INTRODUCCIÓN A LOS Sistemas de bases de datos

SÉPTIMA EDICIÓN

C. J. Date

TRADUCCIÓN:
I. Q. Sergio Luis María Ruiz Faudón
Ingeniero Químico, Analista de Sistemas
Sergio Kourchenko Barrena

REVISIÓN TÉCNICA:
Dr. Felipe López Gamino
Instituto Tecnológico Autónomo de México

Pearson Educación ®

MÉXICO • ARGENTINA • BRASIL • COLOMBIA • COSTA RICA • CHILE
ESPAÑA • GUATEMALA • PERÚ • PUERTO RICO • VENEZUELA

Contenido

CAPITULO 2

Arquitectura de los sistemas de bases de datos

2.1 INTRODUCCIÓN

Ahora estamos en condiciones de presentar la arquitectura para un sistema de base de datos. Nuestro objetivo al presentar esta arquitectura es ofrecer una infraestructura en la que puedan basarse los capítulos siguientes. Dicha infraestructura resulta útil para describir los conceptos generales de las bases de datos y para explicar la estructura de sistemas de bases de datos específicos; pero no afirmamos que todo sistema pueda coincidir enteramente con esta infraestructura en particular, ni queremos sugerir que esta arquitectura represente la única infraestructura posible. En particular, es probable que los sistemas "pequeños" (vea el capítulo 1) no manejen todos los aspectos de la arquitectura. Sin embargo, la arquitectura parece ajustarse bastante bien a la mayoría de los sistemas; es más, es prácticamente idéntica a la arquitectura propuesta por el Grupo de Estudio en Sistemas de Administración de Bases de Datos de ANSI/SPARC (la tan mencionada arquitectura ANSI/SPARC. Vea las referencias [2.1-2.2]). Sin embargo, nosotros decidimos no seguir la terminología ANSI/SPARC en todos sus detalles.

Nota: Este capítulo se asemeja al capítulo 1 en el sentido de que también es en cierto modo abstracto y árido, aunque es fundamental entender el material que contiene para una apreciación completa de la estructura y posibilidades de un sistema de base de datos moderno. Por lo tanto, al igual que en el capítulo 1, tal vez prefiera por ahora sólo darle una leída "ligera" y regresar a él más tarde, cuando sea directamente relevante para los temas que esté abordando.

2.2 LOS TRES NIVELES DE LA ARQUITECTURA

La arquitectura ANSI/SPARC se divide en tres niveles, conocidos como interno, conceptual y externo, respectivamente (vea la figura 2.1). Hablando en términos generales:

- El **nivel interno** (también conocido como el nivel *físico)* es el que está más cerca del al macenamiento físico; es decir, es el que tiene que ver con la forma en que los datos están almacenados físicamente.
- El **nivel externo** (también conocido como el nivel *lógico de usuario)* es el más próximo a los usuarios; es decir, el que tiene que ver con la forma en que los usuarios individuales ven los datos.
- El **nivel conceptual** (también conocido como el nivel *lógico de la comunidad,* o en oca siones sólo como el nivel *lógico,* sin calificar) es un nivel de indirección entre los otros dos.

Observe que el nivel externo tiene que ver con las percepciones de usuarios *individuales,* mientras que el nivel conceptual tiene que ver con la percepción de una *comunidad* de usuarios.

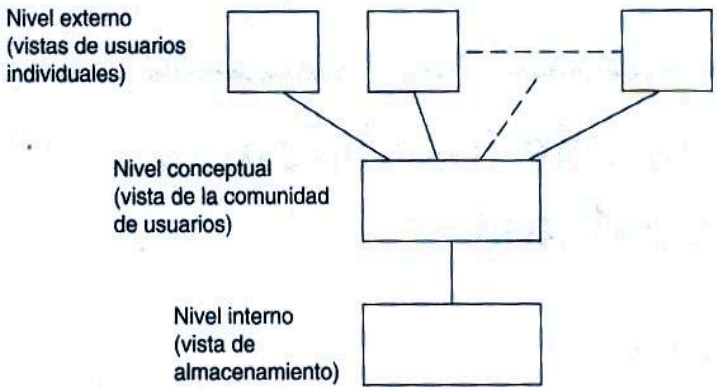

Figura 2.1 Los tres niveles de la arquitectura.

En otras palabras, habrá muchas "vistas externas" distintas, cada una consistente en una representación más o menos abstracta de alguna parte de la base de datos total, y habrá precisamente una "vista conceptual" que del mismo modo consiste en una representación abstracta de la base de datos en su totalidad.* (Recuerde que la mayoría de los usuarios no se interesarán en toda la base de datos, sino sólo en una parte limitada de la misma). En forma similar, habrá precisamente una "vista interna" que represente a la base de datos tal como está almacenada físicamente.

Un ejemplo hará más claras estas ideas. La figura 2.2 muestra la vista conceptual, la vista interna correspondiente y dos de las vistas externas (una para un usuario de PL/I y otra para un usuario de COBOL), todas ellas para una base de datos de personal sencilla. Por supuesto, el ejemplo es completamente hipotético —no pretende reflejar ningún sistema real— y hemos omitido deliberadamente muchos detalles irrelevantes. *Explicación*:

- En el nivel conceptual, la base de datos contiene información concerniente a un tipo de entidad denominada EMPLEADO. Cada empleado individual tiene un NUMERO_EMPLEADO (de seis caracteres), un NUMERO_DEPARTAMENTO (de cuatro caracteres) y un SALARIO (de cinco dígitos decimales).
- En el nivel interno, los empleados están representados por un tipo de registro denominado EMP_ALMACENADO, de veinte bytes de longitud. EMP_ALMACENADO contiene cuatro campos almacenados: un prefijo de seis bytes (que presumiblemente contiene información de control como los indicadores o los apuntadores) y tres campos de datos correspondientes a las tres propiedades de los empleados. Además, los registros de EMP_ALMACENADO están indexados sobre el campo EMP# por medio de un índice de nombre EMPX, cuya definición no se muestra.
- El usuario de PL/I tiene una vista externa de la base de datos en la que cada empleado está representado por un registro PL/I que contiene dos campos (los números de departamento

*Aquí, por *abstracto* simplemente queremos decir que la representación en cuestión comprende construcciones como los registros y campos que están más orientadas a los usuarios, a diferencia de las construcciones como los bits o los bytes que están más orientadas a la máquina.

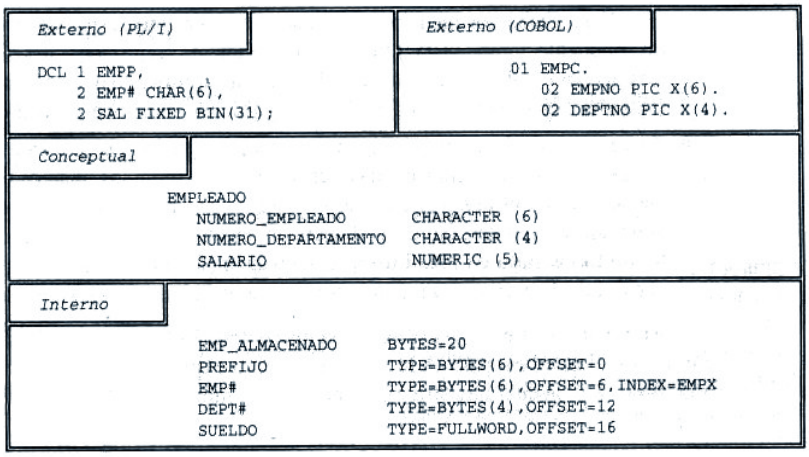

Figura 2.2 Un ejemplo de los tres niveles.

no son de interés para este usuario y por lo tanto fueron omitidos). El tipo del registro está definido por una declaración de estructura PL/I ordinaria según las reglas normales de PL/I.

