distributed Databases (SDD-1)", *TODS*, 5:1, marzo de 1980.
- RTI [1983] *INGRES Reference Manual*, Relational Technology Inc., 1983.
- Roussopoulos, N. [1991] "An Incremental Access Method for View-Cache: Concept, Algorithms, and Cost Analysis", *TODS*, 16:3, septiembre de 1991.

- Rozen, S. y Shasha, D. [1991] "A Framework for Automating Physical Database Design", en *Vldb* [1991].
- Rudensteiner, E. [1992] "Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases", en *Vldb* [1992].
- Rusinkiewicz, M. et al. [1988] "OMNIBASE—A Loosely-Coupled Design and Implementation of a Multidatabase System", *IEEE Distributed Processing Newsletter*, 10:2, noviembre de 1988.
- Rustin, R. (editor) [1972] *Data Base Systems*, Prentice-Hall, 1972.
- Rustin, R. (editor) [1974] *Proceedings of the ACM SIGMOD Debate on Data Models: Data Structure Set Versus Relational*, 1974.
- Sacca, D. y Zaniolo, C. [1987] "Implementation of Recursive Queries for a Data Language Based on Pure Horn Clauses", *Proceedings of the Fourth International Conference on Logic Programming*, MIT Press, 1987.
- Sadri, F. y Ullman, J. [1982] "Template Dependencies: A Large Class of Dependencies in Relational Databases and Its Complete Axiomatization", *JACM*, 29:2, abril de 1982.
- Sagiv, Y. y Yamakakis, M. [1981] "Equivalence among Relational Expressions with the Union and Difference Operators", *JACM*, 27:4, noviembre de 1981.
- Sakai, H. [1980] "Entry-Relationship Approach to Conceptual Schema Design", en *SIGMOD* [1980].
- Salzberg, B. [1988] *File Structures: An Analytic Approach*, Prentice-Hall, 1988.
- Salzberg, B. et al. [1990] "FastSort: A Distributed Single-Input Single-Output External Sort", en *SIGMOD* [1990].
- Salton, G. y Buckley, C. [1991] "Global Text Matching for Information Retrieval", en *Science*, 253, agosto de 1991.
- Samet, H. [1990] *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.
- Samet, H. [1990a] *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, 1990.
- Sammur, C. y Sammut, R. [1983] "The Implementation of UNSW-PROLOG", *The Australian Computer Journal*, mayo de 1983.
- Schenk, H. [1974] "Implementation Aspects of the DBTG Proposal", *Proceedings of the IFIP Working Conference on Database Management Systems*, 1974.
- Scheuermann, P. (editor) [1982] *Improving Database Usability and Responsiveness*, Academic Press, 1982.
- Scheuermann, P., Schiffner, G. y Weber, H. [1979] "Abstraction Capabilities and Invariant Properties Modeling within the Entry-Relationship Approach", en *ER Conference* [1979].
- Schkolnick, M. [1978] "A Survey of Physical Database Design Methodology and Techniques", en *Vldb* [1978].
- Schlageter, G. [1981] "Optimistic Methods for Concurrency Control in Distributed Database Systems", en *Vldb* [1981].
- Schlimmer, J., Mitchell, T. y McDermott, J. [1991] "Justification Based Refinement of Expert Knowledge", en *Platesky-Shapiro y Frawley* [1991].
- Schmidt, J. y Swenson, J. [1975] "On the Semantics of the Relational Model", en *SIGMOD* [1975].
- Schwarz, P. et al. [1986] "Extensibility in the Starburst Database System", en *Dittrich y Dayal* [1986].
- Sciore, E. [1982] "A Complete Axiomatization for Full Join Dependencies", *JACM*, 29:2, abril de 1982.
- Selinger, P. et al. [1979] "Access Path Selection in a Relational Database Management System", en *SIGMOD* [1979].
- Senko, M. [1975] "Specification of Stored Data Structures and Desired Output in DIAM II with FORAL", en *Vldb* [1975].
- Senko, M. [1980] "A Query Maintenance Language for the Data Independent Accessing Model II", *Information Systems*, 5:4, 1980.

- Shekita, E. y Carey, M. [1989] "Performance Enhancement Through Replication in an Object-Oriented DBMS", en *SIGMOD* [1989].
- Shenoy, S. y Ososyoglu, Z. [1989] "Design and Implementation of a Semantic Query Optimizer", *TKDE*, 1:3, septiembre de 1989.
- Sheth, A., Larson, J., Cornelio, A. y Navathe, S. [1988] "A Tool for Integrating Conceptual Schemas and User Views", en *ICDE* [1988].
- Shim, K., Sellis, T. y Nau, D. [1993] "Improvements on a Heuristic Algorithm for Multiple-Query Optimization", en *Data and Knowledge Engineering*, North-Holland, por publicarse en 1993.
- Shipman, D. [1981] "The Functional Data Model and the Data Language DAPLEX", *TODS*, 6:1, marzo de 1981.
- Shneiderman, B. (editor) [1978] *Databases: Improving Usability and Responsiveness*, Academic Press, 1978.
- Sibley, E. [1976] "The Development of Database Technology", *ACM Computing Surveys*, 8:1, marzo de 1976.
- Sibley, E. y Kerschberg, L. [1977] "Data Architecture and Data Model Considerations", *NCC, AFPS*, 46, 1977.
- Siegel, M. y Madnick, S. [1991] "A Metadata Approach to Resolving Semantic Conflicts", en *Vldb* [1991].
- Siegel, M., Sciore, E. y Salverter, S. [1992] "A Method for Automatic Rule Derivation to Support Semantic Query Optimization", *TODS*, 17:4, diciembre de 1992.
- SIGMOD* [1974] *Proceedings of the ACM SIGMOD-SIGFIDET Conference on Data Description, Access and Control*, Rustin, R. (editor), mayo de 1974.
- SIGMOD* [1975] *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data*, King, E. (editor), San Jose, California, mayo de 1975.
- SIGMOD* [1976] *Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data*, Rothnie, J. (editor), Washington, D.C., junio de 1976.
- SIGMOD* [1977] *Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data*, Smith, D. (editor), Toronto, Canadá, agosto de 1977.
- SIGMOD* [1978] *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*, Lowenthal, E. y Dale, N. (editores), Austin, Texas, mayo/junio de 1978.
- SIGMOD* [1979] *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, Bernstein, P. (editor), Boston, Massachusetts, mayo/junio de 1979.
- SIGMOD* [1980] *Proceedings of the 1980 ACM SIGMOD International Conference on Management of Data*, Chen, P. y Sprowls, R. (editores), Santa Monica, California, mayo de 1980.
- SIGMOD* [1981] *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, Lien, Y. (editor), Ann Arbor, Michigan, abril/mayo de 1981.
- SIGMOD* [1982] *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, Schkolnick, M. (editor), Orlando, Florida, junio de 1982.
- SIGMOD* [1983] *Proceedings of the 1983 ACM SIGMOD International Conference on Management of Data*, DeWitt, D. y Gardarin, G. (editores), San Jose, California, mayo de 1983.
- SIGMOD* [1984] *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, Yormark, B. (editor), Boston, Massachusetts, junio de 1984.
- SIGMOD* [1985] *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, Navathe, S. (editor), Austin, Texas, mayo de 1985.
- SIGMOD* [1986] *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, Zaniolo, C. (editor), Washington, D.C., mayo de 1986.
- SIGMOD* [1987] *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, Dayal, U. y Traiger, I. (editores), San Francisco, California, mayo de 1987.

- SIGMOD [1988] *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, Boral, H. y Larson, P. (editores), Chicago, Illinois, junio de 1988.
- SIGMOD [1989] *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Clifford, J., Lindsay, B. y Maier, D. (editores), Portland, Oregon, junio de 1989.
- SIGMOD [1990] *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, Garcia-Molina, H. y Jagadish, H. (editores), Atlantic City, Nueva Jersey, junio de 1990.
- SIGMOD [1991] *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Clifford, J. y King, R. (editores), Denver, Colorado, junio de 1991.
- SIGMOD [1992] *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, Stonebraker, M. (editor), San Diego, California, junio de 1992.
- SIGMOD [1993] *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Buneman, P. y Jajodia, S. (editores), Washington, D.C., junio de 1993.
- Silberschatz, A., Stonebraker, M. y Ullman, J. [1990] "Database Systems: Achievements and Opportunities", en *ACM SIGMOD Record*, 19-4, diciembre de 1990.
- Simpson, A. [1989] *dBase Programmers Reference Guide*, SYBEX Inc., 1989.
- Sirbu, M., Schoichet, S., Kunin, J. y Hammer, M. [1981] "OAM: An Office Analysis Methodology", Massachusetts Institute of Technology Office Automation Group, Memo OAM-016, 1981.
- Skeen, D. [1981] "Non-Blocking Commit Protocols", en *SIGMOD* [1981].
- Smith, G. [1990] "The Semantic Data Model for Security: Representing the Security Semantics of an Application", en *ICDE* [1990].
- Smith, J. y Chang, P. [1975] "Optimizing the Performance of a Relational Algebra Interface", *CACM*, 18-10, octubre de 1975.
- Smith, J. y Smith, D. [1977] "Database Abstractions: Aggregation and Generalization", *TODS*, 2-2, junio de 1977.
- Smith, J. et al. [1981] "MULTIBASE: Integrating Distributed Heterogeneous Database Systems", *NCC, AFIPS*, 50, 1981.
- Smith, K. y Winslett, M. [1992] "Entity Modeling in the MLS Relational Model", en *Vldb* [1992].
- Smith, P. y Barnes, G. [1987] *Files & Databases: An Introduction*, Addison-Wesley, 1987.
- Snodgrass, R. y Ahn, I. [1985] "A Taxonomy of Time in Databases", en *SIGMOD* [1985].
- Spooner, D., Michael, A. y Donald, B. [1986] "Modeling CAD data with Data Abstraction and Object-Oriented Technique", en *ICDE* [1986].
- Srinivasan, V. y Carey, M. [1991] "Performance of B-Tree Concurrency Control Algorithms", en *SIGMOD* [1991].
- Srivastava, D., Ramakrishnan, R., Sudarshan, S. y Sheshadri, P. [1993] "Coral++: Adding Object-orientation to a Logic Database Language", en *Vldb* [1993].
- Stachour, P. y Thuraissingham, B. [1990] "The Design and Implementation of INGRES", *TKDE*, 2-2, junio de 1990.
- Stonebraker, M. [1975] "Implementation of Integrity Constraints and Views by Query Modification", en *SIGMOD* [1975].
- Stonebraker, M. [1993] "The Miro DBMS", en *SIGMOD* [1993].
- Stonebraker, M. (editor) [1986] *The INGRES Papers*, Addison-Wesley, 1986.
- Stonebraker, M. (editor) [1988] *Readings in Database Systems*, Morgan Kaufmann, 1988.
- Stonebraker, M., Hanson, E. y Hong, C. [1987] "The Design of the POSTGRES Rules System", en *ICDE* [1987].
- Stonebraker, M. y Rowe, L. [1986] "The Design of POSTGRES", en *SIGMOD* [1986].
- Stonebraker, M., Rubenstein, B. y Guttman, A. [1983] "Application of Abstract Data Types and Exact Indices to CAD Databases", en *SIGMOD* [1983].
- Stonebraker, M. y Wong, E. [1974] "Access Control in a Relational Database Management System by Query Modification", *Proceedings of the ACM Annual Conference*, 1974.

- Stonebraker, M., Wong, E., Kreps, P. y Held, G. [1976] "The Design and Implementation of INGRES", *TODS*, 1-3, septiembre de 1976.
- Su, S. [1985] "A Semantic Association Model for Corporate and Scientific-Statistical Databases", *Information Science*, 29, 1985.
- Su, S. [1988] *Database Computers*, McGraw-Hill, 1988.
- Su, S., Krishnamurthy, V. y Lam, H. [1988] "An Object-Oriented Semantic Association Model (OSAM*)", en *AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications*, American Institute of Industrial Engineers, 1988.
- Sybase [1990] *Transact - SQL User's Guide*, Sybase, Inc., 1990.
- Tanenbaum, A. [1981] *Computer Networks*, Prentice-Hall, 1981.
- Tansel, A. et al. (editores) [1993] *Temporal Databases: Theory, Design and Implementation*, Benjamin/Cummings, 1993.
- Taylor, R. y Frank, R. [1976] "CODASYL Data Base Management Systems", *ACM Computing Surveys*, 8-1, marzo de 1976.
- Teorey, T. [1990] *Database Modeling and Design: The Entity-Relationship Approach*, Morgan Kaufmann, 1990.
- Teorey, T. y Fry, J. [1982] *Design of Database Structures*, Prentice-Hall, 1982.
- Teorey, T., Yang, D. y Fry, J. [1986] "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", *ACM Computing Surveys*, 18-2, junio de 1986.
- Thomas, J. y Gould, J. [1975] "A Psychological Study of Query By Example", *NCC, AFIPS*, 44, 1975.
- Thomas, R. [1979] "A Majority Consensus Approach to Concurrency Control for Multiple Copy Data Bases", *TODS*, 4-2, junio de 1979.
- Thomsonian, A. [1991] "Performance Limits of Two-Phase Locking", en *ICDE* [1991].
- Todd, S. [1976] "The Peterlee Relational Test Vehicle—A System Overview", *IBM Systems Journal*, 15-4, diciembre de 1976.
- Tou, J. (editor) [1984] *Information Systems COINS-IV*, Plenum Press, 1984.
- Tsangaris, M. y Naughton, J. [1992] "On the Performance of Object Clustering Techniques", en *SIGMOD* [1992].
- Tsichritzis, D. [1982] "Forms Management", *CACM*, 25-7, julio de 1982.
- Tsichritzis, D. y Klug, A. (editores) [1978] *The ANSI/X3/SPARC DBMS Framework*, AFIPS Press, 1978.
- Tsichritzis, D. y Lochovsky, F. [1976] "Hierarchical Data-base Management: A Survey", *ACM Computing Surveys*, 8-1, marzo de 1976.
- Tsichritzis, D. y Lochovsky, F. [1982] *Data Models*, Prentice-Hall, 1982.
- Tsotras, V. y Gopinath, B. [1992] "Optimal Versioning of Object Classes", en *ICDE* [1992].
- Ullman, J. [1982] *Principles of Database Systems*, 2da. ed., Computer Science Press, 1982.
- Ullman, J. [1985] "Implementation of Logical Query Languages for Databases", *TODS*, 10-3, septiembre de 1985.
- Ullman, J. [1988] *Principles of Database and Knowledge-Base Systems*, vol. 1, Computer Science Press, 1988.
- Ullman, J. [1989] *Principles of Database and Knowledge-Base Systems*, vol. 2, Computer Science Press, 1989.
- U.S. Congress [1988] "Office of Technology Report, Appendix D: Databases, Repositories and Informatics", en *Mapping our Genes: Genome Projects: How Big, How Fast?*, The Johns Hopkins University Press, 1988.
- Valduriez, P. y Gardarin, G. [1989] *Analysis and Comparison of Relational Database Systems*, Addison-Wesley, 1989.
- Vassiliou, Y. [1980] "Functional Dependencies and Incomplete Information", en *Vldb* [1980].
- Verheijen, G. y Van Bekkum, J. [1982] "NIAM: An Information Analysis Method", en *Olle et al.* [1982].

- Verhofstadt, J. [1978] "Recovery Techniques for Database Systems", *ACM Computing Surveys*, 10:2, junio de 1978.
- Vielle, L. [1986] "Recursive Axioms in Deductive Databases: The Query-subquery Approach", en EDS [1986].
- Vielle, L. [1987] "Database Complete Proof production based on SLD-resolution", en *Proceedings of the 4th International Conference on Logic Programming*, 1987.
- Vielle, L. [1988] "From QSQ Towards QoSQ: Global Optimization of Recursive Queries", en EDS [1988].
- Vin, H., Zellweger, P., Swinehart, D. y Venkat Rangan, P. [1991] "Multimedia Conferencing in the Etherphone Environment", *IEEE Computer*, número especial sobre sistemas de información de multimedia, 24:10, octubre de 1991.
- VLD8 [1975] *Proceedings of the First International Conference on Very Large Data Bases*, Kerr, D. (editor), Framingham, Massachusetts, septiembre de 1975.
- VLD8 [1976] *Systems For Large Databases*, Lockemann, P. y Neuhold, E. (editores) (*Proceedings of the Second International Conference on Very Large Data Bases*, Bruselas, Bélgica, julio de 1976), North-Holland, 1977.
- VLD8 [1977] *Proceedings of the Third International Conference on Very Large Data Bases*, Merten, A. (editor), Tokio, Japón, octubre de 1977.
- VLD8 [1978] *Proceedings of the Fourth International Conference on Very Large Data Bases*, Bubenko, J. y Yao, S. (editores), Berlín Occidental, Alemania, septiembre de 1978.
- VLD8 [1979] *Proceedings of the Fifth International Conference on Very Large Data Bases*, Furtado, A. y Morgan, H. (editores), Rio de Janeiro, Brasil, octubre de 1979.
- VLD8 [1980] *Proceedings of the Sixth International Conference on Very Large Data Bases*, Lochovsky, F. y Taylor, R. (editores), Montreal, Canadá, octubre de 1980.
- VLD8 [1981] *Proceedings of the Seventh International Conference on Very Large Data Bases*, Zaniolo, C. y Delobel, C. (editores), Cannes, Francia, septiembre de 1981.
- VLD8 [1982] *Proceedings of the Eighth International Conference on Very Large Data Bases*, McLeod, D. y Villaseñor, Y. (editores), Ciudad de México, México, septiembre de 1982.
- VLD8 [1983] *Proceedings of the Ninth International Conference on Very Large Data Bases*, Schkolnick, M. y Thanos, C. (editores), Florencia, Italia, octubre/noviembre de 1983.
- VLD8 [1984] *Proceedings of the Tenth International Conference on Very Large Data Bases*, Dayal, U., Schlageter, G. y Seng, L. (editores), Singapur, agosto de 1984.
- VLD8 [1985] *Proceedings of the Eleventh International Conference on Very Large Data Bases*, Pirrotte, A. y Vassiliou, Y. (editores), Estocolmo, Suecia, agosto de 1985.
- VLD8 [1986] *Proceedings of the Twelfth International Conference on Very Large Data Bases*, Chu, W., Gardarin, G. y Ohsuga, S. (editores), Kioto, Japón, agosto de 1986.
- VLD8 [1987] *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, Stocker, E., Kent, W. y Hammersley, P. (editores), Brighton, Inglaterra, septiembre de 1987.
- VLD8 [1988] *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, Bancilhon, F. y De Witt, D. (editores), Los Angeles, California, agosto/septiembre de 1988.
- VLD8 [1989] *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, Apers, P. y Wiederhold, G. (editores), Amsterdam, Países Bajos, agosto de 1989.
- VLD8 [1990] *Proceedings of the Sixteenth International Conference on Very Large Data Bases*, McLeod, D., Sacks-Davis, R. y Schek, H. (editores), Brisbane, Australia, agosto de 1990.
- VLD8 [1991] *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, Lohman, G., Semadas, A. y Camps, R. (editores), Barcelona, Cataluña, España, septiembre de 1991.
- VLD8 [1992] *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, Yuan, L. (editor), Vancouver, Columbia Británica, Canadá, agosto de 1992.

- VLD8 [1993] *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, Dublin, Irlanda, agosto de 1993.
- Vorhaus, A. y Mills, R. [1967] "The Time-Shared Data Management System: A New Approach to Data Management", System Development Corporation, Report SP-2634, 1967.
- Walton, C., Dale, A. y Jenevein, R. [1991] "A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins", en VLD8 [1991].
- Wang, K. [1990] "Polynomial Time Designs Toward Both BCNF and Efficient Data Manipulation", en SIGMOD [1990].
- Wang, Y. y Madnick, S. [1989] "The Inter-Database Instance Identity Problem in Integrating Autonomous Systems", en ICDE [1989].
- Wang, Y. y Rowe, L. [1991] "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture", en SIGMOD [1991].
- Warren, D. [1992] "Memoing for Logic Programs", *CACM*, 35:3, ACM, marzo de 1992.
- Weddell, G. [1992] "Reasoning About Functional Dependencies Generalized for Semantic Data Models", *TODS*, 17:1, marzo de 1992.
- Weikum, G. [1991] "Principles and Realization Strategies of Multilevel Transaction Management", *TODS*, 16:1, marzo de 1991.
- Weldon, J. [1981] *Data Base Administration*, Plenum Press, 1981.
- Whang, K. [1985] "Query Optimization in Office By Example", IBM Research Report RC 11571, diciembre de 1985.
- Whang, K., Malhotra, A., Sockut, G. y Burns, L. [1990] "Supporting Universal Quantification in a Two-Dimensional Database Query Language", en ICDE [1990].
- Whang, K. y Navathe, S. [1987] "An Extended Disjunctive Normal Form Approach for Processing Recursive Logic Queries in Loosely Coupled Environments", en VLD8 [1987].
- Whang, K. y Navathe, S. [1992] "Integrating Expert Systems with Database Management Systems—an Extended Disjunctive Normal Form Approach", en *Information Sciences*, 64, marzo de 1992.
- Whang, K., Wiederhold, G. y Sagalowicz, D. [1982] "Physical Design of Network Model Databases Using the Property of Separability", en VLD8 [1982].
- Widom, J. y Finkelstein, S. [1990] "Set oriented production rules in relational database systems", en SIGMOD [1990].
- Wiederhold, G. [1983] *Database Design*, 2da. ed., McGraw-Hill, 1983.
- Wiederhold, G. [1984] "Knowledge and Database Management", *IEEE Software*, enero de 1984.
- Wiederhold, G., Beetem, A. y Short, G. [1982] "A Database Approach to Communication in VLSI Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1:2, abril de 1982.
- Wiederhold, G. y Elmasti, R. [1979] "The Structural Model for Database Design", en *ER Conference* [1979].
- Wilkinson, K., Lyngbaek, P. y Hasan, W. [1990] "The IRIS Architecture and Implementation", *TKDE*, 2:1, marzo de 1990.
- Willshire, M. [1991] "How Spacey Can They Get? Space Overhead for Storage and Indexing with Object-Oriented Databases", en ICDE [1991].
- Wilson, B. y Navathe, S. [1986] "An Analytical Framework for Limited Redesign of Distributed Databases", *Proceedings of the Sixth Advanced Database Symposium*, Tokio, Japón, agosto de 1986.
- Wirkowski, G. y Kull, D. [1992] *DB2 Design and Development Guide*, 3ra. ed., Addison-Wesley, 1992.
- Wirth, N. [1972] *Algorithms + Data Structures = Programs*, Prentice-Hall, 1972.
- Woelk, D., Luther, W. y Kim, W. [1987] "Multimedia Applications and Database Requirements", *Proceedings of IEEE Office Automation Symposium*, abril de 1987.

- Wolfson, O. y Milo, A. [1991] "The Multicast Policy and Its Relationship to Replicated Data Placement", *TODS*, 16:1, marzo de 1991.
- Wong, E. [1983] "Dynamic Rematerialization-Processing Distributed Queries Using Redundant Data", *TSE*, 9:3, mayo, 1983.
- Wong, E. y Youssefi, K. [1976] "Decomposition: A Strategy for Query Processing", *TODS*, 1:3, septiembre de 1976.
- Wong, H. [1984] "Micro and Macro Statistical/Scientific Database Management", en *ICDE* [1984].
- Wu, X. e Ichikawa, T. [1992] "KDA: A Knowledge-based Database Assistant with a Query Guiding Facility", en *TKDE*, 4:5, octubre de 1992.
- Yao, S. [1979] "Optimization of Query Evaluation Algorithms", *TODS*, 4:2, junio de 1979.
- Yao, S. (editor) [1985] *Principles of Database Design*, vol. 1: *Logical Organizations*, Prentice-Hall, 1985.
- Youssefi, K. y Wong, E. [1979] "Query Processing in a Relational Database Management System", en *Vldb* [1979].
- Zadeh, L. [1983] "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems", *Fuzzy Sets and Systems*, 11, North-Holland, 1983.
- Zaniolo, C. [1976] "Analysis and Design of Relational Schemata for Database Systems", tesis de doctorado, University of California, Los Angeles, 1976.
- Zaniolo, C. [1988] "Design and Implementation of a Logic Based Language for Data Intensive Applications", *MCC Technical Report #ACA-ST-199-88*, junio de 1988.
- Zaniolo, C. et al. [1986] "Object-Oriented Database Systems and Knowledge Systems", en *EDS* [1986].
- Zicari, R. [1991] "A Framework for Schema Updates in an Object-Oriented Database System", en *ICDE* [1991].
- Zloof, M. [1975] "Query By Example", *NCC, AFPS*, 44, 1975.
- Zloof, M. [1982] "Office By Example: A Business Language That Unifies Data, Word Processing, and Electronic Mail", *IBM Systems Journal*, 21:3, 1982.
- Zobel, J., Moffat, A. y Sacks-Davis, R. [1992] "An Efficient Indexing Technique for Full-Text Database Systems", en *Vldb* [1992].
- Zook, W. et al. [1977] *INGRES Reference Manual*, Department of EECS, University of California at Berkeley, 1977.
- Zvieli, A. [1986] "A Fuzzy Relational Calculus", en *EDS* [1986].

Índice de materias

- abanico 115
- ABORTAR 538
- abrir (orden de acceso a archivos) 82
- abstracción de los datos 7, 635-640, 668-676
- acceso, camino de 23
- acceso, control de 599, 602
- discrecional 602-607
- obligatorio 607-610
- acceso, estructura de 104-134
- acceso, método de 82
- ACCESS 263, 765
- ACID, propiedades véase transacción
- actual de
- jerarquía 368
- tipo de conjuntos 316
- tipo de registros 316
- unidad de ejecución 316
- actualidad, indicadores de, 316-318
- actualización diferida, recuperación en 581, 586-589
- en sistema monousuario 586-587
- en sistema multiusuario 587-589
- protocolo de 585-586
- actualización en el lugar 582
- actualización inmediata, recuperación en 581, 590-591
- monousuario 590-591
- multiusuario 592
- actualización perdida, problema de 535
- actualización temporal, problema de 535-536
- actualizar 5
- (véase también vistas, actualización de)
- ADA, lenguaje 225
- agregación 638-640
- agrupación 167-169
- en QBE 256
- en QUEL 248-249
- en SQL 210-213
- agrupación, atributos de 167-169, 210
- agrupamiento
- campo de 109
- índice de 109, 225
- aislamiento, nivel de 542
- alcance (de archivo) 80
- aleatorización, función de 88
- álgebra relacional 155-167, 180
- conjunto completo del 166
- ejemplos de consultas en 172-174
- expresión en 159

- operaciones adicionales del 167-172 (*véase también* REUNIÓN EXTERNA; UNIÓN EXTERNA; FUNCIÓN; DIVISIÓN)
- operaciones del *véase* PRODUCTO CARTESIANO; DIFERENCIA; INTERSECCIÓN; REUNIÓN; PROYECCIÓN; SELECCIÓN; UNIÓN
- optimización en *véase* consulta, optimización de árbol de reglas del 744-745
- transformación en *véase* transformación
- ALGOL 768
- ALL (QUEL) 246-247
- ALL (SQL) 201-203
- ALL (QBE) 257
- almacenamiento
 - asignación de 80
 - comparación de estructuras de 815-816
 - estructuras de *véase* archivo, índice jerarquía de 69
 - lenguaje de definición de (SDL) 28 (*véase también* almacenamiento primario; almacenamiento secundario)
 - almacenamiento en línea 65
 - almacenamiento fuera de línea 70
 - almacenamiento intermedio 75-76, 582
 - doble 76
 - en DB2 267
 - almacenamiento persistente 14
 - almacenamiento primario 69
 - almacenamiento secundario 69
 - ALTER TABLE (SQL) 194-195
 - ambigüedad de consultas en SQL
 - en consultas anidadas 203
 - en nombres de atributos 198
 - American National Standards Institute *véase* ANSI
 - análisis de datos *véase* requerimientos, análisis de
 - análizador sintáctico 495
 - ANIDAR, operación 656
 - anillo, estructura de datos de 297
 - anomalías *véase* modificación, anomalías de; eliminación, anomalías de; inserción, anomalías de
 - anotaciones 757
 - ANSI 26, 189, 290
 - ANSI/SPARC, arquitectura 26
 - ANY (QUEL) 248
 - ANY (SQL) 202
 - aplicación *véase* base de datos, aplicación de aplicaciones, creación de 16
 - aplicaciones, procesamiento de 266-268
 - APPEND (QUEL) 249
 - apuntador(es)
 - arreglo de 298
 - campo 297
 - (*véase también* registro, apuntador a)
 - apuntador de datos 121, 124-125
 - apuntador al anterior 297-298
 - apuntador al hijo 119
 - apuntador al siguiente 297-298, 301
 - árbol, apuntador a 118, 123
 - árbol, estructura de datos de 118, 351 (*véase también* multinivel índice; esquema en el modelo de datos jerárquico; ejemplar jerárquico)
 - árbol, nodo de 118, 346
 - árbol B 120-123
 - árbol B* 130
 - árbol B+ 123-131
 - búsqueda en 126
 - estructura de 124
 - inserción en 128
 - árbol cuadrático 793
 - árbol de ocurrencias 352-355
 - árbol equilibrado 120
 - árbol R 793
 - archivado 74
 - archivo(s) 5, 77
 - asignación de 80
 - cabecera (descriptor) de 80
 - examinadores de 498
 - operaciones de 80-82
 - organización de 82-98
 - procesamiento de 5-8
 - reorganización de 33, 83, 86-87
 - vs. relación 142-144
 - (*véase también* acceso, método de; acceso, camino de; acceso, estructura de)
 - archivo de transacciones 86
 - archivo invertido 133
 - archivo maestro 86
 - archivo mixto 77, 79, 98
 - archivo no ordenado 83-84
 - archivo ordenado 83-87
 - archivo principal de datos 87
 - archivo relativo 84
 - archivo secuencial 83, 84
 - archivo secuencial indizado 133
 - área (del disco) 98
 - AREA (modelo de red) 336
 - Armstrong, axiomas de (reglas de inferencia) 410

- arquitectura *véase* ANSI/SPARC, arquitectura; SGBD, arquitectura de
- AS (SQL) 199, 207
- AS/400 266
- ASC (SQL) 214, 224
- aserción 223, 644
- asignación (de espacio en disco) 80
- asignación contigua 80
- asignación enlazada 80
- asignación indizada 80
- asociación 638
- asociación, relación de 656
- ASSERT, instrucción 222
- asterisco (*) en SQL 199-200
- atomicidad de una transacción 541
- átomo (cálculo relacional) 237, 250
- atributo 23, 42, 141, 667
- atributo(s) (modelo ER) 42-48
- anidación de 43-44, 47
- clave 45
- compuesto 43
- de clave compuesta 46
- definición formal de 46-47
- derivado 44
- específico 616
- multivaluado 43
- simple (monovaluado) 43
- atributo(s) (modelo OO) 667-677
- oculto 673
- visible 673
- atributo(s) (modelo relacional) 142
- clave *véase* clave, en el modelo relacional
- conservación de 428-429
- de clave externa *véase* clave externa
- de clave primaria *véase* clave primaria
- primario 414
- atributo atómico 141
- atributo de referencia (OO) 672
- atributo de referencia (relacional) 150
- atributo específico 616
- atributo multivaluado 43, 652-653
- atributo no primo 414
- atributo virtual *véase* atributo derivado
- atributos, dominio de *véase* dominio
- atributos, herencia de *véase* herencia
- aumento, regla de
 - para DF 409-412
 - para DMV 441-442
- AUTHID (DB2) 281
- AUTOMATIC, inserción en conjuntos 302
- autonomía local 716
- autorización 600-607
- comprobación de 269
- empleando vistas 604
- en DB2, 281-283
- identificador de 189, 281, 603, 605
- subsistema de 13
- (*véase también* acceso, control de; privilegio)
- AVG (QUEL) 247
- AVG (SQL) 208-210
- AVG (QBE) 256
- AVGU (QUEL) 247
- axiomas 737
- base 737
- deductivos 736
- (*véase también* Armstrong, axiomas de)
- Bachman, diagrama de 292
- Bachman, flecha de 294
- base de cómputo de confianza 610
- base de conocimientos, sistemas de gestión de 730
- base de datos 1, 2, 3, 25
- administrador de *véase* DBA
- aplicación de 6
- arquitectura de *véase* SGBD, arquitectura de
- auditoría de 602
- autorización de *véase* autorización
- clave de 336
- computador de 765, 800
- consistencia de *véase* integridad, restricciones de
- descripción de 24
- diseñador de 9
- ejemplares de 25
- esquema de *véase* esquema
- estado de la 25, 542, 645
- estructura de 22-23
- extensión de *véase* extensión
- hardware de 765-766
- hojear una *véase* hojear
- integración de *véase* integración
- integridad de *véase* integridad, restricciones de
- restricciones de
- intensión de *véase* intensión
- interfases de 30-31
- lenguajes de 29-30
- máquina de 765, 800-801
- nuevas aplicaciones de 770-771
- procedimiento de 288-289
- protección de *véase* seguridad
- recuperación de *véase* recuperación