- En forma similar, el usuario de COBOL tiene una vista externa en la que cada empleado está representado por un registro de COBOL, que una vez más contiene dos campos (esta vez fueron omitidos los salarios). El tipo de registro está definido por una descripción ordinaria de registro COBOL, de acuerdo con las reglas normales de COBOL.

Observe que los elementos correspondientes de los datos pueden tener diferentes nombres en distintos puntos. Por ejemplo, el número de empleado se denomina EMP# en la vista externa de PL/I, EMPNO en la vista externa de COBOL, NUMERO_EMPLEADO en la vista conceptual y nuevamente EMP# en la vista interna. Desde luego, el sistema debe estar al tanto de las correspondencias; por ejemplo, debemos indicar que el campo EMPNO de COBOL se deriva del campo conceptual NUMERO_EMPLEADO, el cual se deriva a su vez del campo almacenado EMP# al nivel interno. Dichas correspondencias, o **transformaciones**, no se muestran de manera explícita en la figura 2.2; para una explicación más amplia, consulte la sección 2.6.

Ahora bien, para los fines del presente capítulo, no tiene mayor importancia si el sistema a considerar es relacional o de otro tipo. Sin embargo, sería útil indicar cómo son concebidos típicamente los tres niveles de la arquitectura en un sistema relacional en particular:

- Primero, el nivel conceptual en dicho sistema será definitivamente relacional, en el sentido de que los objetos visibles en ese nivel serán tablas relacionales y los operadores serán de tipo relacional (incluyendo los operadores *restringir* y *proyectar* en particular, que expusimos brevemente en el capítulo 1).
- Segundo, una vista externa dada también será casi siempre relacional, o algo muy parecido a ello; por ejemplo, las declaraciones de registro de PL/I y COBOL de la figura 2.2 podrían ser consideradas a grandes rasgos como análogas de las declaraciones de PL/I y COBOL, respectivamente, de una tabla relacional en un sistema relacional.

Nota: Debemos mencionar de paso que el término "vista externa" (a menudo abreviado solamente como "vista") tiene por desgracia un significado más bien específico en contextos relacionales y que éste *no* es idéntico al significado que se le asigna en este capítulo. Para una explicación y exposición del significado relacional, consulte los capítulos 3 y 9.

- Tercero, el nivel interno *no* será relacional, ya que los objetos en ese nivel no serán sólo tablas relacionales (almacenadas); en vez de ello, serán los mismos tipos de objetos que se encuentran en el nivel interno de cualquier otro tipo de sistema (registros almacenados, apuntadores, índices, tablas de dispersión, etcétera). De hecho, el modelo relacional como tal **no tiene nada en absoluto que decir** acerca del nivel interno; para repetir lo dicho en el capítulo 1, tiene que ver con la forma en que la base de datos se presenta ante el *usuario.*

Ahora procederemos a explicar con más detalle los tres niveles de la arquitectura, comenzando con el nivel externo. A lo largo de nuestra explicación haremos constantes referencias a la figura 2.3, la cual muestra los principales componentes de la arquitectura y sus interrelaciones.

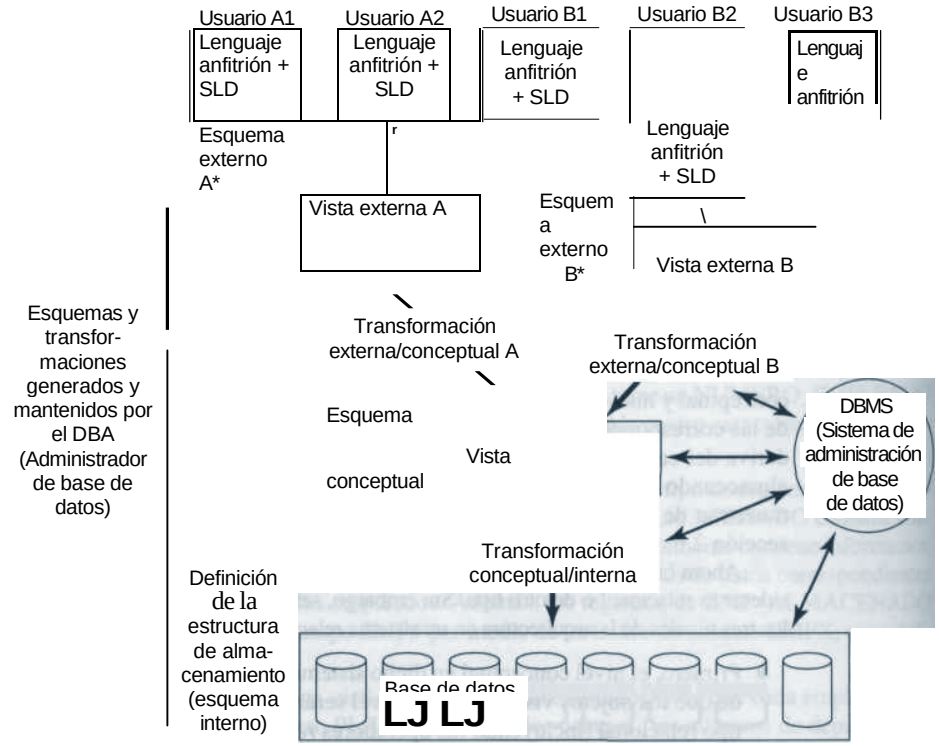

Figura 2.3 Arquitectura detallada del sistema.

2.3 /EL NIVEL EXTERNO

El nivel externo es el nivel del usuario individual. Como expliqué en el capítulo 1, un usuario dado puede ser un programador de aplicaciones o bien un usuario final con cualquier grado de sofisticación. (El DBA es un importante caso especial; pero a diferencia de otros usuarios, el DBA también necesitará interesarse en los niveles conceptual e interno. Vea las dos secciones siguientes.) Cada usuario tiene a su disposición un **lenguaje:**

- Para el programador de aplicaciones, éste será ya sea un lenguaje de programación con vencional (por ejemplo, PL/I, C++, Java) o bien un lenguaje de tipo propietario que sea es pecífico al sistema en cuestión. A menudo, a estos lenguajes de tipo propietario se les denomina lenguajes de "cuarta generación" (4GLs); siguiendo las —¡confusas!— bases de que (a) el código de máquina, el lenguaje ensamblador y los lenguajes como PL/I pueden ser vistos como tres "generaciones" de lenguajes anteriores, y (b) los lenguajes de tipo pro pietario representan la misma clase de mejora sobre los lenguajes de "tercera generación" (3GLs) que la que tuvieron estos lenguajes sobre el lenguaje ensamblador y la que tuvo el lenguaje ensamblador sobre el código de máquina.
- Para el usuario final, el lenguaje será ya sea un lenguaje de consulta o bien algún lenguaje de finalidad específica, tal vez controlado por formularios o por menús, confeccionado para los requerimientos de ese usuario y manejado por algún programa de aplicación en línea (como expliqué en el capítulo 1).

En nuestro caso, lo importante acerca de dichos lenguajes es que incluirán un **sublenguaje de datos;** es decir, un subconjunto del lenguaje total que se ocupe específicamente de los objetos y operaciones de la base de datos. Se dice que el sublenguaje de datos (abreviado como SLD en la figura 2.3) está **incrustado** dentro de su **lenguaje anfitrión** correspondiente. El lenguaje anfitrión es el responsable de proporcionar diversas propiedades que no son específicas de la base de datos, como las variables locales, las operaciones de cálculo, la lógica de bifurcación, etcétera. Un sistema determinado podría manejar cualquier cantidad de lenguajes anfitrión y cualquier número de sublenguajes de datos; sin embargo, un sublenguaje de datos específico soportado por casi todos los sistemas actuales es el lenguaje SQL que explicamos brevemente en el capítulo 1. La mayoría de dichos sistemas permiten que SQL sea utilizado de manera *interactiva,* como un lenguaje de consulta independiente, e *incrustado* en otros lenguajes como PL/I o Java (para una explicación más amplia, consulte el capítulo 4).

Ahora bien, aunque para fines de la arquitectura es conveniente distinguir entre el sublenguaje de datos y el lenguaje anfitrión que lo contiene, ambos podrían *no* ser distintos en lo que al usuario concierne; y de hecho, desde el punto de vista del usuario, es probablemente mejor que no lo sean. Si no son distintos, o si difícilmente pueden distinguirse, decimos que están **fuertemente acoplados.** Si son clara y fácilmente separables, decimos que están **débilmente** acoplados. Aunque algunos sistemas comerciales (en especial los orientados a objetos; vea el capítulo 24) sí manejan el acoplamiento fuerte, la mayoría no lo hace; en particular, los sistemas SQL generalmente sólo manejan el acoplamiento débil. (El acoplamiento fuerte ofrece un conjunto más uniforme de propiedades para el usuario, aunque obviamente implica más esfuerzo por parte de los desarrolladores del sistema, un hecho que presumiblemente cuenta para el *statu quo.)*

En principio, cualquier sublenguaje de datos determinado es en realidad una combinación de por lo menos dos lenguajes subordinados: un DDL **(lenguaje de definición de datos),** que

permite la definición o declaración de objetos de base de datos, y un DML **(lenguaje de manipulación de datos),** que permite la manipulación o procesamiento de dichos objetos. Por ejemplo, considere al usuario de PL/I de la figura 2.2 de la sección 2.2. El sublenguaje para ese usuario consiste en aquellas características de PL/I que se usan para comunicarse con el DBMS:

- La parte del DDL consiste en aquellas construcciones declarativas de PL/I necesarias para declarar objetos de base de datos; la propia instrucción DECLARE (DCL), ciertos tipos de datos de PL/I, tal vez extensiones especiales de PL/I para manejar nuevos objetos que no maneja el PL/I existente.
- La parte del DML consiste en aquellas instrucciones ejecutables de PL/I que transfieren in formación hacia y desde la base de datos; de nuevo, que incluyen tal vez nuevas instruccio nes especiales.

Nota: Para ser más precisos, debemos decir que al momento de la publicación de este libro, PL/I no incluía ninguna característica específica de base de datos. En particular, las instrucciones "DML" por lo regular sólo son instrucciones CALL de PL/I que invocan al DBMS (aunque esas instrucciones podrían de alguna forma estar disfrazadas sintácticamente para hacerlas un poco más sencillas para el usuario; vea la explicación de SQL incrustado, en el capítulo 4).

Volviendo a la arquitectura, ya indicamos que un usuario individual se interesará generalmente sólo por alguna parte de toda la base de datos; es más, la vista que el usuario tiene de esa parte normalmente será un poco abstracta al compararla con la forma en que los datos están almacenados físicamente. El término ANSI/SPARC utilizado para la vista de un usuario individual es el de **vista externa.** Por lo tanto, una vista externa es el contenido de una base de datos como lo ve algún usuario en particular (es decir, para ese usuario, la vista externa *es* la base de datos). Por ejemplo, un usuario del Departamento de Personal podría ver a la base de datos como una colección de ocurrencias de los registros de empleado y departamento, y podría desconocer las ocurrencias de los registros de parte y proveedor que ven los usuarios del Departamento de Compras.

Entonces, en general, una vista externa consiste en muchas ocurrencias de cada uno de los tipos de **registro externo** (que *no* es necesariamente lo mismo que un registro almacenado).* El sublenguaje de datos del usuario está definido en términos de registros externos; por ejemplo, una operación de *recuperación* del DML recuperará ocurrencias de registros externos, no ocurrencias de registros almacenados. *Nota:* Ahora podemos ver que el término "registro lógico", empleado ocasionalmente en el capítulo 1, se refería en realidad a un registro externo. De hecho, desde este punto de vista, trataremos de evitar el término "registro lógico".

Cada vista externa está definida por medio de un **esquema externo,** el cual consiste básicamente en definiciones de cada uno de los diversos tipos de registros externos de esa vista (observe una vez más en la figura 2.2, un par de ejemplos sencillos). El esquema externo fue escrito utilizando la parte DDL del sublenguaje de datos del usuario. (De ahí que en ocasiones se haga referencia a ese DDL como *DDL externo.)* Por ejemplo, el tipo de registro externo de empleado podría definirse como un campo de número de empleado con seis caracteres, más un campo de

---

*Aquí, damos por hecho que toda la información está representada a un nivel externo en forma de registros. Sin embargo, algunos sistemas permiten que la información sea representada en otras formas en vez de, o también en forma de, registros. Para que un sistema emplee dichos métodos alternativos, será necesario adecuar las definiciones y explicaciones dadas en esta sección. Se aplican también observaciones similares a los niveles conceptual e interno. La consideración detallada de estos aspectos está más allá del alcance de esta primera parte del libro; para una explicación más amplia, consulte los capítulos 13 (en especial la sección de "Referencias y bibliografía") y 24.

salario con cinco dígitos (decimales) y así sucesivamente. Además, debe haber una definición de la *transformación* entre el esquema externo y el esquema *conceptual* subyacente (vea la siguiente sección). Más adelante, en la sección 2.6, consideraremos dicha transformación.

4 EL NIVEL CONCEPTUAL

La **vista conceptual** es una representación de todo el contenido de la información de la base de datos, de nuevo (al igual que con la vista externa) en una forma un poco abstracta comparada con la forma en la que por lo regular se almacenan los datos físicamente. También será muy diferente de la forma en que cualquier usuario específico ve los datos. En términos generales, la vista conceptual pretende ser una vista de los datos "tal como son", en vez de tal como los usuarios están obligados a verlos debido a las limitaciones (por ejemplo) del lenguaje o el hardware en particular que pudieran utilizar.

La vista conceptual consiste en muchas ocurrencias de varios tipos de **registro conceptual.** Por ejemplo, podría consistir en un conjunto de ocurrencias de los registros de departamento, más un conjunto de ocurrencias de los registros de empleado, más un conjunto de ocurrencias de los registros de proveedor, más un conjunto de ocurrencias de los registros de parte (y así sucesivamente). Por otra parte, un registro conceptual no es necesariamente lo mismo que un registro externo, ni que un registro almacenado.

La vista conceptual está definida por medio del **esquema conceptual,** el cual comprende definiciones de cada uno de los diversos tipos de registros conceptuales (de nuevo, consulte la figura 2.2 para ver un ejemplo sencillo). El esquema conceptual está escrito con otro lenguaje de definición de datos, el *DDL conceptual.* Si se va a lograr la independencia física de los datos, entonces las definiciones conceptuales de DDL no deben comprender en lo absoluto ninguna consideración de la representación física ni de la técnica de acceso; deben ser *únicamente* definiciones del contenido de la información. Por lo tanto, en el esquema conceptual no debe haber ninguna referencia para la representación de campos almacenados, la secuencia de registros almacenados, los índices, los esquemas de dispersión, los apuntadores o cualquier otro detalle de almacenamiento y acceso. Si el esquema conceptual se hace verdaderamente independiente de los datos, entonces los esquemas externos, que están definidos en términos del esquema conceptual (vea la sección 2.6), también serán *forzosamente* independientes de los datos.