registro de *véase* registro
 respaldo de 15, 594-595
 seguridad de *véase* seguridad
 sistema de gestión de *véase* SGBD
 tecnología de, avance de la 763-770
 usuarios de *véase* usuario
 vigilancia de *véase* vigilancia
 base de datos deductiva 14, 730-758
 interpretaciones de reglas de 737-739
 introducción a 731-732
 mecanismos de inferencia de 739-742
 notación de 732-736
 sistemas de 753-758
 base de datos distribuida replicada 712-716
 parcialmente 713
 totalmente 712
 base de datos local 716
 base de datos lógica (IMS) *véase* base de datos deductiva; LDB (IMS)
 bases de datos activas 780-785
 bases de datos científicas 789-792
 bases de datos distribuidas 705-726
 conceptos y arquitectura de 707-710
 razones para distribuir 705-707
 tipos de 716-717
 bases de datos estadísticas/científicas 789-792
 bases de datos federadas 34, 716
 bases de datos geográficas *véase* GIS
 bases de datos temporales 571, 792-795
 basura, recolección de 593
 BETWEEN (SQL) 213-214
 bit 70
 tipo de datos (SQL) 190
 bit de modificación 582
 bitácora 539-540
 de escritura anticipada *véase* escritura anticipada, bitácora de
 forzar escritura de *véase* forzar escritura de la bitácora
 BLOB (objeto binario extenso) 77, 680, 786
 bloque (disco) 72-73
 ancla de 106-108
 dirección de 72
 tiempo de transferencia de 73-74, 809-810
 bloquear elemento 559-560
 bloque 559-568, 572-573
 en DB2 285
 estados de 559-561
 gestor de 559
 implementaciones de 560, 561
 operaciones de *véase* bloquear elemento; desbloquear; bloquear_lectura;
 bloquear_escritura; degradar (un bloque); promover (un bloque)
 protocolo de *véase* bloquear de dos fases
 tabla de 560
 tabla de compatibilidad de 572
 bloque de dos fases básico 564
 bloque de dos fases conservador 564
 bloque de dos fases estricto 564
 bloque_escritura, operación 561
 bloque_lectura, operación 560-562
 bloque mortal 565-568
 detección de 567
 distribuido 723
 prevención de 567
 bolsa 196, 642
 constructor de (OO) 669
 Boyce-Codd, forma normal de *véase* FNBC
 BTREE (QUEL) 244-245
 buscar (orden de acceso a archivos) 81
 buscar siguiente (orden de acceso a archivos) 81
 búsqueda
 árbol de 120-121
 campo de 120
 condición de *véase* selección, condición de tiempo de 73, 808-810
(véase también dispersión; índice, búsqueda en; árbol B+; búsqueda en)
 búsqueda binaria 85, 86, 498, 523
 búsqueda lineal 80, 83, 498, 523
 búsqueda primero en amplitud 742
 búsqueda primero en profundidad 741
 BY (QUEL) 247-248
 byte 70
 C++, lenguaje 6, 13, 29, 666, 691, 692-698, 768
 C, lenguaje 189, 225, 265, 768
 cabeza de lectura/escritura 73
 cabeza fija, disco de 73
 cabeza móvil, disco de 73
 caché, memoria 69
 CAD (diseño asistido por computador), bases de datos de 771-775
 cadena, tipo de datos 641
 CAE (ingeniería asistida por computador), bases de datos de 771
 CAF (recurso "llamado de conexión") 265-266
 caída, recuperación de una *véase* recuperación
 CALC
 clave (IDMS) 334-340
 modo de localización 334

cálculo de predicados 235, 237, 735
 cálculo de predicados de primer orden 234, 730
 cálculo relacional 234-243, 250-252
 de dominios 250-252
 de tuplas 234-243
 calificación de nombres (SQL) 198
 CALL, interfaz (IMS) 384
 CAM (fabricación asistida por computador) 771
 cambio de nombre
 de atributos 154-155
 de funciones 678
 en O2 687
 en SQL 199-200, 207
 camino, expresión de 651
 campo (de un registro) 76-77
 campo almacenado (DB2) 279
 campo clave de un archivo 84, 88, 106
 campo de clave primaria 106
 campo de referencia 98
 campo opcional 77-78
 campo ordenado 84
 campo repetitivo 77, 79
 canal furtivo 609
 candado compartido 560-562
 candado binario 559-560
 candado de certificación 572
 candado exclusivo 560-562
 carácter *véase* byte
 cardinalidad, razón de 51
 carga de datos 25
 carga, utilidad de 32
 en DB2 280-281
 CASCADE (SQL) 192-195
 CASE, herramientas 174, 481, 797, 804
 catálogo 6, 31, 484-493
 de DB2 *véase* DB2, catálogo de
 de red 489-491
 en SQL 189-190
 funciones de costo del 522-523
 relacional 485-489
 tipo de información almacenada en el 491-493 *(véase también* diccionario de datos)
 categoría (modelo ECR) 625-629
 transformación al modelo relacional 634-635
 categoría parcial 627-628
 categoría total 628
 CBTREE (QUEL) 245
 CD-ROM 74
 cerrar (orden de acceso a archivos) 82
 certificación, protocolos de 573-574
 CHASH (QUEL) 245
 CHECK (DDL de red) 306, 311
 CHECK (SQL) 222
 CHECK OPTION (vistas de SQL) 221, 284
 CHECK, utilidad (DB2) 306, 311
 ciclo de vida
 de un sistema de base de datos 455
 de un sistema de información 454-455
 CICS (sistema de control de información de clientes) 263-267, 375-376
 cierre
 de un conjunto de atributos 411
 de un conjunto de DF 409-412
 de un conjunto de DF y DMV 442
 recursivo 170, 748
 cifrado 601
 cilindro (disco) 72, 809
 CM (fabricación integrada por computador) 771
 cinta magnética 69, 74
 clase
 en el modelo EER 658
 en el modelo OO 674, 676, 678-679
 clase agregada 659
 clase base 693
 clase de objetos abstractos 658
(véase también abstracción de los datos)
 clase de objetos concretos 658
 clase derivada 693
 clase persistente 679
 clase raíz 678
 clase, definición de 684-685
 clase, jerarquía/retícula de
 en el modelo EER 615-628
 en el modelo OO 678-679
 clase, propiedad de 637
 clasificación 636-637, 678-679
 de atributos 608
 de tuplas 608
 múltiple 637
 clave, atributo *véase* atributo (ER), atributo (relacional)
 clave, restricciones de 642-643
 en el modelo ER 45-46, 55
 en el modelo relacional 145-146, 414, 435-436
(véase también clave externa)
 clave a dirección, transformación de *véase* dispersión, función de
 clave aparente 608
 clave candidata 146, 709
 clave de ordenación, campo de 84

- clave externa 149, 445
- clave parcial 55
- clave primaria 146, 149, 414
- clave secundaria 414
- clave secundaria, campo de 109
- clave sustituta 635
- cliente 33
- cliente-servidor, arquitectura 33, 263, 706-710
- CLOSE CURSOR (SQL) 227
- CLUSTER (SQL) 225
- CNT. (QBE) 256
- cobertura
 - de un conjunto de DF 411
 - mínima 412, 430
- COBOL 6, 225, 265, 290, 315, 768
- cociente *véase* DIVISION, operación
- CODASYL (Conference on Data Systems Languages) 290
- coincidencia parcial, comparación de
 - en QUEL 248-249
 - en SQL 213-214
- colección (OO) 669-670
- colección persistente 675
- colección, tipos de (OO) 669-695
- colisión (dispersión) 90
- columna 140-190
- COMMIT (DB2) 284-285
- comparación de modelos de datos 811-813
- compartimiento de los datos 8, 705
- compatibilidad de unión 160
- compilación *véase* consultas, compilador de
- compilador 11
 - (*véase también* IDDL, compilador de; DML, compilador de)
- compleción relacional 234-235
- comportamiento 23
- composición de funciones 651
- comunicación
 - costo de la 717-719
 - falla de 706
 - red de 707
 - software de 33
- comunicación de datos 769
- condición, cuadro de (QBE) 254
- conexión de propiedad 658
- configuración 684
- confirmación 538-541
 - punto de 540
 - (*véase también* transacción, confirmación de; dos fases, confirmación de)
- CONFIRMAR_TRANSACCIÓN 538
- conjunto (modelo de red) 292-302
- comparación con un conjunto matemático 293-294
- declaración de 309-314
- ejemplar (ocurrencia) 293
- implementación de 297-298
- opciones de inserción en, 302, 304-305
- opciones de ordenamiento en 305
- opciones de retención en 303-304
- opciones de selección en 309-311
- tipo de 292, 305-309
- conjunto acoplado al propietario 292-298
- conjunto constante (SQL) *véase* conjunto explícito
- conjunto de escritura 564, 574
- conjunto de lectura 564, 574
- conjunto explícito (SQL) 206-207
- conjunto mágico 750-751
- conjunto multimiembro 295
- conjunto potencia 46, 414
- conjunto propiedad del sistema 295
- conjunto singular *véase* conjunto propiedad del sistema
- conjunto valor 46-47
 - (*véase también* dominio)
- CONNECT (DML de red) 329-330
- conocimientos, representación de (KR) 635-640
- conservación de la consistencia, propiedad de 541-542
- consistencia de los datos *véase* integridad, restricciones de
- CONSTRAINT (SQL) 192, 193-194
- constructor de arreglos (OO) 668-669
- constructor de átomos (OO) 668-673
- constructor de conjuntos (OO) 668-673, 693
- constructor de tuplas (OO) 668-673
- constructor, operación (OO) 674-675, 693
- consulta(s) 5
 - analizador sintáctico de 495
 - árbol de 508
 - compilador de 32, 495
 - costos de 520
 - descomposición de 720
 - en Prolog/Datalog 734
 - grafo de 516
 - lenguajes de *véase* DAPLEX, FQL, QUEL, QBE, SQL mediante formas 720
 - modificación de (vistas) 221
 - optimización de 508-527
 - optimización de árbol de 508-516
 - optimización de grafo de 516-519

- cuarta forma normal *véase* forma normal
- cuarta generación, lenguaje de (4GL) 768
- cubeta (de dispersión) 91-97
 - división de 95, 97
- cuenta privilegiada 601
- CULPRIT, redactor de informes 335, 769
- CURRENT OF CURSOR (SQL) 228
- cursor (SQL) 226-228
- d (restricción de disyunción en EER) 621
- D. (QBE) 257-258
- DAPLEX 650
- Datalog 731-736, 742-752
 - evaluación de 742-752
 - notación de 734-736
 - programas en 742-752
 - reglas de 731-736
- datos 2
 - datos espaciales, gestión de 792-795
 - datos replicados 723-725
 - datos virtuales 8
- DB/DC (base de datos/comunicación de datos) sistema 33
- DB2 228, 253, 262-288
 - almacenamiento de datos en 277-281
 - arquitectura de 263-269
 - autorización en 281-283
 - bloqueo en 683
 - características de rendimiento de 286-288
 - catálogo de 270-271
 - DB2/2 265
 - DB2i (DB2 interactivo) 265, 268-269
 - manipulación de datos en 271-277
 - procesamiento de aplicaciones en 266-267
 - procesamiento de transacciones en 283-284
 - seguridad en 281-282
 - utilerías de 280
- DBA 9, 30
- DBA, interfaces del 30
- DBACTRL, privilegio (DB2) 283
- DBADM, privilegio (DB2) 283
- Dbase 2, 263, 765
- DBD (definición de base de datos) (IMS) 376, 378
- DBRM (Módulo de solicitud a la base de datos) (DB2) 266
- DBS *véase* sistema de base de datos
- DBTG (Database Task Group) 292
- DBTG, modelo *véase* modelo de red
- DCLGEN (generadores de declaraciones) (DB2) 275
- procesamiento de 495-508
- transformación de 509-511
- validación de 495
- consulta anidada (SQL) 202-204
- correlacionada 203-204
- consulta estadística 611
- consulta recursiva 170, 748
- consultas, procesamiento distribuido de 717-722
 - costos de transferencia de datos 717-719
 - descomposición de consultas 720-722
 - técnica de semirreunión 719-720
- CONTAINS (SQL) 204
- control de concurrencia 8, 534-536, 558-577
 - (*véase también* bloqueo, multiversión, seriabilidad, marca de tiempo)
- control de concurrencia distribuido 722-726
 - copia primaria 724
 - sitio de respaldo 724
 - sitio primario 724
 - votación *véase* votación
- control de concurrencia optimista 573-574
- conversión, herramientas de 33
- coordinador 593, 723-724
- copia primaria 724
- CORAL 659-660
- correspondencias entre niveles 27
- costo de acceso 519-527
- costo de transferencia de datos 717-719
- costo, estimación de 520-527
- COUNT (QUEL) 246-247
- COUNT (SQL) 208-213
- COUNTU (QUEL) 247
- creación de ejemplares 636-637
- creación de múltiples ejemplares 608-610
- CREATE (QUEL) 244-245
- CREATE ASSERTION (SQL) 222
- CREATE DOMAIN (SQL) 189, 605
- CREATE INDEX (SQL) 189, 224-225
- CREATE SCHEMA (SQL) 189, 605
- CREATE TABLE (SQL) 190-193
- CREATE VIEW (SQL) 219
- CREATETAB, privilegio 605
- CRU *véase* actual de unidad de ejecución
- CSP (Producto intersistemas) 265-266
- cuantificador existencial 237-240
 - (*véase también* ANY (QUEL), EXISTS (SQL))
- cuantificador universal 206, 237-243
- cuantificadores 237
 - transformaciones de 240

DDL 28
compilador de 32, 501
(*véase también* modelo de datos jerárquico; modelo de datos de red; enfoque orientado a objetos; modelo de datos relacional; SQL)
decisiones, sistema de apoyo a la toma de *véase* DSS
DECLARE CURSOR (SQL) 227
deducción
mecanismos de 732-750
reglas de 14, 731-739
DEFAULT (SQL) 191-192
definición de datos 7
en DB2 269-270
(*véase también* modelo de datos jerárquico; modelo de datos de red; enfoque orientado a objetos; modelo de datos relacional)
definición de tablas (de una vista) 219
degradar (un bloqueo) 562-563
DELETE (QUEL) 249
DELETE (SQL) 217
privilegio 604
dependencia funcional de manera total 416
dependencia multivaluada *véase* DMV
dependencia no transitiva 417
dependencia parcial 416
dependencia transitiva 417
dependencias funcionales (DF) 406-412
conjuntos mínimos de 412, 430
equivalencia de conjuntos de 411-412
proyección de un conjunto de 429
reglas de inferencia de 408-411, 442
dependencias, conservación de las 414, 429
DESANDAR, operación 416, 656
desbloquear, operación 559-562
desborde, archivo de 87
DESC (SQL) 214, 224
descomposición de consultas 516-519, 720-721
descomposición de relaciones 414-421, 428-436, 444
descomposición relacional 414-421, 434-435
descriptor (SQL) 189
DESHACER, entradas de bitácora tipo 583
DESHACER, operación 538, 540, 591
DESHACER/NO REHACER, protocolo 580, 590
DESHACER/REHACER, protocolo 580, 590-591
desnormalización 476-478
DESTROY (QUEL) 245
orientadas a objetos 699, 700
destructivo, operador (OO) 674-675
DF *véase* dependencia funcional
diagrama de estructura de datos *véase* Bachman, diagrama de
diagrama de
notaciones alternativas 56, 804-807
diario *véase* bitácora
diccionario de datos activo 484
diccionario de datos, sistemas de 33, 453, 484
DIFERENCIA, operación 160-162
dirección *véase* bloque, dirección de; registro, dirección de
direccionamiento abierto 90
DIRECTO, modo de localización (modelo de red) 335
directorio de datos 33
disco (magnético) 31-32, 70-73, 535, 808-810
capacidad de 71
formato de (iniciación) 72
paquete de 71
parámetros de 808-810
planificación de 32
unidad de 72
(*véase también* bloque, tiempo de transferencia de; velocidad de transferencia masiva; cilindro; latencia; reescritura, tiempo de; retardo rotacional; búsqueda, tiempo de; pista; transferencia, velocidad de)
disco óptico de una sola escritura *véase* WORM
disco óptico, almacenamiento en 69, 766
DISCONNECT (DMT de red) 330
diseño *véase* base de datos, diseño de; sistemas de información, diseño de; diseño relacional
diseño asistido por computador (CAD), bases de datos de 771-775
diseño conceptual 39-41, 459-468, 624-625
descendente vs. ascendente 625
ejemplo de (base de datos COMPANIA) 41-42
47-48, 55-56
refinación del 55-56, 459-468
diseño conceptual descendente 392
refinación del 424-425
diseño de bases de datos
conceptual 39-41, 459-468
de red 311-314
fases del 39-40, 456
físico 39, 471-472
jerárquicas 360-366
lógico 39, 470-471
orientadas a objetos 699, 700