Entonces, la vista conceptual es una vista del contenido total de la base de datos, y el esquema conceptual es una definición de esa vista. Sin embargo, sería engañoso dar por hecho que el esquema conceptual no es nada más que un conjunto de definiciones muy similar a las definiciones que se encuentran (por ejemplo) en un programa COBOL actual. Las definiciones del esquema conceptual pretenden incluir muchas características adicionales, como las restricciones de seguridad y de integridad mencionadas en el capítulo 1. Algunas autoridades van más lejos al sugerir que el objetivo final del esquema conceptual es describir toda la empresa; no sólo los datos como tales, sino también la forma en que son utilizados, la forma en que fluyen de un punto a otro dentro de la empresa, su función en cada punto, lo controles de auditoría u otros que se aplican en cada punto, etcétera [2.3]. Sin embargo, debemos enfatizar que en la actualidad ningún sistema soporta realmente un esquema conceptual de cualquier cosa que se aproxime a este grado de amplitud; en la mayoría de los sistemas existentes, el "esquema conceptual" es en realidad algo más que una simple unión de todos los esquemas externos individuales más ciertas restricciones de seguridad y de integridad. Aunque en verdad es posible que los sistemas futuros sean mucho más sofisticados en cuanto al soporte del nivel conceptual.

2.5 EL NIVEL INTERNO

El tercer nivel de la arquitectura es el nivel interno. La **vista interna** es una representación de bajo nivel de toda la base de datos y consiste en muchas ocurrencias de cada uno de los diversos tipos de **registros internos.** "Registro interno" es el término de ANSI/SPARC para la construcción que hemos venido llamando registro *almacenado* (y que seguiremos utilizando). Por lo tanto, la vista interna está todavía distante del nivel físico, ya que no tiene que ver con términos como registros *físicos* —también denominados **bloques** o **páginas**— ni con ninguna consideración específica de los dispositivos, como el tamaño de los cilindros o de las pistas. En otras palabras, la vista interna en efecto da por hecho un espacio de direcciones lineal infinito; los detalles de cómo el espacio de direcciones se asocia con el almacenamiento físico, son en gran medida específicos del sistema y se omiten deliberadamente de la arquitectura general. *Nota:* El bloque. o página, es la **unidad de** E/S; es decir, es la cantidad de datos transferidos entre el almacenamiento secundario y la memoria principal en una sola operación de E/S. Los tamaños típicos de página son 1 KB, 2 KB o 4 KB (1 KB = 1024 bytes).

La vista interna se describe por medio del **esquema interno,** el cual no sólo define los diversos tipos de registros almacenados sino que especifica también qué índices existen, cómo están representados los campos almacenados, en qué secuencia están dichos registros, etcétera. (Una vez más, observe un ejemplo sencillo en la figura 2.2.) El esquema interno está escrito utilizando otro lenguaje más de definición de datos: el *DDL interno*. *Nota:* En este libro usaremos normalmente los términos más intuitivos "estructura de almacenamiento" o "base de datos almacenada" en lugar de "vista interna", así como el término "definición de la estructura de almacenamiento" en lugar de "esquema interno".

Para terminar, señalamos que, en ciertas situaciones excepcionales, a los programas de aplicación —en particular, las aplicaciones *de utilería* (vea la sección 2.11)— se les podría permitir operar directamente en el nivel interno en vez del nivel externo. Sobra decir que no es recomendable esta práctica, pues representa un riesgo para la seguridad (ya que se ignoran las restricciones de seguridad) y un riesgo para la integridad (debido a que, de igual manera, se ignoran las restricciones de integridad). Además, para iniciar, el programa será dependiente de los datos; aunque, en ocasiones, ésta podría ser la única forma de obtener la funcionalidad o el rendimiento requeridos (tal como le sucede al usuario de un lenguaje de programación de alto nivel que ocasionalmente tendría que descender al lenguaje ensamblador para satisfacer ciertos objetivos de funcionalidad o rendimiento).

2.6 TRANSFORMACIONES

Además de los tres niveles *corno tales,* la arquitectura de la figura 2.3 comprende ciertas **transformaciones;** en general, una transformación conceptual/interna y varias transformaciones externas/conceptual:

- La transformación *conceptual/interna* define la correspondencia entre la vista conceptual y la base de datos almacenada, y especifica cómo están representados los registros y campos conceptuales en el nivel interno. Si se modifica la estructura de la base de datos —es decir, si se hace un cambio a la definición de la estructura de almacenamiento—, entonces por consiguiente será necesario modificar la transformación conceptual/interna, de manera que

el esquema conceptual pueda permanecer invariable. (Por supuesto, es responsabilidad del DBA administrar dichos cambios.) En otras palabras, los efectos de dichos cambios deben aislarse por debajo del nivel conceptual, a fin de preservar la independencia física de los datos.

- La transformación *externa/conceptual* define la correspondencia entre una vista externa en particular y la vista conceptual. En general, las diferencias que puedan existir entre estos dos niveles son análogas a aquellas que puedan existir entre la vista conceptual y la base de datos almacenada. Por ejemplo, los campos pueden tener diferentes tipos de datos; los nombres de los registros y campos pueden ser cambiados; varios campos conceptuales pueden combinarse en un solo registro externo (virtual); etcétera. Puede existir cualquier cantidad de vistas externas al mismo tiempo; cualquier número de usuarios puede compartir una vista externa dada; es posible traslapar diferentes vistas externas.

Nota: Debe quedar claro que así como la transformación conceptual/interna es la clave para la independencia física de los datos, también las transformaciones externas/conceptual son la clave para la independencia **lógica** de los datos. Como vimos en el capítulo 1, un sistema proporciona la independencia física de los datos [1.3] si los usuarios y los programas de usuario son inmunes a los cambios en la estructura física de la base de datos almacenada. De igual manera, un sistema proporciona la independencia *lógica* de los datos [1.4] si los usuarios y los programas de usuario también son inmunes a los cambios en la estructura *lógica* de la base de datos (lo que significa cambios al nivel conceptual o "lógico de la comunidad"). En los capítulos 3 y 9, tendremos más que decir respecto de este tema importante.

Por cierto, la mayoría de los sistemas permiten que la definición de ciertas vistas externas se exprese en términos de otras (en efecto, mediante transformaciones *externas/externas),* en vez de requerir siempre una definición explícita de la transformación al nivel conceptual; una característica útil si varias vistas externas son parecidas entre sí. En particular, los sistemas relacionales brindan dicha posibilidad.

2.7 EL ADMINISTRADOR DE BASE DE DATOS

Como expliqué en el capítulo 1, el DA (administrador de *datos)* es la persona que toma las decisiones de estrategia y política con respecto a los datos de la empresa y el DBA (administrador de *base* de datos) es la persona que proporciona el apoyo técnico necesario para implementar dichas decisiones. Por lo tanto, el DBA es el responsable del control general del sistema al nivel técnico. Ahora podemos describir con un poco más de detalle algunas de las tareas del DBA. En general, estas tareas comprenden al menos todas las siguientes:

- *Definir el esquema conceptual*

  Es trabajo del administrador de datos decidir exactamente qué información contendrá la base de datos; en otras palabras, identificar las entidades de interés para la empresa e identificar la información que hay que registrar acerca de dichas entidades. Por lo regular a este proceso se le conoce como **diseño lógico** —en ocasiones *conceptual*— **de la base de datos.** Una vez que el administrador decidió el contenido de la base de datos a un nivel abstracto, entonces el DBA creará el esquema conceptual correspondiente, utilizando el DLL conceptual. El DBMS usará la forma objeto (compilada) de ese esquema para responder a las

peticiones de acceso. La forma fuente (sin compilar) actuará como documento de referencia para los usuarios del sistema.

Nota: En la práctica, las cosas pueden no ser tan claras como sugieren los señalamientos anteriores. En algunos casos, el administrador de datos podría crear directamente el esquema conceptual. En otras, el DBA podría hacer el diseño lógico.

■ *Definir el esquema interno*

El DBA también debe decidir la forma en que van a ser representados los datos en la base de datos almacenada. A este proceso se le conoce comúnmente como diseño físico de la base de datos.* Una vez realizado el diseño físico, el DBA deberá crear la definición de la estructura de almacenamiento correspondiente (es decir, el esquema interno), utilizando el DDL interno. Además, también deberá definir la transformación conceptual/interna asociada. En la práctica, es factible que uno de los dos DDLs (el conceptual o el interno; pero más probablemente el primero) incluya los medios para definir esa transformación; aunque las dos funciones (crear el esquema y definir la transformación) deben ser claramente separables. Al igual que el esquema conceptual, tanto el esquema interno como la transformación correspondiente existirán en las formas fuente y objeto.

■ *Establecer un enlace con los usuarios*

Es asunto del DBA enlazarse con los usuarios para asegurar que los datos necesarios estén disponibles y para escribir (o ayudar a escribir) los esquemas externos necesarios, utilizando el DDL externo aplicable. (Como ya mencionamos, un sistema dado podría manejar varios DDLs externos distintos.) Además, también es necesario definir las transformaciones externas/conceptual correspondientes. En la práctica, es probable que el DDL externo incluya los medios para especificar dichas transformaciones, pero, una vez más, los esquemas y las transformaciones deben ser claramente separables. Cada esquema externo, con la transformación correspondiente, existirá en las formas tanto fuente como objeto.

Otros aspectos de la función de enlace con los usuarios incluyen la asesoría sobre el diseño de aplicaciones; una capacitación técnica; ayuda en la determinación y resolución de problemas; así como otros servicios profesionales similares.

Definir las restricciones de seguridad y de integridad

Como ya expliqué, las restricciones de seguridad y de integridad pueden ser vistas como parte del esquema conceptual. El DDL conceptual debe incluir facilidades para especificar dichas restricciones.

■ *Definir las políticas de vaciado y recarga*

Una vez que una empresa se compromete con un sistema de base de datos, se vuelve drásticamente dependiente del funcionamiento exitoso de dicho sistema. En el caso de que se produzca un daño en cualquier parte de la base de datos —ocasionado, por ejemplo, por un error humano o por una falla en el hardware o en el sistema operativo— resulta esencial poder reparar los datos afectados con el mínimo de demora y con tan poco efecto como sea posible sobre el resto del sistema. Por ejemplo, de manera ideal no debería afectarse la disponibilidad de los datos que *no* fueron afectados. El DBA debe definir e implementar un esquema apropiado de control de daños que comprenda (a) la descarga o "vaciado" periódico

*Observe la secuencia: Primero decida qué datos desea, luego decida cómo representarlos en el almacenamiento. El diseño físico sólo deberá hacerse *después* de realizar el diseño lógico.

de la base de datos en un dispositivo de almacenamiento de respaldo y (b) la recarga de la base de datos cuando sea necesario, a partir del vaciado más reciente.

Por cierto, la necesidad de una rápida reparación de los datos es una de las razones por las que podría ser buena idea repartir todos los datos en varias bases de datos, en vez de mantenerlos todos en un mismo lugar; una base de datos individual podría muy bien definir una unidad para fines de vaciado y de recarga. En este sentido, observe que ya existen los *sistemas de terabytes** —es decir, los sistemas comerciales que almacenan más de un billón de bytes, aproximadamente— y se predice que los sistemas futuros serán mucho más grandes. Sobra decir que tales sistemas *VLDB* ("base de datos muy grande") requieren de una administración muy cuidadosa y sofisticada, en especial si hay necesidad de una disponibilidad continua (lo que sucede normalmente). Sin embargo, para efectos de simplicidad, seguiremos hablando como si de hecho sólo existiera una base de datos individual.

*■ *Supervisar el rendimiento y responder a los requerimientos cambiantes*

Como indiqué en el capítulo 1, el DBA es el responsable de organizar el sistema de tal manera que se obtenga el rendimiento "ideal para la empresa" y de hacer los ajustes apropiados —es decir, **afinar**— conforme las necesidades cambien. Por ejemplo, podría ser necesario **reorganizar** de vez en cuando la base de datos almacenada para asegurar que los niveles de rendimiento se mantengan aceptables. Como ya mencioné, todo cambio al nivel (interno) de almacenamiento físico debe estar acompañado por el cambio correspondiente en la definición de la transformación conceptual/interna, de manera que el esquema conceptual permanezca constante.

Desde luego, la anterior no es una lista detallada; simplemente pretende dar una idea del alcance y naturaleza de las responsabilidades del DBA.

2.8 EL SISTEMA DE ADMINISTRACIÓN / DE BASE DE DATOS

El DBMS **(sistema de administración de base de datos)** es el software que maneja todo acceso a la base de datos. De manera conceptual, lo que sucede es lo siguiente (consulte una vez más la figura 2.3):

1. Un usuario emite una petición de acceso, utilizando algún sublenguaje de datos específico (por lo regular SQL).
2. El DBMS intercepta esa petición y la analiza.
3. El DBMS inspecciona, en su momento, (las versiones objeto de) el esquema externo para ese usuario, la transformación externa/conceptual correspondiente, el esquema conceptual, la transformación conceptual/interna y la definición de la estructura de almacenamiento.
4. El DBMS ejecuta las operaciones necesarias sobre la base de datos almacenada.

---

*1024 bytes = 1 KB (kilobyte); 1024 KB = 1 MB (megabyte); 1024 MB = 1 GB (gigabyte); 1024 GB = 1 TB (terabyte); 1024 TB = 1 PB (petabyte); 1024 PB = 1 EB (exabyte). Tome nota que un gigabyte equivale aproximadamente a mil millones de bytes y que algunos textos en inglés emplean en ocasiones la abreviatura BB en lugar de GB.

A manera de ejemplo, considere lo que implica la recuperación de una ocurrencia de un registro externo en particular. En general, los campos serán solicitados desde varias ocurrencias de registros conceptuales, y cada ocurrencia de un registro conceptual solicitará a su vez campos de varias ocurrencias de registros almacenados. Entonces, de manera conceptual, el DBMS debe primero recuperar todas las ocurrencias solicitadas de los registros almacenados, luego construir las ocurrencias solicitadas de los registros conceptuales y después construir las ocurrencias solicitadas de los registros externos. En cada etapa, podrían ser necesarias conversiones de tipos de datos u otras.