dominio simple *véase* dominio atómico
dominio, variable de 250-251
dominios, cálculo relacional de 250-252
dorsal, máquina 708
dos fases, protocolo de confirmación de 593, 723, 725
dos fases, protocolos de bloqueo de 562-565
básicos 564
conservadores 564
estrictos 564
multiversión 572-573
DOS/VSE, sistema operativo 262-263, 375
DROP INDEX 225
DROP SCHEMA (SQL) 194
DROP TABLE (SQL) 194
DROP VIEW (SQL) 220
DSS 777-778
duplicación de información 12
DUPLICATES NOT ALLOWED (DDL de red) 308
durabilidad, propiedad de 541-542
DXT (Utilería para extracción de datos) 265
EER, modelo 615-630
definición formal del 629-630
diagramas del 616-617, 804-807
ejemplo 629-631
operaciones de actualización del 646-649
transformación al modelo OO 699-700
transformación al modelo relacional 630-635
(*véase también* categoría, generalización, especialización, subclase)
ejecución concurrente 75
ejecución perezosa 755
ejemplar *véase* base de datos, ejemplar de; entidad, ejemplar de; interrelación, ejemplar de
ejemplo, valor de (QBE) 253
ejemplo, variable de (QBE) 253
elecciones (en sistemas distribuidos) 725
elemento de información 5
elemento de información 5, 6, 76, 344
en el modelo de red 293, 307-309
real 293
virtual 293
eliminación
anomalías de 402
bloqueo de 576
marcador de 84
Eliminar (modelo relacional) 151, 152-153
procesamiento de 39-41, 456-472
relacionales 174-179, 397-406, 428-445
(*véase también* normalización)
diseño de bases de datos distribuidas 710-716
(*véase también* reparto; fragmentación; replicación)
diseño de bases de datos relacionales
algoritmos de 428-445
normalización en 413-421
pautas de 397-406
transformación ER a relacional para 174-179
diseño físico de bases de datos 39, 471-472
diseño lógico de bases de datos 39, 470-471
diseño, bases de datos para 770-775
disparador 223
dispersión 88-98, 131
clave de 88
como índice 131
dinámica 93-95
extensible 95-97
externa (en disco) 91-93
función de 88-90
interna 88-89
lineal 97-98
lógica vs. física 131
disponibilidad 705
DISTINCT (SQL) 200-203
distribución de los datos *véase* reparto
disyunción, restricción de 621
total vs. parcial 422
DIVISION, operación de 166-167, 204, 206
DL/I (Data Language One) 383-387
DMCL (lenguaje de control de dispositivos) 338-339
DML 29
comparación de 812-815
compilador de 32
(*véase también* HDML)
DML no por procedimientos 29
DML orientado a conjuntos 29
DML por procedimientos 29
DMS-1100 333-334
DMV 440
no trivial 441
reglas de inferencia de 441-442
trivial 441
documentos, gestión de 785-787
dominio 46, 140
atómico 140
restricciones de 145, 640-641
dominio/clave, forma normal *véase* FNDK

eliminar (orden de acceso a archivos) 81
 encadenamiento 90
 encadenamiento hacia adelante 739-740
 encadenamiento hacia atrás 740-742
 encapsulamiento 673-676
 enfoque orientado a objetos
 análisis *véase* OOA
 conceptos del 665-667
 definición de datos en el 672
 diseño de bases de datos con 699-701
 esquema 672
 modelos de datos 23, 35, 667-681
 restricciones en el 644
 SGBD *véase* SGBDOO
 enlace 682
 enlace, tipo de registro de 298, 301
 enlace automático (DB2) 269
 enlace tardío 682
 enlazador (DB2) 266
 entero, tipo de datos (SQL) 190
 entidad(es) 23, 25, 42
 conjunto de 45
 integridad de 149, 608
 relación de 174
 tipo de 44-46
 entidad-vínculo *véase* ER
 entidades débiles, tipo de 54-55, 60-62
 EQUOL 249
 EQUIRREUNIÓN, operación 165
 equivalencia de conflictos 548
 equivalencia de
 conjuntos de dependencias 411
 expresiones relacionales 511-515
 planes de transacciones *véase* plan
 ER, modelo 38-67
 (*véase también* EER; entidad; vínculo; atributo)
 ER extendido, modelo *véase* EER, modelo
 ERASE (DML de red) 330
 ERASE (IDMS) 340
 ES-UN, vínculo 615, 629
 escribir _elemento, operación 533
 escritura anticipada, protocolo de bitácora de 583
 escritura ciega 554
 escritura no restringida 554
 escritura, fase de (de una transacción) 574
 espacio de índice (DB2) 277-280
 espacio de tabla (DB2) 277-280
 con particiones 279
 segmentado 279

especialización 616-628, 637
 definida por atributos 621
 definida por el usuario 621
 definida por predicados 620
 jerarquía/retícula de 622-625
 proceso de 618-619
 restricciones de 620-621
 transformación al modelo relacional 630-634
 especialización parcial 621
 especialización total 621
 espera cautelosa 566
 espera indefinida 567
 esquema
 correspondencia de 469
 DDL de 335
 diagrama de 24-25, 29
 elemento de 25
 en SQL 189-190
 estrategias de diseño de 461-465
 evolución de 667, 692-693
 gestor de 692
 integración de 462-465, 716-717
 (*véase también* vistas; integración de)
 traducción de *véase* transformación
 transformación de *véase* transformación
 (*véase también* EER; ER; modelo de datos
 jerárquico; modelo de datos de red;
 enfoque orientado a objetos)
 esquema conceptual 26, 39, 459-462
 esquema ER 56
 ER a jerárquico, transformación de modelos 360-366
 ER a OO, transformación de modelos 699-700
 ER a red, transformación de modelos 313-316
 ER a relacional, transformación de modelos 174-179
 esquema externo 26
 esquema interno 26
 estado consistente 542, 645
 estado de relación permitido (ejemplar) 407
 estado válido 26
 estado, indicador de 319-320, 368
 estado, restricción de *véase* restricción
 estimación *véase* costo, estimación de
 estrella, propiedad de 608
 EXCEL 2
 excepción, objetos de 637
 EXCEPT (OO) 683
 EXCEPT (SQL) 201

forma 30, 766
 forma clausal 735-736
 forma normal (de las relaciones) 412-422
 Boyce-Codd (FNBC) 421-422
 cuarta (4FN) 442-444
 dominio/clave (FNDC) 447
 primera (1FN) 144, 414-416
 proyección-reunión (FNPR) 445
 quinta (5FN) 445
 segunda (2FN) 416-417, 416-419
 tercera (3FN) 417, 419-421
 formas, interfaz de 766
 fórmula
 en cálculo relacional 237, 251
 en Prolog/Datalog 734
 fórmula atómica 734
 fórmula bien formada (fbf) 237
 FORTRAN 265, 767-768
 forzar escritura de la bitácora 541
 FQL *véase* lenguaje de consulta funcional
 fragmentación 710-712
 de esquemas 712
 horizontal 710
 mixta 711-712
 vertical 711
 fragmentación completa 712
 fragmento 710
 fragmento mixto *véase* fragmentación
 fragmento vertical *véase* fragmentación
 frontal, máquina 708
 función 7, 676
 (*véase también* funciones agregadas; operación)
 función derivada 650-654
 función integrada *véase* funciones agregadas
 función inversa 651
 funciones agregadas 167-170
 en DML de red 328
 en QUEL 247-248
 en SQL 208-213
 fusión (o combinación) de versiones 683
 G. (QBE) 256-257
 generalización 618-624, 629-631, 638
 jerarquía de 623-624
 proceso de 619
 restricciones de 620-622
 retícula de 623
 transformación al modelo relacional 631-634
 generalización total 621
 GENESIS, sistema 796
 EXCLUSIVE (DB2) 285
 EXEC SQL 225, 284, 286
 EXECUTE (SQL dinámico) 286
 exhibición 768-769
 existencia, dependencia de 51
 EXISTS (SQL) 204-206
 EXODUS, sistema 796
 explosión de componentes *véase* vínculo
 recursivo
 exportación, esquema de 716
 exportación, recurso de (O2) 690
 expresión *véase* álgebra relacional, cálculo
 relacional
 expresión del cálculo relacional segura 242-243
 EXPRESS, lenguaje 774
 extensión 25, 45, 141
 extracción de información 787
 fabricación asistida por computador (CAM) 771
 fabricación integrada por computador (CIM) 771
 fabricación, bases de datos para 771-772
 factor de bloques 79-80
 (*véase también* registro(s), bloques de)
 factor de carga (archivo disperso) 97
 fallos, tipos de 537
 fase de lectura 573-574
 fecha, tipo de datos 641-642
 en SQL 190-192
 FETCH (SQL) 227-228
 orientación 228
 fiabilidad 705
 FIFO (first-in-first-out) *véase* FEPS
 FIJA, retención de conjuntos (modelo de red) 305
 fila 140, 190
 filtrado 608
 finalizar transacción 547
 FIND (DML de red) 322-327
 FIND ANY (DML de red) 322
 FIND CALC (IDMS) 339
 FIND DBKEY (IDMS) 339
 FIND DUPLICATE (DML de red) 322
 FIND FIRST (DML de red) 324
 FIND NEXT (DML de red) 324
 FIND OWNER (DML de red) 324
 FIND PRIOR (DML de red) 324
 FNBC 321-322
 FNDC *véase* forma normal
 FNPR (proyección-reunión) *véase* forma normal
 FOR UPDATE OF (SQL) 228-229

genoma, bases de datos de 777-780
 gestor de datos almacenados 31-32
 en DB2 267
 GET (DML de red) 321
 GET FIRST (HDM) 368
 (véase también GET UNIQUE)
 GET FIRST PATH (HDM) 370
 GET HOLD (DL/I) 383-384
 GET HOLD (HDM) 374
 GET NEXT (DL/I) 385
 GET NEXT (HDM) 368
 GET NEXT PATH (HDM) 370
 GET NEXT WITHIN PARENT (DL/I) 385
 GET NEXT WITHIN PARENT (HDM) 371
 GET UNIQUE (DL/I) 368, 383-385
 GH véase GET HOLD (DL/I)
 GIS (Sistema de información geográfica) 793
 GN véase GET NEXT (DL/I)
 GNP véase GET NEXT WITHIN PARENT (DL/I)
 grado
 de una relación 14
 de un vínculo 49-50, 59-62
 grafo acíclico 548-550, 567
 grafo de espera 567
 grafo de precedencia 548-550
 grafo de seriación 548-549
 grafo de versiones 683
 GRANT (SQL) 605-606
 en DB2 281-282
 GRANT OPTION (SQL) 604-607
 en DB2 281-282
 granularidad de los elementos de información 575
 GROUP BY (SQL) 210-213
 grupo (de bloques en disco) 72, 80
 grupo repetitivo 77, 293
 GU véase GET UNIQUE (DL/I)
 guardia, condición de 710-712
 HASH (QUEL) 245
 having (SQL) 212-213
 HDAM (método de acceso directo jerárquico) 390
 HDDL 359-362
 HDML 366-375
 HEAP (QUEL) 245
 HEAPSORT (QUEL) 245
 hecho 2, 371
 herencia
 en el modelo EER 616, 623-626
 en el modelo OO 676-678

implementación, modelo de datos de 23
 impulsor 73
 IMS, sistema 263-265, 375-392
 arquitectura del 376-377
 base de datos lógica en 380-381
 estructuras de almacenamiento en 385-392
 estructuras lógicas de datos en 378-383
 indización secundaria en 391-392
 manipulación de datos (DL/I) en 383-387
 vistas en 380-383
 IMS/DC 264
 IMS/VS-DC 264-265, 375-376
 IN (SQL) 202-203
 inanición 568
 inclusión, dependencia de 445, 632
 incompatibilidad de impedancia 14, 691
 inconsistencia 12
 independencia de los datos 27
 (véase también independencia de programas y datos)
 independencia de programas y datos 7
 independencia de programas y operaciones 7
 índice 106-134, 223-225
 abanico de véase abanico
 basado en dispersión 131
 bloqueo de 576
 búsqueda en 117-118
 en DB2 280
 en QUEL 245
 en SQL 223-225
 entrada de (registro) 106
 exploración de 498
 factor de bloques de 105
 niveles de 115
 tipos de 105-115
 (véase también árbol B; árbol B+; agrupamiento)
 índice denso 108, 110
 índice dinámico de múltiples niveles 118
 índice lógico 131
 índice no denso 108-109
 índice primario 106-109
 índice secundario 109-114
 indización
 atributos de 224
 campos de 106
 inferencia 14, 731
 inferencia estadística 698
 inferencia, máquina de 731, 739-742
 INFORMATION_SCHEMA (SQL) 190
 INFORMIX 262-263, 765
 en O2 685-687
 selectiva 625, 683
 herencia múltiple 623, 682-683
 en el modelo EER 623
 en el modelo OO 682-683
 en O2 687
 herramientas 11
 herramientas de diseño 480-481
 HIDAM (método de acceso jerárquico indizado directo) 390-391
 hijo virtual 356
 HIPAC, sistema 784
 hiperespacio (DB2) 287
 hiperfondo (DB2) 287
 hipermedia 787
 HIPO (entrada/procesamiento/salida jerárquicos) 459
 HISAM (método de acceso indizado secuencial jerárquico) 388-391
 historia (de transacciones) 542-543
 (véase también plan)
 hoja, tipo de registro 350-351
 hojear 30
 hora, tipo de datos 641-642
 en relaciones véase base de datos temporal
 en SQL 190-192
 Horn, cláusulas de 735-736
 HSAM (método de acceso secuencial jerárquico) 387-388
 I. (QBE) 257-258
 idempotencia de la recuperación 591-592
 identidad, conexión de 658
 identificación 637-638
 (véase también objetos, identidad de)
 IDM (Máquina de base de datos inteligente) 765
 IDMS 333-342
 arquitectura de 334-335
 definición de datos en 336-337
 estructuras de almacenamiento en 340-342
 manipulación de datos en 339-341
 IDS (Almacén de datos integrados) 333-334
 IGES (Especificación para intercambio de gráficos inicial) 773
 igualdad de objetos 770-772
 IMAGE, sistema 333-334
 imagen
 atributo de 633
 indización de 788
 imagen "antes" (BFIM) 582
 imagen "después" (AFIM) 582

ingeniería asistida por computador (CAE), bases de datos de 770-771
 ingeniería concurrente 683
 ingeniería de software 797-798
 ingeniería, datos de diseño en 797-798
 INGRES 228, 243, 262-263, 769
 (véase también QUEL)
 iniciar transacción 541
 inmutabilidad (de OUP) 668
 inserción
 anomalías de 401-402
 bloqueo en 576
 opciones de (modelo de red) véase conjuntos, inserción en
 INSERT (SQL) 216-217
 INSERT, privilegio 604
 insertar (modelo relacional) 152
 insertar (orden de acceso a archivos) 81
 integración véase esquemas, integración de; vistas, integración de
 integridad, aserción de
 integridad, restricciones de 15-16, 640-646
 comparación de 816-817
 en el modelo de red 303-307
 en el modelo ER véase ER, modelo
 en el modelo jerárquico 359
 en el modelo relacional 145-151
 en SQL 193-195, 222-223
 imposición de 643-644
 subsistema de 644
 tipos de véase restricciones, tipos de
 integridad referencial 149-150, 143
 en SQL 192-194
 opciones cuando se viola la 153, 192-194
 (véase también clave externa)
 integridad semántica, restricción de 151, 643-644
 inteligencia artificial 798-799
 INTELLECT 767
 intensión 23, 45, 141
 (véase también esquema)
 Interbase 785
 intercalación de operaciones 75, 532-533
 interfaz
 de lenguaje natural 30-31, 766-767
 de una función (operación) 7, 673
 (véase también signatura)
 interfaz amable con el usuario 29-30, 258
 interfaz gráfica 30, 768-769
 interpretación (de las reglas) 737-739