Desde luego, la descripción anterior está muy simplificada; en particular, implica que todo el proceso es interpretativo, ya que asume que todo el proceso de analizar la petición, inspeccionar los diversos esquemas, etcétera, se realiza en tiempo de ejecución. La interpretación, por su parte, a menudo implica un rendimiento deficiente debido a la sobrecarga del tiempo de ejecución. Sin embargo, en la práctica podría ser posible *compilar* las peticiones de acceso antes del tiempo de ejecución (en particular, varios productos SQL actuales hacen esto; vea la anotación a las referencias [4.12] y [4.26] en el capítulo 4).

Analicemos ahora las funciones del DBMS con un poco más de detalle. Dichas funciones comprenderán por lo menos el manejo de todas las siguientes (consulte la figura 2.4):

- *Definición de datos*

  El DBMS debe ser capaz de aceptar definiciones de datos (esquemas externos, el esquema conceptual, el esquema interno y todas las transformaciones respectivas) en la forma fuente y convertirlas a la forma objeto correspondiente. En otras palabras, el DBMS debe incluir entre sus componentes un **procesador DDL,** o **compilador DDL,** para cada uno de los diversos DDLs (lenguajes de definición de datos). El DBMS también debe "entender" 1: definiciones DDL, en el sentido que, por ejemplo, "entienda" que los registros externos EMPLEADO incluyen un campo SALARIO; entonces, debe poder utilizar este conocimiento para analizar y responder a las peticiones de manipulación de datos (por ejemplo. "Obtener todos los empleados con salario < $50,000").

- *Manipulación de datos*

  El DBMS debe ser capaz de manejar peticiones para recuperar, actualizar o eliminar datos existentes en la base de datos o agregar nuevos datos a ésta. En otras palabras, el DBMS debe incluir un componente **procesador DML** o **compilador DML** para tratar con el DML (lenguaje de manipulación de datos).

- En general, las peticiones DML pueden ser "planeadas" o "no planeadas":

  a. Una petición **planeada** es aquella cuya necesidad fue prevista antes del momento de eje cutar la petición. Probablemente el DBA habrá afinado el diseño físico de la base de datos de tal forma que garantice un buen desempeño para las peticiones planeadas.

  b. En contraste, una petición **no planeada** es una consulta *ad hoc;* es decir, una petición para la que no se previó por adelantado su necesidad, sino que en vez de ello, surgió sin pensarlo. El diseño físico de la base de datos podría o no ser el adecuado para la peti ción específica en consideración.

  Para utilizar la terminología presentada en el capítulo 1 (sección 1.3), las peticiones planeadas son características de las aplicaciones "operacionales" o de "producción", **mientra** que las peticiones no planeadas son características de las aplicaciones de "apoyo a la **toma** de decisiones". Además, peticiones planeadas serán emitidas generalmente desde programas de aplicación preescritos, mientras que las no planeadas, por definición, serán emitidas

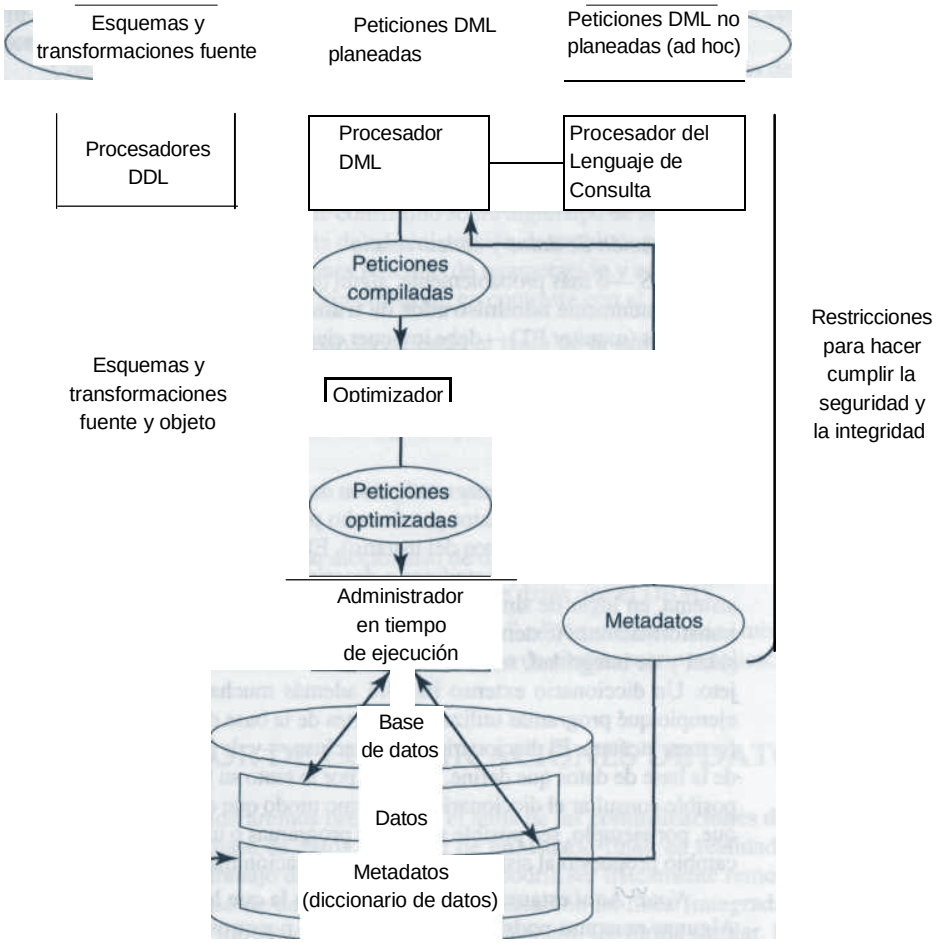

Figura 2.4 Funciones y componentes principales del DBMS.

en forma interactiva mediante algún **procesador del lenguaje de consulta.** *Nota:* Como vimos en el capítulo 1, el procesador del lenguaje de consulta es en realidad una aplicación integrada en línea, no una parte del DBMS *como tal;* lo incluimos en la figura 2.4 para ampliar la explicación.

Optimización y ejecución

Las peticiones DML, planeadas o no planeadas, deben ser procesadas por el componente **optimizador,** cuya finalidad es determinar una forma eficiente de implementar la petición. En el capítulo 17 explicamos la optimización en detalle. Las peticiones optimizadas se ejecutan entonces bajo el control del **administrador en tiempo de ejecución.** *Nota:* En la práctica,

el administrador en tiempo de ejecución recurrirá probablemente a algún tipo de *administrador de archivos* para acceder a los datos almacenados. Al final de esta sección explicamos brevemente los administradores de archivos.

- *Seguridad e integridad de los datos*

  El DBMS debe vigilar las peticiones del usuario y rechazar todo intento de violar las restricciones de seguridad y de integridad definidas por el DBA (vea la sección anterior). Estas tareas pueden realizarse durante el tiempo de compilación, de ejecución o entre ambos.

- *Recuperación de datos y concurrencia*

  El DBMS —o más probablemente, algún otro componente de software relacionado, denominado comúnmente **administrador de transacciones** o **monitor de procesamiento de transacciones** (monitor PT)— debe imponer ciertos controles de recuperación y concurrencia. Los detalles de estos aspectos del sistema están fuera del alcance de este capítulo; para una explicación más a fondo, consulte la parte IV de este libro. *Nota:* El administrador de transacciones no se muestra en la figura 2.4 debido a que, por lo regular, no forma parte del DBMS *como tal.*

- *Diccionario de datos*

  El DBMS debe proporcionar una función de **diccionario de datos.** Este diccionario puede ser visto como una base de datos por derecho propio (aunque una base de datos del sistema mí que como una base de datos del usuario). El diccionario contiene "datos acerca de los datos" (en ocasiones llamados *metadatos* o *descriptores);* es decir, *definiciones* de otros objetos del sistema, en lugar de simples "datos en bruto". En particular, todos los diversos esquemas y transformaciones (externos, conceptuales, etcétera) y todas las diversas restricciones de seguridad y de integridad, serán almacenadas en el diccionario, tanto en forma fuente como objeto. Un diccionario extenso incluirá además mucha información adicional; mostrará por ejemplo qué programas utilizan qué partes de la base de datos, qué usuarios necesitan qué informes, etcétera. El diccionario podría incluso —y de hecho, debería— estar integrado dentro de la base de datos que define, e incluir por lo tanto su propia definición. En realidad, debe ser posible consultar el diccionario del mismo modo que cualquier otra base de datos, de maner que, por ejemplo, sea posible saber qué programas o usuarios se podrían ver afectados por un cambio propuesto al sistema. Para una explicación más amplia, consulte el capítulo 3.

  Nota: Aquí estamos tocando un área en la que hay mucha confusión de terminología. Algunas personas podrían referirse a lo que nosotros llamamos diccionario como ***directorio*** o *catálogo* —con la implicación de que los directorios y catálogos son, en cierta forma. inferiores a un verdadero diccionario— y podrían reservar el término *diccionario* para hacer referencia a una clase específica (importante) de herramienta de desarrollo de aplicaciones. Otros términos que también son utilizados, a veces, para hacer referencia a este último tipo de objeto son *depósito de datos* (vea el capítulo 13) y *enciclopedia de datos.*

- *Rendimiento*

  Sobra decir que el DBMS debe realizar todas las tareas antes identificadas de la manera más eficiente posible.

Podemos resumir todo lo anterior diciendo que la finalidad general del DBMS consiste en proporcionar una **interfaz de usuario** para el sistema de base de datos. Podemos definir la interfaz de usuario como un límite en el sistema debajo del cual todo es invisible para el usuario. Por lo tanto, por definición la interfaz de usuario se encuentra en el nivel *externo.* Sin embargo, como veremos en el capítulo 9, existen algunas situaciones en las que es poco probable que la

vista externa difiera de manera significativa de la parte relevante de la vista conceptual subyacente, por lo menos en los productos comerciales SQL actuales.

Concluimos esta sección con una breve comparación entre los sistemas de administración de bases de datos y los sistemas de administración de **archivos** *(administradores de archivos,* o *servidores de archivos,* para abreviar). En esencia, el administrador de archivos es el componente del sistema operativo subyacente que administra los archivos almacenados; por lo tanto, hablando en términos generales, está "más cerca del disco" de lo que lo está el DBMS. (De hecho, el DBMS es generalmente construido sobre algún tipo de administrador de archivos.) Por lo tanto, el usuario de un sistema de administración de archivos podrá crear y destruir archivos almacenados y realizar operaciones sencillas de recuperación y actualización sobre registros almacenados en dichos archivos. Sin embargo, en contraste con el DBMS:

- Los administradores de archivos no están al tanto de la estructura interna de los registros almacenados, de ahí que no puedan manejar peticiones que se basen en el conocimiento de esa estructura.
- Por lo regular ofrecen poco o ningún soporte para las restricciones de seguridad y de in tegridad.
- Por lo regular ofrecen poco o ningún soporte para los controles de recuperación y con currencia.
- No hay un concepto real de diccionario de datos en el nivel del administrador de archivos.
- Proporcionan mucho menos independencia de datos que el DBMS.
- Por lo regular los archivos no están "integrados" o "compartidos" en el mismo sentido que en una base de datos (normalmente son exclusivos de cierto usuario o aplicación en particular).

2.9 EL ADMINISTRADOR DE COMUNICACIONES DE DATOS

En esta sección consideraremos brevemente el tema de las **comunicaciones de datos.** Las peticiones emitidas a la base de datos por parte de un usuario final, en realidad son transmitidas desde la estación de trabajo del usuario —la cual podría ser físicamente remota con respecto al propio sistema de base de datos— hacia cierta aplicación en línea (integrada u otra) y de ahí hacia el DBMS en la forma de *mensajes de comunicación.* De forma similar, las respuestas que regresan del DBMS y la aplicación en línea hacia la estación de trabajo del usuario, también son transmitidas en forma de dichos mensajes. Todas estas transmisiones de mensajes se llevan a cabo bajo el control de otro componente de software, el **administrador de comunicaciones de datos** (administrador CD).

Este administrador no forma parte del DBMS, sino que es un sistema autónomo por derecho propio. Sin embargo, puesto que es claramente necesario que trabaje en armonía con el DBMS, en ocasiones se les considera como socios igualitarios en una empresa de más alto nivel denominada **sistema de base de datos/comunicaciones de datos** (sistema BD/CD); en el cual el DBMS se ocupa de la base de datos y el administrador CD maneja todos los mensajes hacia y desde el DBMS, o más precisamente, hacia y desde las aplicaciones que el DBMS utiliza. No obstante, en este libro tendremos relativamente poco que decir con respecto al manejo de mensajes como tal (por sí solo, es un tema muy extenso). En la sección 2.12 explicamos brevemente la cuestión de la comunicación *entre sistemas distintos* (es decir, entre máquinas distintas en una red de comunicaciones, como Internet); aunque en realidad ése es un tema aparte.

2.10 ARQUITECTURA CLIENTE-SERVIDOR

Hasta ahora, en este capítulo hemos venido explicando los sistemas de bases de datos desde el punto de vista de la así llamada arquitectura ANSI/SPARC. En la figura 2.3 en particular, presentamos una imagen simplificada de esta arquitectura. En esta sección estudiaremos los sistemas de bases de datos desde una perspectiva ligeramente diferente. Desde luego, la finalidad principal de dichos sistemas es apoyar el desarrollo y la ejecución de las aplicaciones de bases de datos. Por lo tanto, desde un punto de vista más elevado, un sistema de base de datos puede ser visto como un sistema que tiene una estructura muy sencilla de dos partes, las cuales consisten en un **servidor** (también denominado *parte dorsal o servicios de fondo)* y un conjunto de **clientes** (también llamados *partes frontales, aplicaciones para el usuario o interfaces).* Consulte la figura 2.5. *Explicación:*

- El *servidor* es precisamente el propio DBMS. Soporta todas las funciones básicas del DBMS expuestas en la sección 2.8: definición de datos, manipulación de datos, seguridad e integri dad de los datos, etcétera. En particular, proporciona todo el soporte de los niveles externo, conceptual e interno explicados en las secciones 2.3 a 2.6. Por lo tanto, en este contexto, "servidor" es sólo el nombre del DBMS.
- Los *clientes* son las diversas aplicaciones que se ejecutan sobre el DBMS, tanto aplicaciones escritas por el usuario como aplicaciones integradas (es decir, aplicaciones proporcionadas por el fabricante del DBMS o por alguna otra compañía). Por supuesto, en lo que concierne al servidor, no hay diferencia entre las aplicaciones escritas por el usuario y las integradas; todas usan la misma interfaz con el servidor, como la interfaz de nivel externo expuesta en la sección 2.3.

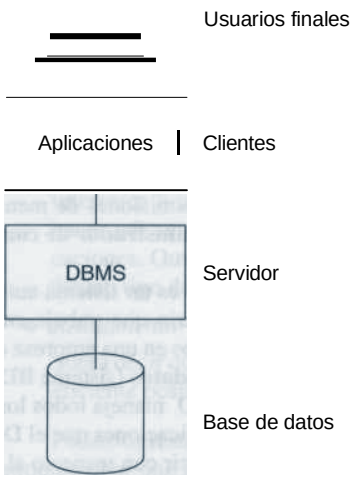

Figura 2.5 Arquitectura cliente-servidor.