- por la teoría de demostraciones 737
- por la teoría de modelos 737
- interpretación de valores nulos desconocidos 44, 403
- falantes 44, 403
- no aplicables 44, 403
- INTERSECT (SQL) 201
- intervalo
 - consulta de 498
 - relación de 235-236
- intervalo, tipo de datos de (SQL) 192
- intervenciones, registro de 602
- INTO (QUEL) 249
- INTO (SQL incorporado) 226-228
- IS NOT NULL (SQL) 207
- IS NULL (SQL) 207
- ISAM (método de acceso secuencial indizado) 133
- ISAM (QUEL) 245
- ISO (International Standards Organization) 189
- JD 445
- trivial 445
- jerarquía 257, 685-686
- jerarquía (clase) véase especialización, generalización
- JOIN (SQL) 207-208
- KID, sistema 766
- KR, sistema 765
- LADDER, sistema 767
- LAN véase red de área local
- latencia véase retardo rotacional
- LDB (IMS) 376, 381-382
- LDL, sistema 753-756
- lectura no repetible, problema de 536
- lectura sucia 535-536
- leer (orden de acceso a archivos) 81
- leer elemento, operación 533
- lenguaje anfitrión 29, 225, 317
- lenguaje de consulta funcional 652
- lenguaje de definición de datos véase DDL
- lenguaje de especificación de aserciones 644
- lenguaje de manipulación de datos véase DML
- lenguaje de programación 801-802
- lenguaje declarativo 29, 732
- lenguaje natural, interfaz de 30-31, 767-768
- lenguaje relacionalmente completo 234-235
- léxico 658
- LIKE (SQL) 214
- lista enlazada circular véase anillo
- listas, constructor de (OO) 668-673
- literal 735-736
 - negativa 736
 - positiva 736
- localizar (orden de acceso a archivos) 81
- LOCK TABLE (DB2) 285
- longitud fija, registro de 77
- longitud variable
 - campo de 77-78
 - registro de 77-79
- Lotus 1-2-3 765
- LRU (menos recientemente utilizado) 582
- MANDATORY, conjunto (modelo de red) 305
- MANUAL, conjunto (modelo de red) 304
- máquina dorsal 708
- máquina frontal 708
- marca de tiempo 565-568
- generación de 568-569
- tipo de datos (SQL) 190, 192
- marca de tiempo de escritura 568-569, 571
- marca de tiempo de lectura 568-571
- materialización
 - de datos distribuidos 720-721
 - de un vínculo 178-179
 - de una vista 221
- matriz de acceso 603
- MAX (QUEL) 247
- MAX (SQL) 208-209
- MAX. (QBE) 256
- mecanismos de inferencia 739-740
 - ascendentes 739-740
 - descendentes 740-742
- memoria intermedia 32, 533
- desalojo de 582
- memoria principal véase almacenamiento primario
- memoria volátil 69
- mensaje 673
- menú 30
- metaclase 637
- metadatos 3, 6, 24, 482
- método 7, 35, 673-676
 - (véase también función, operación)
- MIN (QUEL) 247
- MIN (SQL) 208-209
- mín-máx, restricción 57
- MIN. (QBE) 256
- mínimundo 2, 15, 640

- MINUS (SQL) véase DIFERENCIA, EXCEPT (SQL)
- Miro, sistema 796
- MOBILE-Burotique 776
- modelado geométrico 771-775
- modelo (bases de datos deductivas) 737-738
 - (véase también modelo de datos)
- modelo de datos binario 650
- modelo de datos de red 34, 292-313
- DDL del 307-313
 - diseño de bases de datos en el 313-316
- DML del 317-333
- estructuras de datos del 293-304
- operaciones de actualización en el 328-333
- pautas para diseño físico en el 476-478
- recorrido en el 322-327
- restricciones del 303-307
 - (véase también IDMS; registro; conjunto)
- modelo de datos físico 23
- modelo de datos funcional 650-653
- modelo de datos relacional 34, 140-174
- definición de datos en el 154-156, 189-196
- lenguajes del véase QBE, QUEL, álgebra relacional, cálculo relacional, SQL
- notación en el 145
- operaciones de actualización en el 152-154
- operaciones de obtención en el véase álgebra relacional
- pautas para diseño físico en el 474-475
- restricciones de integridad en el 145-149
- modelo de datos semántico 658-659
- modelo de datos jerárquico 35, 348-375
- apuntadores en el 356-358, 389
- árbol de ocurrencia en el véase árbol de ocurrencia
- camino en el 355, 384
- diseño de bases de datos en el 360-366
- esquema en el 350-352
- estructuras de datos del 349-358
- lenguaje de definición de datos del véase HDL
- lenguaje de manipulación de datos del véase HDML
- pautas para diseño físico en el 478-479
- restricciones de integridad en el 359
- secuencia en el 354-355, 367
- modelo estructural 658-659
- modelo relacional anidado 652-656
- modelos conceptuales de datos 23, 615-630
- modelos de datos 7, 22, 34, 763-765
- categorías (tipos) de 23
- transformación de 39 (véase también ER a jerárquico, ER a red, ER a OO, ER a relacional, EER a relacional)
- operaciones de 23
- modelos de datos basados en registros 23
- modelos de datos de representación 23, 25
- modificación, anomalías de 400-403
- modificar (operación relacional) 152-154
- modificar (orden de acceso a archivos) 81
- MODIFY (DML de red) 331
- MODIFY (QUEL) 245
- modo de localización (modelo de red) 336-337
- monículo, archivo de 83
- montón, archivo de 83
- muchos a muchos (M:N), vínculo en el modelo de red 300-303
- en el modelo ER 51-53
- en el modelo jerárquico 360-366
- en el modelo OO 699-701
- en el modelo relacional 174-179
- multibase de datos
 - sistema de 34, 716
- transacción de 593
- multiconjunto 195
- multimedia, base de datos de 785-789
- multinivel
 - índice 115-131
 - relación 608
 - seguridad 607-610
- multiprogramación 532
- multiversión (control de concurrencia)
- ordenamiento por marca de tiempo 570-571
- bloqueo de dos fases 572-573
- MVS 262-263, 375
- NAIL, sistema 758
- NATURAL JOIN, operación (SQL) 165-166
- NATURAL JOIN 208
- navegación (o recorrido) 318
- NDL (lenguaje de definición de red) 292
- negación estratificada 752
- nivel
 - de aislamiento véase aislamiento, nivel de
 - de índice véase índice, nivel de
 - de un nodo de árbol 118
 - nivel conceptual 26
 - nivel externo 26
 - nivel físico véase nivel interno
 - nivel interno 26
 - no esperar, protocolo de 566
 - NO-DESHACER/NO-REHACER, protocolo 580, 582, 592-593

- NO-DESHACER/REHACER, protocolo 580, 582, 585, 588
- nodo (de un sistema distribuido) véase sitio
- nodo descendiente 118
- nodo hijo 118
- nodo hoja 118, 125
- nodo interno (árbol) 118, 124-125
- nombrar (en el modelo ER) 59
- nombre de papel inverso 51
- nombre persistente 675, 688
- normalización, algoritmos de 428-444
- normalización, proceso de 412-414
- NOT EXISTS (SQL) 202-204
- NOT NULL (SQL) 191-192
- notaciones diagramáticas
 - EER 616-617
 - ER 56-58
 - otras 804-807
- n-tupla 141
- NULL (SQL) 191, 193, 207
- O2, sistema 684-692
 - arquitectura del 692-693
 - definición de datos en el 684-687
 - lenguaje de consulta del 688-690
 - lenguaje O2C del 685-689
 - manipulación de datos en el 687-691
- O2LOOK 687
- O2SQL 687-691
- O2TOOLS 687
- OBE 777
- ObjectStore 14, 693-698
 - definición de datos en 692-696
 - lenguaje de consulta de 698-699
 - manipulación de datos en 696-699
 - recurso de vínculos en 696
- objetivo (consulta deductiva) 739-742
 - compuesto 741
 - subobjetivo 741
- objeto
 - complejo 679-681
 - constructor de 674
 - destructor de 674-675
 - estructura de 668-672
 - gestor de 692
 - identidad de 668, 684-685
 - identificador de (OID) 667, 685
 - implementación de 673
 - interfaz de 673
 - persistente 675
- transitorio 675
- valor de (estado) 669-671
- objeto binario grande (BLOB) 77, 680, 786
- objeto molecular 640
- objeto persistente 14, 675, 685, 688, 698
- objetos complejos
 - agrupamiento de 681
 - ensamble de 681
 - estructurados 680-681
 - no estructurados 679-680
 - objetos idénticos 670-672
 - obtener (orden de acceso a archivos) 81
 - ocurrencia véase ejemplar
- ODBC (Conectividad de bases de datos abierta) 263
- Office By Example véase OBE
- OFFIS, sistema 776
- oficina, sistema de información de (ois) 775-776
- OLQ (Consulta en línea) 335
- OOA 804
- OPCIONAL, retención de conjuntos (modelo de red) 304-305
- OPEN CURSOR (SQL) 227-229
- operación
 - implementación de 7
 - interfaz de 7
 - (véase también modelos de datos, operaciones de; archivos, operaciones de)
- operación abstracta 8, 673-676
- operaciones de conjuntos (relacionales)
 - 160-163
 - diferencia véase DIFERENCIA
 - implementación de 506
 - intersección véase INTERSECCIÓN
 - producto cartesiano véase PRODUCTO CARTESIANO
- unión véase UNIÓN
- operaciones definidas por el usuario 23, 39, 646-649
 - (véase también función, método)
- operaciones idempotentes véase DESHACER, REHACER
- operadores aritméticos
 - en QUEL 248-249
 - en SQL 213-214
- operadores, sobrecarga de 681-682
- optimización
 - algebraica 508-516
 - de árbol de consulta véase consulta, optimización de árbol de

- de consultas véase consultas, optimización de
- grafo de consulta
- del álgebra relacional véase árbol de consulta, optimización de
- del cálculo relacional véase consulta, optimización de grafo de
- distribuida véase consultas, procesamiento distribuido de
- estimación de costos para 519-527
- optimización heurística de consultas 508-519
- optimización semántica de consultas 527
- optimización sistemática de consultas 520
- ORACLE 228, 263, 766
- orden
 - de un árbol B 121-122
 - de un árbol B+ 123-125
 - de un árbol de búsqueda 119-120
- ordenación externa 84-85
- ordenación por fusión (o combinación) véase ordenación externa
- ordenamiento básico por marca de tiempo 568
- ordenamiento estricto por marca de tiempo 568-570
- ORDER BY (SQL) 214
- ORDER IS (DDL de red) 312
- organizaciones primarias de archivos 70, 84-99
- ORION, sistema 774
- OS/2, gestor de bases de datos (DB2/2) 263, 265
- OS_Bag 696
- OS_List 696
- OS_Set 695-696
- OSAM (método de acceso de desborde secuencial) 390
- OSAM*, sistema 791
- otorgar privilegios 603-607
- P (QBE) 253-258
- padre
 - nodo 118
 - apuntador al 118
 - registro 349-352
 - tipo de registros 349
- padre virtual 356
- padre-hijo, vínculo véase VPH
- página 72, 277, 592-593
 - (véase también bloque)
- paginación de sombra 592-593
- papel (de participación en un vínculo) 50, 625-627
- especializado 617
- nombre de 50
- PARADOX 2, 263
- paralelo, procesamiento en 770
- parte gobernante 657
- participación
 - en un vínculo 49
 - parcial 50
 - restricción de 51
 - total 50
- PASCAL 6, 13, 189
 - incorporado 225-228, 366-375
 - red de, incorporado 322-333
- patrón, dependencia de 446-447
- pautas para el diseño físico de bases de datos
 - en sistemas de red 476-478
 - en sistemas jerárquicos 478-480
 - en sistemas relacionales 474-476
- PBS (IMS) 376
- PCB (IMS) 376
- PDB (IMS) 376, 378-380
- PDES (Intercambio de datos de productos mediante STEP) 774
- PEFS (primero en entrar, primero en salir) 582
- pista (disco) 70-71
- PL/1 225, 265, 317, 768
- plan de acceso véase plan de ejecución
- plan de aplicación 266-268
- plan de ejecución (estrategia) 495-496
- plan de transacciones 542-554
 - completo 543
 - en serie 546
 - equivalencia de 548-549, 554
 - estricto 545
 - proyección confirmada del 544, 552
 - que evita la reversión en cascada 544
 - recuperable 544
 - serializable 547-548
 - serializable por conflictos 548
 - serializable por vistas 553
 - plan sin cascada 544
- plantilla de una relación (QBE) 253
- población 610
- polimorfismo de las operaciones 681-682
- POSTGRES 784
- precompilador 32
 - en DB2 266
 - (véase también preprocesador)
- predicado, bloqueo de 577
- predicado definido por hechos 742
- predicado definido por reglas 742
- predicados (en Prolog/Datalog) 732-747
- base (de ejemplares) 732

- base (de ejemplares) 732
- grafo de dependencias de 746
- preorden, recorrido en 354
- PREPARE (SQL dinámico) 286
- preprocesador 225
- presentación y exhibición 768-770
- PRIMARY KEY (SQL) 191-193
- primera forma normal *véase* forma normal
- privilegio de acceso 602-607
- privilegios 602-608
 - a nivel de cuenta 603
 - a nivel de relación *véase* relación, privilegios de
 - otorgamiento de 603-608
 - propagación de 604, 606
 - revocación de 604, 606
- PROBE, sistema 796
- procedimiento remoto, llamada a (RPC) 691
- procesador de aplicaciones (PA) 708
- procesador de base de datos en tiempo de ejecución 31
- procesador de datos (DP) 708
- procesamiento de transacciones en línea 34
- PRODUCTO CARTESIANO, operación 162-163
- PRODUCTO CRUZADO, operación 162-163, 199-200
- profundidad (dispersión extensible) 95-97
- profundidad global (dispersión) 95
- profundidad local (dispersión) 95
- programa
 - área de trabajo de *véase* usuario, área de trabajo del
 - interfaz de 768-769
 - variable de 317, 366, 533
- programa de aplicación 39
- programador de aplicaciones 10
- PROLOG 265, 732-742
 - notación de 732-734
- promover (un bloque) 562-563
- propiedad 658-659
- propiedad, conexión de 657
- propiedad, semántica de 681
- propietario
 - apuntador al 299-300, 303
 - de cuenta 603
 - entidad 55
 - registro 295
 - tipo de entidad 55
 - tipo de registro 294
- propietario identificador 54-55
- protección de los datos *véase* seguridad
- protección *véase* seguridad
- protocolo de ordenamiento por marca de tiempo 568-570
 - estricto 570
 - multiversión 570-572
- protocolo esperar-morir 570-571
- protocolo herir-esperar 566
- protocolos *véase* recuperación
- proyección, atributos de 158-159
- proyección confirmada 544, 552
- proyección-reunión, forma normal de *véase* forma normal
- PROYECTAR, operación 158-159
- implementación de la 506
- PUBLIC (DB2) 283
- punto de entrada (modelo de red) 297
- punto de control 541
- QBE (Consulta por ejemplo) 188, 253-258
- QBF (Consulta por formas) 253
- QMF (Recurso de gestión de consultas) 265
- QUEL 188, 243-250
 - actualizaciones en 249-250
 - comparación con SQL 250-251
 - consultas en 245-249
 - definición de datos y de almacenamiento en 243-245
 - tipos de datos en 243
- quinta forma normal *véase* forma normal
- raíz persistente 688
- RANGE (QUEL) 245
- razonamiento, mecanismos de 636
- RBASE 5000 263
- RDB 262-263
- RECONNECT (DML de red) 332-333
- recorrido (o navegación) 318
- recuperabilidad de planes 544-545
- recuperación 15, 536-537, 580-595
 - bosquejo de la 581-582
 - conceptos de 582-586
 - de un fallo catastrófico 594-595
 - en múltiples bases de datos 593-594
 - gestor de 594
 - tipos de fallos 537
- (*véase también* actualización diferida; actualización inmediata; REHACER, confirmación de dos fases; DESHACER)
- recuperación distribuida 725-726
- red (comunicaciones)
 - de área local 34, 707
 - de larga distancia 33, 707

- reglas de las operaciones relacionales 744-745
- reglas de negocios 640-641
- reglas de transformación
 - para cuantificadores 240
 - para operaciones relacionales 511-515
- reglas recursivas 734, 746
- reglas recursivas, evaluación de 748-751
 - conjunto mágico 750-751
 - simple, semisimple 750
- reglas seguras 742-744
- REHACER, entradas de bitácora tipo 583
- REHACER, operación 538, 540, 583
- reinicio cíclico 567, 569
- relación 141
 - anidada *véase* relaciones anidadas
 - atributos de *véase* atributo (relacional)
 - características de la 142-144
 - claves de *véase* clave
 - definición alternativa de 143
 - definición de (declaración) 153-155
 - ejemplar de 141
 - esquema de 141
 - estado de una 141
 - extensión de 141
 - grado de una *véase* grado de una relación
 - intensión de 141
 - privilegios de 603-604
 - semántica de una 397-400, 407
 - (*véase también* relación base, relación virtual)
 - relación base 218
 - relación dependiente 656-657
 - relación no en 1FN *véase* relación anidada
 - relación primaria 657
 - relación referida 657
 - relación universal 406, 428
 - relación virtual 218
 - (*véase también* vista (relacional))
 - relaciones anidadas 415, 652-656
 - rendimiento, vigilancia del *véase* vigilancia
 - REORG, utilería (DB2) 280
 - reorganización *véase* archivos, reorganización de
 - REPAIR, utilería (DB2) 280
 - reparto (en bases de datos distribuidas) 712-716
 - REPLACE (QUEL) 250
 - replicación, esquema de 712
 - representación conceptual 7
 - requerimientos, recolección/análisis de 39-41, 458-459
 - rerreferencia funcional 650-651
- partición de 723
- topología de 707
- transparencia de *véase* transparencia de distribución
 - controlada 12, 400-403
 - controlada 12-13
 - reescribir un bloque 83, 809
 - reescritura, tiempo de 809
- REFERENCES, privilegio (SQL) 604
- referencia inversa 51, 650-651, 694
- referencias lógicas 98
- registro(s) 5, 75-80, 293-294
 - apuntador a 109-114
 - bloques de (en disco) 79-80
 - formato de 76
 - identificador de (ID) 279-280
 - tipo de 76, 293-294, 307-311
 - (*véase también* longitud fija, registro de; longitud variable, registro de)
- registro actual 81
- registro almacenado (DB2) 279
- registro ancla 106
 - (*véase también* bloque, ancla de)
- registro de intervenciones 602
- registro fantasma 576
- registro ficticio, tipo de 298, 301
- registro físico de bases de datos (fms) 380
- registro hijo 349-351
- registro miembro 295
- registro miembro, tipo de 294
- registro por registro, DML de 29
- registro raíz 350
- registro raíz, tipo de 350
- registro virtual 356
- registros extendidos, organización de 79
- registros no extendidos, organización de 79
- regla 732-751, 784
 - cabeza de (LHS) 732
 - conclusión de (LHS) 732
 - corpo de (RHS) 732
 - interpretación de 737-739
 - premisa de (RHS) 732
 - reescritura de 750-751
 - seguridad de 742-744
 - regla rectificada 747
- reglas de inferencia completas para DF 410
- DF y DMV 442-443
- reglas de inferencia para dependencias *véase* dependencias funcionales, dependencias multivaluadas

- reserva virtual de memoria intermedia (DB2) 287
- resolución de colisiones 90
- respaldo 75
 - copia de 724
 - sitio de 724
 - utilería de 33
- (*véase también* base de datos, respaldo de)
- restricción 640-646
 - (*véase también* integridad, restricciones de)
- restricciones estructurales 53, 643
- restricciones, tipos de
 - codificadas 643
 - de clave 642-643
 - de dominio 640-641
 - de estado 644-645
 - de transición 645-646
 - de vínculo 642-643
 - declarativas 644
 - explícitas 644
 - implícitas 644
 - inherentes 644
 - por procedimientos 643-644
- RESTRICT (SQL) 195-196
- resumen incorrecto, problema de 536
- RETAINING CURRENCY (DML de red) 332-333
- retardo rotacional 73, 808-809
- retícula de clases *véase* especialización, retícula de; generalización, retícula de
- RETRIEVE (QUEL) 245
- reunión 163-166, 436-437
 - atributos de 165
 - condición de 165-166, 178-179, 198, 239, 248
 - bidireccional 500
 - dependencia de *véase* ID
 - factor de selección de la 502, 526
 - funciones de costo de la 525-527
 - implementación de la 500-506
 - multidireccional 500
 - selectividad de 166, 526
 - sin pérdidas *véase* reunión sin pérdidas
- REUNIÓN, operación 163-166
- reunión, tipos de *véase* equirreunión, reunión natural, reunión externa, semirreunión, reunión theta
- REUNIÓN CRUZADA, operación 164, 166
- reunión de ciclo anidado 500
- reunión de ciclo interno-externo *véase* reunión de ciclo anidado
- reunión de ordenamiento-combinación 501-502
- REUNIÓN EXTERNA, operación 170-172, 436
 - en SQL 208
- REUNIÓN EXTERNA COMPLETA, operación 171
- REUNIÓN EXTERNA DRECHA, operación 171
 - en SQL 208
- REUNIÓN EXTERNA IZQUIERDA, operación 171
 - en SQL 208
- REUNIÓN NATURAL, operación 165-166
- reunión por dispersión 501-506
- reunión por dispersión híbrida 505
- reunión sin pérdidas (no aditiva) 405, 413-414
- descomposición de 434-436
- pauta informal para 403-405
- propiedad de 431-434
- prueba de 431-432
- REUNIÓN THETA, operación 165
- reversión
 - de transacción *véase* transacción, reversión de
 - reversión de
 - en cascada 569, 583-585
 - revisor 495
 - revocación de un privilegio *véase* privilegios
 - REVOKE (SQL) 604, 606
 - en DB2 281-282
 - RIM 263
 - ROLLBACK (DB2) 284-285
 - ROSE, sistema 774
 - RUNSTATS, utilería (DB2) 281
 - SDM *véase* modelo de datos semántico
 - secuencia, clave de 385
 - segmento de archivo 80
 - segunda forma normal *véase* forma normal
 - seguridad 13, 599-611
 - clases de 607-608
 - clasificaciones de 607-608
 - en bases de datos estadísticas 610-611
 - kernel de 610
 - mediante vistas 604
 - niveles de 610
 - papel del DBA en la 601
 - subsistema de 13
 - tipos de 599-601
 - (*véase también* autorización; acceso, control de)
 - selección, condición de
 - en archivos 83-84
 - en el modelo relacional 156-158, 239, 251-252
 - en SQL 198
 - selección-proyección-reunión, consulta de 186, 245

- SELECCIONAR, operación 156-158
- funciones de costo de la 524
- implementación de la 497-500
- SELECT (SQL) 195-215
- SELECT, privilegio 604
- selectividad
 - de REUNIR 166, 526
 - de SELECCIONAR 157, 499
- self (o2) 687
- semántica de los datos 15, 397-400
- semántica de referencia 680
- seminreunión 719-720
- separación entre bloques 70
- separador, carácter 78-79
- SEQUENC 188
- seriabilidad 545-554
 - aplicaciones de la 550-553
 - prueba de 548-550
 - (*véase también* plan)
- seriabilidad de conflictos 545-550, 570
- servidor 33, 707
- SET DEFAULT (SQL) 192, 193
- SET NULL (SQL) 192, 193
- SET SELECTION BY (DML de red) 311-312, 328-329
- seudónimos (SQL) 199
- SGBD 2, 3
 - arquitectura de los 25-26
 - catálogo de 485-493
 - centralizado 34
 - clasificación de los 34-35
 - costo de adquisición de un 34
 - diseñadores/implementadores de 11
 - distribuidos *véase* SGBCD
 - elección de un 468-470
 - lenguajes de 28-30
 - módulos de 30-31, 491-493
 - ventajas de los 11-16
- SGBD activo 782-783
- SGBD de propósito especial 34
- SGBD de propósito general 34
- SGBD extensible 795-797
- SGBD multiusuario 34, 532
- SGBD relacional 263
- catálogo de 485-489
- criterios de 288-29
- ejemplo de sistema *véase* DB2
- SGBCD 34, 704-707
- federados 34, 710
- heterogéneos 34, 716, 770
- homogéneos 34, 716
- SGBCD 13, 684-698
 - o2 684-692
 - ObjectStore 692-698
 - SHARE (DB2) 284-285
 - signatura 7, 763
 - SIMULA 665
 - sincronía, punto de 285
 - síntesis relacional 430
 - sistema
 - administrador del, *véase* DBA
 - analista de 11
 - ciclo de vida del *véase* ciclo de vida
 - cuenta del 601
 - entorno del 11
 - sistema de base de datos centralizado 34, 704
 - (*véase también* SGBCD)
 - sistema de base de datos experto 730
 - sistema de información
 - ciclo de vida del 454
 - papel en las organizaciones del 452-454
 - sistema monousuario 34, 532
 - sistema operativo 11, 32-33
 - sistemas abiertos 770
 - sistemas de base de datos 3
 - arquitectura de los *véase* tres esquemas, arquitectura de
 - breve historia de los 20-21
 - características de los 5-9
 - ciclo de vida de los 454-455
 - desventajas de los 16
 - entorno de los 31-33
 - implementación de 472
 - utilerías de *véase* utilerías
 - sistemas relacionales extensibles 35
 - sitio 34, 706
 - autonomía del *véase* autonomía local
 - sitio del resultado 717-718
 - sitio primario 723
 - SMALLTALK 14, 665
 - software privilegiado 13
 - sombras, creación de 582
 - SOME (SQL) 202
 - SORT BY (QUEL) 248-249
 - SQL dinámico 285-286
 - SQL incorporado 225-228
 - SQL, lenguaje 188-228, 768
 - autorización en 603-607
 - concepto de catálogo en 189-190
 - concepto de esquema en 189-190
 - consultas en 202-215
 - cuantificadores en 243-244
 - DDL (definición de datos) 189-195

incorporado 225-228
 instrucciones de alteración en 194-195
 omisión de instrucciones 194
 operaciones de actualización en 215-217
 privilegios en 603-607
 restricciones en 193-195, 222-223
 tipos de datos en 190-192, 214
 vistas en 218-222
 SQL Server 263
 SQL-92 189
 SQL/DS 225, 262-263
 SQL1 189
 SQL2 188-228
 SQL3 189, 701
 SQLCA (SQL Communication Area) 275
 SQLCODE 227, 284
 SQLERROR 284
 SQLSTATE 227
 STARBURST, proyecto 785
 STEP (norma para intercambio de datos de productos) 774
 STORE (DML de red) 328-330
 STOSPACE, utilidad (DB2) 280-281
 subárbol 118
 subcadena, comparación de (SQL) 213
 subclase 615-628, 659
 compartida 623-633
 definida por predicados 620
 en EER 615-628
 en OO 678-679
 restricción de 658-659
 (véase también generalización; especialización; superclase)
 subesquema (IDMS)
 definición de 338
 en DDL 335
 sublenguaje de datos 29
 subobjetivo 741
 subtipo
 en EER véase subclase
 en OO 677-678
 subtipos, jerarquía de 720
 suceso, clase de 658-659
 suceso-condición-acción, regla de 781-783
 sucesos, tipos de 784
 SUM (QUEL) 247
 SUM (SQL) 208-209
 SUM. (QBE) 256
 SUMU (QUEL) 247
 superclase/subclase, jerarquía/réticula de
 en EER 615-628

tipo enumerado 641
 tipos atómicos 684
 tipos de datos carácter 641
 en SQL 190
 tipos de datos masivos (OO) 669
 tipos de datos numéricos 641
 en SQL 190
 tipos extensibles, sistema de 680
 TODOS, sistema 776
 topología véase red, topología de
 TOTAL (SQBD) 292
 transacción (transacciones)
 abortar 538-540, 583
 asentamiento de véase bitácora
 conceptos de 531-554
 confirmación de 538-542
 definida por el usuario 39
 diseño de 465-468
 especificación de 647-649
 estados de 538-539
 fallo de 544
 historia de véase plan
 identificador de 539
 marca de tiempo de 539
 operaciones de 533, 538
 plan de véase plan
 procesamiento de 8, 531-542, 769
 propiedades ACID de 541-542
 reversión de 583-585
 sólo de lectura 538
 (véase también control de concurrencia; bloqueo mortal; recuperación)
 transacción abortada 538-539, 583
 transacción activa 583
 transacción confirmada 539, 583
 transacción fallida 539
 transacción larga 773-774
 transacción local 716
 transacción programada 10, 30
 transferencia, costo de (en bases de datos distribuidas) 717-718
 transferencia, velocidad de 807-809
 transformación 143, 470-475
 (véase también esquema ER)
 transición, restricción de véase restricción
 transmisión, costo de véase transferencia, costo de
 transparencia de distribución 709, 716
 transparencia de ubicación véase transparencia de distribución
 traslape, restricción del, (especialización) 621
 tres esquemas, arquitectura de 26-27

TSO (Opción de compartimiento de tiempo) 263-265
 tupla (relacional) 140-144
 variable 235-236
 tupla de referencia 150
 tuplas colgantes 436
 tuplas espurias 403-405
 tuplas, cálculo relacional de 235-243

U. (QBE) 257-258
 UFI, véase interfaz amable con el usuario
 unidad de ejecución 317-320
 UNIFY 228, 262-263, 769
 UNION (SQL) 201
 UNIÓN EXTERNA, operación 172, 633
 UNIÓN, operación 161-162
 UNIQUE (QUEL) 245, 246-247
 UNIQUE (SQL) 191, 193, 205, 224
 UNISQL 763, 796-797
 Universo de Discurso (UoD) 2, 640
 uno a muchos (1:N), vínculo 51, 294
 uno a uno (1:1), vínculo 51, 52, 300-301
 (véase también muchos a muchos)
 UPDATE (SQL) 217-218
 USER (DB2) 281-282
 usuario 8-11
 área de trabajo de véase UWA
 cuenta de 602
 identificación de 602
 interfaz de 14, 30-31, 766-767, 800
 vista de 8, 25
 usuario autónomo 10
 usuario avanzado 10
 usuario esporádico 10
 usuario final 10
 usuario global 705
 usuario local 705
 usuario paramétrico 10, 30
 usuario simple 10, 29
 utilerías 32
 en DB2 281
 UWA 317, 366

validación, fase de 573
 validación, protocolos de 573-574
 valor 42, 45
 en O2 684-685
 valor atómico 140, 144, 414, 668
 valor de verdad (de un átomo) 237
 valor nulo 44, 141, 144, 403, 437

- variable
 - en el área de trabajo del usuario véase UWA
 - en el modelo OO 673
- variable libre 237-238
- variable ligada 237-238, 740
- variable limitada 744
- variable persistente 698
- VAX-DBMS 333
- vector (modelo de red) 293
- velocidad de transferencia masiva 73, 809
- versiones, creación de 683-684
- VIA INDEX (IDMS) 336
- VIA, modo de localización de conjuntos (IDMS) 336
- víctimas, selección de 567
- vigilancia 33
- vínculo 15, 23, 48-54, 629
 - atributos de 53-54
 - conjunto de 48
 - ejemplar de 48
 - relación de 181
 - restricciones de 51-53, 642 (véase también cardinalidad, razón de; grado; existencia, dependencia de; mín-máx, restricción; participación, restricción de)
 - tipo de 48
 - vs. atributo 48, 50-57
- vínculo binario 49
- vínculo ES-UN 615, 628
- vínculo específico 616
- vínculo identificador 54-55
- vínculo n-ario 59-62
- vínculo recursivo 50-51
- vínculo ternario 49, 59-62
- vínculo padre-hijo virtual véase VPHV
- vínculos inversos (en ObjectStore) 694
- violación de la integridad, acciones por 152-153
 - cascada 152-153
 - colocar nulos 153
 - rechazar 152-153
 - vista(s) (relacional(es))
 - actualización de 220-221
 - como mecanismo de seguridad 604
 - definición de 219
 - en DB2 281-282
 - en SQL 218-222
 - implementación de 221-222
 - vistas, integración de 461-465
 - (véase también usuario, vista de)
 - vistas, seriability de 553-554, 570-571
 - VM/CMS, sistema operativo 262-263
 - votación 725
 - voz, reconocimiento de 767-768
 - VPH 349-350
 - ocurrencia de 349
 - virtual véase VPHV
 - tipo de 349
 - VPHV 355-358
 - VSAM 133, 390
 - WHERE (QUEL) 245
 - WHERE (SQL) 196
 - WHERE CURRENT OF (SQL) 228
 - WISS (Sistema de almacenamiento Wisconsin) 692
 - WITH CHECK OPTION (SQL) 221
 - WITHIN SET (DML de red) 324-325
 - WORM 766
 - XDB 263

Vocabulario técnico bilingüe

- abanico
 - currency indicators
- abortar
 - actualización diferida, recuperación en deferred update recovery
- abrir (orden de acceso a archivos)
 - open
- abstracción de los datos
 - data abstraction
- acceso, camino de
 - access path
- acceso, control de
 - access control
- acceso, estructura de
 - access structure
- acceso, método de
 - access method
- actual de jerarquía
 - current of hierarchy
- actual de tipo de conjuntos
 - current of set type
- actual de tipo de registros
 - current of record type
- actual de unidad de ejecución
 - current of run unit
- actualidad, indicadores de,
 - currency indicators
- actualización diferida, recuperación en
 - deferred update recovery
- actualización en el lugar
 - in-place updating
- actualización inmediata, recuperación en
 - immediate update recovery
- actualización perdida, problema de
 - lost update problem
- actualización temporal, problema de
 - temporary update problem
- actualización, anomalías de
 - update anomalies
- actualizar
 - update
- administrador de bases de datos
 - database administrator
- agregación
 - aggregation
- agrupación
 - grouping
- agrupación, atributos de
 - grouping attributes

agrupamiento
clustering

aislamiento, nivel de
isolation level

alcance (de archivo)
extent

aleatorización, función de
randomizing function

álgebra relacional
relational algebra

almacenamiento
storage

almacenamiento en línea
on-line storage

almacenamiento fuera de línea
off-line storage

almacenamiento intermedio
buffering

almacenamiento secundario
secondary storage

análisis de datos
data analysis

analizador sintáctico
parser

anidación de atributos
nesting attribute

ANIDAR, operación
NEST operation

anillo, estructura de datos de
ring data structure

anotaciones
annotations

aplicaciones, creación de
application development

aplicaciones, procesamiento de
application processing

apuntador a datos
data pointer

apuntador al anterior
prior pointer

apuntador al hijo
child pointer

apuntador al siguiente
next pointer

árbol B
B-tree

árbol cuadrático
quad tree

árbol de ocurrencias
occurrence tree

árbol equilibrado
balanced tree

atributo compuesto
composite attribute

atributo de referencia (oo)
referencing attribute

atributo derivado
derived attribute

atributo específico
specific attribute

atributo clave
key attribute

atributo multivaluado
multivalued attribute

atributo no primo
nonprime attribute

atributo primo
prime attribute

atributo virtual
virtual attribute

aumento, regla de
augmentation rule

autonomía local
local autonomy

axiomas base
ground axioms

axiomas deductivos
deductive axioms

base de cómputo de confianza
trusted computing base

base de conocimientos, sistemas de gestión de
knowledge base management systems

base de datos
database

base de datos lógica
logic database

base de datos deductiva
deductive database

base de datos distribuida replicada
replicated distributed database

base de datos local
local database

base de datos lógica (lms)
logical database

bases de datos activas
active databases

bases de datos científicas
scientific databases

bases de datos distribuidas
distributed databases

bases de datos estadísticas/científicas
statistical/scientific databases

bases de datos federadas
federated databases

bases de datos geográficas
geographic databases

base de datos temporales
temporal databases

basura, recolección de
garbage collection

bit de modificación
dirty bit

bitácara
log

bloqueo (disco)
block

bloquear elemento
lock_item

bloqueo
locking

bloqueo de dos fases básico
basic two-phase locking

bloqueo de dos fases conservador
conservative two-phase locking

bloqueo de dos fases estricto
strict two-phase locking

bloqueo mortal
deadlock

bloqueo escritura, operación
write_lock operation

bloqueo lectura, operación de
read_lock operation

bolsa
bag

buscar (orden de acceso a archivos)
find

buscar siguiente (orden de acceso a archivos)
findnext

búsqueda
search

búsqueda binaria
binary search

búsqueda lineal
linear search

búsqueda primero en amplitud
breadth-first search

búsqueda primero en profundidad
depth-first search

búsqueda, tiempo de
seek time

cabecera (descriptor) de archivo file header	catálogo catalog
cabeza de lectura/escritura read/write head	categoría total total category
cabeza de regla (IHS) rule head	cerradura closure
cabeza fija, disco de fixed-head disk	cerrar (orden de acceso a archivos) close
cabeza móvil, disco de movable head disk	certificación, protocolos de certification protocols
caché, memoria cache memory	ciclo de vida life cycle
cadena, tipo de datos string data type	cifrado encryption
caída, recuperación de una crash recovery	cilindro (disco) cylinder
cálculo de predicados predicate calculus	cinta magnética magnetic tape
cálculo de predicados de primer orden first order predicate calculus	clase agregada aggregate class
cálculo relacional relational calculus	clase base base class
cambio de nombre renaming	clase de objetos abstractos abstraction class
camino, expresión de path expression	clase de objetos concretos concrete class
campo almacenado (DB2) stored field	clase derivada derived class
campo de referencia referencing field	clase raíz root class
campo clave de un archivo key field of a file	clave a dirección, transformación de key-to-address transformation
campo repetitivo repeating field	clave aparente apparent key
canal furtivo covert channel	clave candidata candidate key
candado lock	clave de ordenación, campo de ordering key field
candado binario binary lock	clave externa foreign key
candado compartido shared lock	clave primaria primary key
candado de certificación certify lock	clave secundaria secondary key
candado exclusivo exclusive lock	clave sustituta surrogate key
cardinalidad, razón de cardinality ratio	clave, atributo key attribute
carga de datos data loading	clave, restricciones de key constraints
carga, utilidad de load utility	cliente-servidor, arquitectura client-server architecture

conjunto valor value set	cobertura cover
conocimientos, representación de (KR) knowledge representation	cociente quotient
conservación de la consistencia, propiedad de consistency preservation property	coincidencia parcial, comparación de partial match comparison
consistencia de los datos data consistency	colisión (dispersión) collision
constructor de arreglos (oo) array constructor	compartimiento de los datos data sharing
constructor de átomos (oo) atom constructor	compatibilidad de unión union compatibility
constructor de conjuntos (oo) set constructor	compilación compilation
constructor de tuplas (oo) tuple constructor	compilador compiler
constructor, operación (oo) constructor operation	compleción relacional relational completeness
consulta anidada (SQL) nested query	composición de funciones function composition
consulta anidada correlacionada (SQL) correlated nested query	comunicaciones de datos data communications
consulta estadística statistical query	condición, cuadro de (QBE) condition box
consulta recursiva recursive query	conexión de referencia reference connection
control de acceso discrecional discretionary access control	configuración configuration
control de acceso obligatorio mandatory access control	confirmar (confirmación) commit
control de concurrencia distribuido distributed concurrency control	conjunto acoplado al propietario owner-coupled set
control de concurrencia optimista optimistic concurrency control	conjunto constante (SQL) constant set
conversión, herramientas de conversion tools	conjunto de escritura write set
copia primaria primary copy	conjunto de lectura read set
costo de acceso access cost	conjunto explícito (SQL) explicit set
costo de transferencia de datos data transfer cost	conjunto mágico magic set
costo, estimación de cost estimation	conjunto multimiembro multimember set
creación de ejemplares instantiation	conjunto potencia power set
creación de múltiples ejemplares polyinstantiation	conjunto propiedad del sistema system-owned set
cuantificador existencial existential quantifier	conjunto singular singular set

cuantificador universal
 universal quantifier
cuarta generación, lenguaje de (4GL)
 fourth generation language
cubeta (de dispersión)
 bucket
cuenta privilegiada
 privileged account
cuerpo de regla (RHS)
 rule body

datos espaciales, gestión de
 spatial data management
datos replicados
 replicated data
datos virtuales
 virtual data
definición de datos
 data definition
degradar un bloque
 downgrading a lock
dependencia de los datos
 data dependency
dependencia funcional completa
 full functional dependency
dependencia multivaluada
 multivalued dependency
dependencia no transitiva
 nontransitive dependency
dependencia transitiva
 transitive dependency
dependencias funcionales (DF)
 functional dependencies
desalajo de la memoria intermedia
 buffer flushing
DESANDAR, operación
 UNNEST operation
desbloquear, operación
 unlock operation
desborde, archivo de
 overflow file
descendente
 top-down
descomposición de consultas
 decomposition of queries
descomposición de relaciones
 decomposition of relations
descomposición relacional
 relational decomposition
DESHACER, entradas de bitácora tipo
 UNDO-type log entries

DESHACER, operación
 UNDO operation
DESHACER/NO REHACER, protocolo
 UNDO/NO REDO protocol
DESHACER/REHACER, protocolo
 UNDO/REDO protocol
desnormalización
 denormalization
destructor, operador (OO)
 destructor operator
DF
 FD
diagrama de estructura de datos
 data structure diagram
diagrama ER
 ER diagram
diario
 journal
diccionario de datos activo
 active data dictionary
diccionario de datos, sistemas de
 data dictionary systems
DIFERENCIA, operación
 DIFFERENCE operation
diferencia de impedancia
 impedance mismatch
dirección
 address
direccionamiento abierto
 open addressing
DIRECTO, modo de localización (modelo de red)
 DIRECT location mode
directorio de datos
 data directory
disco (magnético)
 disk
disco óptico de una sola escritura
 write-once optical disk
disco óptico, almacenamiento en
 optical disk storage
diseño conceptual
 conceptual design
diseño conceptual descendente
 top-down conceptual design
diseño de bases de datos
 database design
diseño lógico de bases de datos
 logical database design
diseño, bases de datos para
 design databases
disparador
 trigger

dispersión
 hashing
distribución de los datos
 data distribution
disyunción, restricción de
 disjointness constraint
DML no por procedimientos
 nonprocedural DML
DML orientado a conjuntos
 set oriented DML
DML por procedimientos
 procedural DML
DMV
 MDV
documentos, gestión de
 document management
dominio atómico
 atomic domain
dominio/clave, forma normal de
 domain/key normal form
dominio simple
 simple domain
dominio, variable de
 domain variable
dominios, cálculo relacional de
 domain relational calculus
dos fases, protocolo de confirmación de
 two-phase commit protocol
dos fases, protocolos de bloqueo de
 two-phase locking protocols
duplicación de información
 duplication of information
durabilidad, propiedad de
 durability property

ejecución concurrente
 concurrent execution
ejecución perezosa
 lazy execution
ejecución segmentada
 pipelined execution
ejemplar
 instance
ejemplo, valor de (QBE)
 example value
ejemplo, variable de (QBE)
 example variable
elemento de información
 data element, data item
eliminación
 deletion

Eliminar (modelo relacional)
 Delete
eliminar (orden de acceso a archivos)
 delete
encadenamiento
 chaining
encadenamiento hacia adelante
 forward chaining
encadenamiento hacia atrás
 backward chaining
encapsulamiento
 encapsulation
enfoque orientado a objetos
 object-oriented approach
enlace
 binding
enlace automático (DB2)
 automatic binding
enlace tardío
 late binding
enlace temprano
 early binding
enlace, tipo de registro de
 linking record type
enlazador (DB2)
 bind
entero, tipo de datos (SQL)
 integer data type
entidad-interrelación
 entity-relationship
entidad-vínculo
 entity-relationship
entidades débiles, tipo de
 weak entity type
EQUIRREUNIÓN, operación
 EQUIJOIN operation
equivalencia de conflictos
 conflict equivalence
ER extendido, modelo
 enhanced-ER model
escribir elemento, operación
 write_item operation
escritura anticipada, protocolo de bitácora de
 write-ahead log protocol
escritura ciega
 blind write
escritura irrestricta
 unconstrained write
escritura, fase de (de una transacción)
 write phase

espacio de índice (DB2)	factor de bloques
indexspace	blocking factor
espacio de tabla (DB2)	factor de carga (archivo disperso)
tablespace	load factor
especialización total	fase de lectura
total specialization	read phase
espera cautelosa	fecha, tipo de datos
cautious waiting	date data type
espera indefinida	fiabilidad
livelock	reliability
esquema conceptual	FJA, retención de conjuntos (modelo de red)
conceptual schema	FIXED set retention
esquema externo	fila
external schema	row
esquema interno	filtrado
internal schema	filtering
estado consistente	forma clausal
consistent state	clausal form
estado de relación permitido (ejemplar)	formas normales (de las relaciones)
legal relation state	normal forms
estado, indicador de	formas, interfaz de
status indicator	forms interface
estado, restricción de	fórmula atómica
state constraint	atomic formula
estado válido	fórmula bien formada (bfb)
valid state	well-formed formula
estrella, propiedad de	fragmentación completa
star property	complete fragmentation
ES-UN, vínculo	fragmento
ISA relationship	fragment
examen de archivo	fragmento mixto
file scan	mixed fragment
excepción, objetos de	fragmento vertical
exception objects	vertical fragment
exhibición	función derivada
display	derived function
existencia, dependencia de	función integrada
existence dependency	built-in function
explosión de componentes	función inversa
parts explosion	inverse function
exportación, esquema de	funciones agregadas
export schema	aggregate functions
exportación, recurso de (O2)	 fusión (combinación) de versiones
export facility	merging of versions
expresión segura del cálculo relacional	generalización
safe relational calculus expression	generalization
extensión	generalización total
extension	total generalization
extracción de información	genoma, bases de datos de
information retrieval	genome databases

gestor de datos almacenados	igualdad de objetos
stored data manager	equality of objects
grafo acíclico	imagen
acyclic graph	image
grafo de espera	imagen "antes" (AFIM)
wait-for graph	before image
grafo de precedencia	imagen "después" (AFIM)
precedence graph	after image
grafo de seriación	implementación, modelo de datos de
serialization graph	implementation data model
grafo de versiones	impulsor
version graph	actuator
granularidad de los elementos de información	inanición
granularity of data items	starvation
grupo (de bloques en disco)	inclusión, dependencia de
cluster	inclusion dependency
grupo repetitivo	independencia de los datos
repeating group	data independence
guardia, condición de	independencia de programas y datos
guard condition	program-data independence
hecho	independencia de programas y operaciones
fact	program-operation independence
herencia múltiple	índice denso
multiple inheritance	dense index
herencia selectiva	índice dinámico multinivel
selective inheritance	dynamic multilevel index
herramientas de diseño	índice lógico
design tools	logical index
hijo virtual	índice no denso
virtual child	nondense index
hiperespacio (DB2)	índice secundario
hyperspace	secondary index
hiperfondo (DB2)	indización
hyperpool	indexing
hipermedia	inferencia estadística
hypermedia	statistical inference
hoja, tipo de registros	inferencia, máquina de
leaf record type	inference engine
hojear	ingeniería concurrente
browsing	concurrent engineering
hora, tipo de datos	ingeniería de software
time data type	software engineering
Horn, cláusulas de	iniciar transacción
Horn clauses	begin transaction
idempotencia de la recuperación	inserción
idempotence of recovery	insertion
identidad, conexión de	insertar (orden de acceso a archivos)
identity connection	insert
	integración
	integration
	integridad, aserción de
	integrity assertion

integridad, restricciones de integrity constraints	lenguaje relacionalmente completo relationally complete language
integridad referencial referential integrity	léxico lexicon
integridad semántica, restricción de semantic integrity constraint	lista enlazada circular circular linked list
inteligencia artificial artificial intelligence	listas, constructor de (oo) list constructor
intensión intension	longitud fija, registro de fixed-length record
intercalación de operaciones interleaving of operations	longitud variable, registro de variable-length record
interfaz amable con el usuario user-friendly interface	MANUAL, conjunto (modelo de red) MANUAL set
interfaz gráfica graphical interface	máquina dorsal backend machine
INTERSECCIÓN, operación INTERSECTION operation	máquina frontal frontend machine
intervalo range	marca de tiempo timestamp
intervalo, tipo de datos de (sql) interval data type	marca de tiempo de escritura write timestamp
jerarquía hierarchy	marca de tiempo de lectura read timestamp
latencia latency	masivos, tipos de datos (oo) bulk data types
lectura no repetible, problema de unrepeatable read problem	matriz de acceso access matrix
lectura sucia dirty read	mecanismos de inferencia inference mechanisms
leer (orden de acceso a archivos) read	memoria intermedia buffer
leer_elemento, operación read item operation	memoria principal main memory
lenguaje anfitrión host language	memoria volátil volatile storage
lenguaje de consulta funcional functional query language	mensaje message
lenguaje de definición de datos data definition language	metacalse meta-class
lenguaje de especificación de aserciones assertion specification language	metadatos meta-data
lenguaje de manipulación de datos data manipulation language	método method
lenguaje de programación programming language	minimundo miniworld
lenguaje declarativo declarative language	modelado geométrico geometric modeling
lenguaje natural, interfaz de natural language interface	modelo de datos binario binary data model

nivel externo external level	modelo de datos de red network data model
nivel interno internal level	modelo de datos funcional functional data model
no esperar, protocolo de no waiting protocol	modelo de datos jerárquico hierarchical data model
NO-DESHACER/NO-REHACER, protocolo NO-UNDO/NO-REDO protocol	modelo de datos relacional relational data model
NO-DESHACER/REHACER, protocolo NO-UNDO/REDO protocol	modelo de datos semántico semantic data model
nodo descendiente descendant node	modelo estructural structural model
nodo hijo child node	modelo relacional anidado nested relational model
nodo hoja leaf node	modelos conceptuales de datos conceptual data models
nodo interno (en un árbol) internal node	modelos de datos data models
nombrar (en el modelo ER) naming	modelos de datos basados en registros record-based data models
nombre de papel inverso inverse role name	modelos de datos de representación representational data models
normalización, algoritmos de normalization algorithms	modificar (orden de acceso a archivos) modify
normalización, proceso de normalization process	MODO DE LOCALIZACIÓN (modelo de red) LOCATION MODE
notaciones diagramáticas diagrammatic notations	montículo, archivo de heap file
objetivo (consulta deductiva) goal	montón, archivo de pile file
objeto molecular molecular object	muchos a muchos (M:N), vínculo many to many relationship
objetos complejos complex objects	multibase de datos multidatabase
objetos idénticos identical objects	multiconjunto multiset
OBLIGATORIO, conjunto (modelo de red) MANDATORY set	multimedia, base de datos de multimedia database
obtener (orden de acceso a archivos) get	multinivel multilevel
ocurrencia occurrence	multiprogramación multiprogramming
oficina, sistema de información de (ois) office information system	multiversión (control de concurrencia) multiversion
OPCIONAL, retención de conjuntos (modelo de red) OPTIONAL set retention	n-tupla n-tuple
operación abstracta abstract operation	negación estratificada stratified negation
operaciones definidas por el usuario user-defined operations	nivel conceptual conceptual level

operaciones idempotentes
idempotent operations

operadores aritméticos
arithmetic operators

operadores, sobrecarga de
operator overloading

optimización heurística de consultas
heuristic query optimization

optimización semántica de consultas
semantic query optimization

optimización sistemática de consultas
systematic query optimization

orden
order

ordenación externa
external sorting

ordenación por fusión
merge-sort

ordenamiento básico por marca de tiempo
basic timestamp ordering

ordenamiento estricto por marca de tiempo
strict timestamp ordering

otorgar privilegios
granting of privileges

padre virtual
virtual parent

padre-hijo, vínculo
parent-child relationship

página
page

paginación de sombra
shadow paging

paquete de discos
disk pack

paralelo, procesamiento en
parallel processing

parte gobernante
ruling part

participación
participation

patrón, dependencia de
template dependency

PEFS (primero en entrar, primero en salir)
FIFO (first-in-first-out)

pista (disco)
track

plan de acceso
access plan

plan de aplicación
application plan

plan de ejecución (estrategia)
execution plan

plan de transacciones
schedule of transactions

plan estricto
strict schedule

plan recuperable
recoverable schedule

plan sin cascada
cascadeless schedule

planificación de disco
disk scheduling

plantilla de una relación (QBE)
template of a relation

polimorfismo de las operaciones
polymorphism of operations

por la teoría de demostraciones
proof-theoretic

por la teoría de modelos
model-theoretic

precompilador
precompiler

predicado, bloqueo de
predicate locking

predicado base (de ejemplares)
ground predicate

predicado definido por hechos
fact-defined predicate

predicado definido por reglas
rule-defined predicate

preorden, recorrido en
preorder traversal

preprocesador
preprocessor

presentación y exhibición
presentation and display

privilegio de acceso
access privilege

privilegios
privileges

procedimiento remoto, llamada a (RPC)
remote procedure call

procesador de aplicaciones (PA)
application processor

procesador de base de datos en tiempo de ejecución
runtime database processor

procesador de datos (DP)
data processor

procesamiento de transacciones en línea
on-line transaction processing

recuperación
recovery

red (comunicaciones)
network

red de área local
local area network

red dividida
partitioned red

redundancia
redundancy

reescribir un bloque
rewriting a block

reescritura, tiempo de
rewrite time

referencia inversa
inverse reference

referencias lógicas
logical references

registro actual
current record

registro almacenado (DB2)
stored record

registro ancla
anchor record

registro de intervenciones
audit trail

registro fantasma
phantom record

registro ficticio, tipo de
dummy record type

registro hijo
child record

registro miembro
member record

registro por registro, DML de
record-at-a-time DML

registro raíz
root record

registro virtual
virtual record

registros extendidos, organización de
spanned record organization

registros no extendidos, organización de
unspanned record organization

regla rectificada
rectified rule

reglas de inferencia para dependencias
inference rules for dependencies

reglas de negocios
business rules

reglas de transformación
transformation rules

PRODUCTO CARTESIANO, operación
CARTESIAN PRODUCT operation

PRODUCTO CRUZADO, operación
CROSS PRODUCT operation

profundidad (en dispersión extensible)
depth

profundidad global (en dispersión)
global depth

profundidad local (en dispersión)
local depth

programa de aplicación
application program

programador de aplicaciones
application programmer

promover un candado
upgrading a lock

propiedad
property

propiedad, conexión de
ownership connection

propiedad, semántica de
ownership semantics

propietario identificador
identifying owner

protección de los datos
data protection

protocolo de ordenamiento por marca de tiempo
timestamp ordering protocol

protocolo esperar-morir
wait-die protocol

protocolo herir-esperar
wound-wait protocol

proyección, atributos de
projection attributes

proyección confirmada
committed projection

proyección-reunión, forma normal de
project-join normal form

PROYECTAR, operación
PROJECT operation

punto de control
checkpoint

punto de entrada (modelo de red)
entry point

razonamiento, mecanismos de
reasoning mechanisms

recorrido (navegación)
navigation

recuperabilidad de planes
recoverability of schedules

reglas recursivas recursive rules	retícula de clases lattice of classes
reglas seguras safe rules	reunión join
REHACER, entradas de bitácora tipo REDO type log entries	REUNIÓN CRUZADA, operación CROSS JOIN operation
REHACER, operación REDO operation	reunión de ciclo anidado nested loop join
reinicio cíclico cyclic restart	reunión de ciclo interno-externo inner-outer loop join
relación base base relation	reunión de ordenamiento-combinación sort-merge join
relación dependiente nest relation	REUNIÓN EXTERNA COMPLETA, operación FULL OUTER JOIN operation
relación no en 1FN non-1NF relation	REUNIÓN EXTERNA DERECHA, operación RIGHT OUTER JOIN operation
relación referida referenced relation	REUNIÓN EXTERNA, operación OUTER JOIN operation
relación universal universal relation	REUNIÓN EXTERNA IZQUIERDA, operación LEFT OUTER JOIN operation
relación virtual virtual relation	REUNIÓN NATURAL, operación NATURAL JOIN operation
relaciones anidadas nested relations	reunión por dispersión hash join
relaciones reunidas (SQL) joined relations	reunión por dispersión híbrida hybrid hash join
rendimiento, vigilancia del performance monitoring	reunión sin pérdidas (no aditiva) lossless (non-additive) join
reorganización reorganization	REUNIÓN THETA, operación THETA JOIN operation
reparto (en bases de datos distribuidas) allocation	reversión rollback
replicación, esquema de replication schema	reversión en cascada cascading rollback
representación conceptual conceptual representation	revisor scanner
requerimientos, recolección/análisis de requirements collection/analysis	revocación de un privilegio revoking a privilege
rerreferencia funcional functional reference	secuencia, clave de sequence key
reserva virtual de memoria intermedia (DB2) virtual buffer pool	seguridad en bases de datos estadísticas statistical database security
resolución de colisiones collision resolution	selección, condición de selection condition
respaldo backup	SELECCIONAR, operación SELECT operation
restricciones estructurales structural constraints	selectividad selectivity
restricciones, tipos de constraint types	semántica de los datos data semantics
retardo rotacional rotational delay	

sistema operativo operating system	semántica de referencia reference semantics
sistemas abiertos open systems	semirreunión semi-join
sistemas de base de datos database systems	separación entre bloques inter-block gap
sistemas relacionales extensibles extendible relational systems	separador, carácter separator character
sitio site	seriability serializability
sitio de respaldo backup site	seriability de conflictos conflict serializability
sitio del resultado result site	seriable por conflictos conflict serializable
sitio primario primary site	seriable por vistas view serializable
sólo de lectura read-only	servidor server
sombras, creación de shadowing	seudónimos (SQL) aliasing
SQL dinámico dynamic SQL	SCBD DBMS
SQL incorporado embedded SQL	SCBD activo active DBMS
subárbol subtree	SCBD de propósito especial special purpose DBMS
subcadenas, comparación de (SQL) substring comparison	SCBD de propósito general general purpose DBMS
subclase subclass	SCBD extensible extensible DBMS
subesquema (DBMS) subscheme	SCBD multiusuario multiuser DBMS
sublenguaje de datos data sublanguage	SCBDD DBDBMS
subobjetivo subgoal	SCBDD heterogéneos heterogeneous DBDBMS
subtipo subtype	SCBDD homogéneos homogeneous DBDBMS
subtipos, jerarquía de subtype hierarchy	signatura signature
suceso, clase de event class	sincronía, punto de syncpoint
suceso-condición-acción, regla de event-condition-action rule	síntesis relacional relational synthesis
superclase/subclase, jerarquía/retícula superclass/subclass hierarchy/lattice	sistema de base de datos centralizado centralized database system
superclase/subclase, vínculo superclass/subclass relationship	sistema de base de datos experto expert database system
superclave superkey	sistema de información information system
supertipo (OO) supertype	sistema monousuario single-user system

supervisor durante la ejecución (DB2)	runtime supervisor	tipos de restricciones por procedimientos	procedural restriction types
suposición de escritura restringida	constrained write assumption	tipos extensibles, sistema de	extensible type system
tabla base	base table	topología	topology
tabla de páginas	page table	transacción (transacciones)	transaction
tabla de páginas sombra	shadow page table	transacción abortada	aborted transaction
tabla virtual	virtual table	transacción activa	active transaction
texto, extracción de	text retrieval	transacción confirmada	committed transaction
Thomas, regla de escritura de	Thomas' write rule	transacción fallida	failed transaction
tiempo predefinido	timeout	transacción larga	long transaction
tipo de conjuntos recursivos	recursive set type	transacción local	local transaction
tipo de datos	data type	transacción programada	scheduled transaction
tipo de datos abstractos	abstract data type	transacción cancelada	cancelled transaction
tipo de datos booleano	boolean data type	transferencia, costo de (en bases de datos distribuidas)	transfer cost
tipo de registro hijo	child record type	transferencia, velocidad de	transfer rate
tipo enumerado	enumerated type	transformación	mapping
tipos atómicos	atomic types	transformación de modelos de datos	data models mapping
tipos de datos de carácter	character data types	transición, restricción de	transition constraint
tipos de datos numéricos	numeric data types	transmisión, costo de	transmission cost
tipos de restricciones codificadas	coded restriction types	transparencia de distribución	distribution transparency
tipos de restricciones de dominio	domain restriction types	transparencia de ubicación	location transparency
tipos de restricciones de estado	state restriction types	traslapo, restricción de (especialización)	overlapping constraint
tipos de restricciones de clave	key restriction types	tres esquemas, arquitectura de	three-schema architecture
tipos de restricciones de transición	transition restriction types	tupla (relacional)	tuple
tipos de restricciones de vínculo	relationship restriction types	tupla de referencia	referencing tuple
tipos de restricciones declarativas	declarative restriction types	tuplas colgantes	dangling tuples
		tuplas espurias	spurious tuples

tuplas, cálculo relacional de	tuple relational calculus	variable libre	free variable
unidad de disco	disk drive	variable ligada	bound variable
unidad de ejecución	run unit	variable limitada	limited variable
UNIÓN EXTERNA, operación	OUTER UNION operation	vector (modelo de red)	vector
UNIÓN, operación	UNION operation	velocidad de transferencia masiva	bulk transfer rate
universo de discurso (UoD)	universe of discourse	versiones, creación de	versioning
uno a muchos (1:N), vínculo	one-to-many relationship	víctima, selección de	victim selection
uno a uno (1:1), vínculo	one-to-one relationship	vigilancia	monitoring
usuario autónomo	stand-alone user	vínculo	relationship
usuario avanzado	sophisticated user	vínculo binario	binary relationship
usuario esporádico	casual user	vínculo específico	specific relationship
usuario final	end user	vínculo identificador	identifying relationship
usuario global	global user	vínculo n-ario	n-ary relationship
usuario local	local user	vínculo padre-hijo virtual	virtual parent child relationship
usuario paramétrico	parametric user	vínculo recursivo	recursive relationship
usuario simple	naive user	vínculo ternario	ternary relationship
utilerías	utilities	vínculos inversos (en ObjectStore)	inverse relationships
validación, fase de	validation phase	violación de la integridad, acciones por	integrity violation actions
validación, protocolos de	validation protocols	vista (relacional)	view
valor atómico	atomic value	vistas, integración de	view integration
valor de verdad (de un átomo)	truth value	vistas, seriabilidad de	view serializability
valor nulo	null value	votación	voting
		voz, reconocimiento de	speech recognition

La ventaja más importante de este libro es su tratamiento coherente y balanceado de la tecnología de bases de datos. La segunda edición cubre temas de vanguardia como las bases de datos orientadas a objetos. Este libro es un recurso con el que todo profesional de las bases de datos debe contar.

Eugene Sheng
Western Illinois University

En lo que respecta a profundidad, amplitud y actualidad, esta obra no tiene igual. La nueva edición establece un justo medio entre la cobertura de las más modernas tecnologías de bases de datos y explicaciones claras de la teoría y el diseño, así como un tratamiento amplio de los modelos y los sistemas reales. En un reflejo de la preponderancia del modelo relacional de bases de datos en la práctica, este libro ofrece una amplia cobertura de las tecnologías relacionales, incluida la transformación ER a relacional, una descripción actualizada de SQL2 y un capítulo completo sobre DB2. Hay tres capítulos nuevos sobre aspectos prácticos de implementación, enfocándose en el procesamiento de transacciones, el control de concurrencia y las técnicas de recuperación. Además, se examinan los últimos avances en los campos de las bases de datos orientadas a objetos, la arquitectura cliente-servidor y las bases de datos deductivas. En un capítulo final se presentan las tendencias que están apareciendo en el área de las bases de datos.

Algunas características de esta segunda edición son:

- Se trata ampliamente el modelo relacional, incluidas las restricciones de integridad relacionales, las operaciones de actualización y el diseño de bases de datos relacionales.
- Se hace hincapié en el procesamiento de transacciones y las técnicas de implementación.
- Se describe la gestión de las bases de datos orientadas a objetos y se examinan los sistemas comerciales Q2 y ObjectStore.
- Se presentan las bases de datos deductivas, incluyendo ejemplos de LDL y otros sistemas deductivos.
- Se incluye la información más reciente sobre bases de datos distribuidas, considerando las arquitecturas cliente-servidor y la tendencia hacia los sistemas de múltiples bases de datos.
- Se reseñan las últimas investigaciones sobre bases de datos, destacando las bases de datos activas, temporales, espaciales, de multimedia y científicas.

OTRAS OBRAS DE INTERÉS PUBLICADAS POR ADDISON-WESLEY IBEROAMERICANA:

BARKER: *Metodología CASE* (60111)

BATINI, CERI y NAVATHE: *Discreto conceptual de bases de datos* (60120)

BERTINO y MARTINO: *Sistemas de bases de datos orientadas a objetos* (65371)

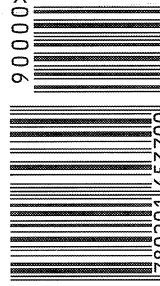
DATE: *Introducción a los sistemas de bases de datos, quinta edición* (51859)



Addison-Wesley Iberoamericana

Argentina • Chile • Colombia • España • Estados Unidos
México • Perú • Puerto Rico • Venezuela

9 0000>



9 780201 653700

ISBN 0-201-65370-2