Nota: Ciertas aplicaciones especiales de "utilería" podrían constituir una excepción a lo anterior, ya que en ocasiones podrían necesitar operar directamente en el nivel *interno* del sistema (como mencioné en la sección 2.5). Estas utilerías son consideradas más bien como componentes integrales del DBMS, en vez de aplicaciones en el sentido usual. Las explico con más detalle en la siguiente sección.

Profundizaremos un poco sobre la cuestión de aplicaciones escritas por el usuario en comparación con las aplicaciones proporcionadas por el fabricante:

- Las *aplicaciones escritas por el usuario* son en esencia programas de aplicación comunes, escritos por lo regular ya sea en un lenguaje convencional de tercera generación (como C o COBOL) o en algún lenguaje de tipo propietario de cuarta generación; aunque en ambos casos es necesario acoplar de alguna manera el lenguaje con un sublenguaje de datos apropiado, como expliqué en la sección 2.3.
- Las *aplicaciones proporcionadas por el fabricante* (a menudo llamadas **herramientas)** son aplicaciones cuya finalidad básica es auxiliar en la creación y ejecución de otras aplicaciones. Las aplicaciones creadas son aplicaciones confeccionadas para alguna tarea específica (podrían no parecer aplicaciones en el sentido convencional; de hecho, la idea general de las herramientas es permitir a los usuarios, en especial a los usuarios finales, la creación de aplicaciones *sin* tener que escribir programas en un lenguaje de programación convencional). Por ejemplo, una herramienta proporcionada por el fabricante sería un *generador de informes,* cuyo propósito es permitir a los usuarios obtener del sistema informes con cierto formato. Toda petición de informe puede verse como un pequeño programa de aplicación, escrito en un *lenguaje generador de informes* de muy alto nivel (y de finalidad especial).

  Las herramientas proporcionadas por el fabricante pueden dividirse en varias clases relativamente distintas:

  a. Procesadores de lenguaje de consulta;
  b. Generadores de informes;
  c. Subsistemas de gráficos comerciales;
  d. Hojas de cálculo;
  e. Procesadores de lenguaje natural;
  f. Paquetes estadísticos;
  g. Herramientas de administración de copias o "extracción de datos";
  h. Generadores de aplicaciones (incluyen procesadores 4GL);
  i. Otras herramientas de desarrollo de aplicaciones, incluyendo productos de ingeniería de software asistida por computadora (CASE);

  y muchos otros. Los detalles de dichas herramientas exceden el alcance de este libro; sin embargo, señalaremos que (como dijimos antes) debido a que la idea general de un sistema de base de datos es apoyar la creación y ejecución de aplicaciones, la calidad de las herramientas disponibles es, o debería ser, un factor primordial en "la decisión de la base de datos" (es decir, en el proceso de seleccionar un nuevo producto de base de datos). En otras palabras, el DBMS *como tal* no es el único factor a tomar en consideración, ni tampoco es necesariamente el factor más importante.

Cerramos esta sección con una referencia anticipada. Puesto que el sistema en su conjunto puede ser dividido claramente en dos partes (clientes y servidores), surge la posibilidad de

operar los dos en *máquinas diferentes.* En otras palabras, existe el potencial para el **procesamiento distribuido.** Procesamiento distribuido significa que es posible conectar distintas máquinas en cierto tipo de red de comunicaciones, de tal manera que una sola tarea de procesamiento de datos pueda dividirse entre varias máquinas de la red. De hecho, esta posibilidad es tan atractiva —por diversas razones, principalmente económicas— que el término "cliente-servidor" ha llegado a aplicarse casi exclusivamente al caso en el que el cliente y el servidor están, en efecto, en máquinas distintas. En la sección 2.12 explicaré con más detalle el procesamiento distribuido.

2.11 UTILERÍAS

Las utilerías son programas diseñados para ayudar al DBA en sus numerosas tareas administrativas. Como mencioné en la sección anterior, algunos programas de utilería operan en el nivel externo del sistema y en realidad no son más que aplicaciones de propósito especial; algunas podrían incluso no ser proporcionadas por el fabricante del DBMS, sino por alguna otra compañía. Sin embargo, otras utilerías operan directamente en el nivel interno (en otras palabras, son realmente parte del servidor), de ahí que deban ser proporcionadas al fabricante del DBMS.

Aquí hay algunos ejemplos del tipo de utilerías que comúnmente son requeridas en la práctica:

- Rutinas de **carga,** para crear la versión inicial de la base de datos a partir de uno o más archivos del sistema operativo;
- Rutinas de **descarga/recarga,** para descargar la base de datos (o partes de ella), para respaldar los datos almacenados y para recargar datos desde dichas copias de respaldo (por **supuesto,** la "rutina de recarga" es básicamente idéntica a la rutina de carga recién mencionada);
- Rutinas de **reorganización,** para reordenar los datos en la base de datos almacenada, por distintas razones que normalmente tienen que ver con el desempeño; por ejemplo, agrupar datos en el disco de alguna forma en particular o recuperar espacio ocupado por datos que se volvieron obsoletos;
- Rutinas **estadísticas,** para calcular diversas estadísticas de desempeño, como el tamaño de los archivos, las distribuciones de valores, los contadores de operaciones de E/S, etcétera:
- Rutinas de **análisis,** para analizar las estadísticas arriba mencionadas;

y así sucesivamente. Por supuesto, esta lista representa sólo una pequeña muestra del rango de funciones que ofrecen generalmente las utilerías; existe una gran cantidad de otras posibilidades.

2.12 EL PROCESAMIENTO DISTRIBUIDO

Para repetir lo dicho en la sección 2.10, el término "procesamiento distribuido" significa que distintas máquinas pueden conectarse en una red de comunicaciones como Internet, de tal manera que una sola tarea de procesamiento de datos pueda extenderse a varias máquinas de la red. (En ocasiones, también se usa el término "procesamiento paralelo" básicamente con el mismo significado, con excepción de que las máquinas distintas tienden a estar físicamente muy cerca en un sistema "paralelo", lo que no es necesario en un sistema "distribuido"; en último caso, por

ejemplo, podrían estar geográficamente dispersas). La comunicación entre las diversas máquinas es manejada mediante algún tipo de software de administración de redes; tal vez una extensión del administrador CD que explicamos en la sección 2.9 y más probablemente un componente de software independiente.

El procesamiento distribuido presenta muchos niveles o variedades posibles. Para repetir lo dicho en la sección 2.10, un caso sencillo comprendería la operación de los servicios dorsales del DBMS (el servidor) en una máquina y las aplicaciones de usuario (los clientes) en otra. Consulte la figura 2.6.

Como mencioné al final de la sección 2.10, aunque estrictamente hablando el término "cliente-servidor" es puramente arquitectónico. Se ha convertido casi en sinónimo de la organización ilustrada en la figura 2.6, en la que un cliente y un servidor se ejecutan en máquinas diferentes. De hecho, hay muchos argumentos a favor de dicho esquema:

- El primero es básicamente el simple argumento de procesamiento paralelo normal: es decir, ahora se están aplicando muchas unidades de procesamiento para las tareas en conjunto, mientras que el procesamiento del servidor (base de datos) y del cliente (aplicación) se están haciendo en paralelo. De ahí que el tiempo de respuesta y la velocidad real de transporte tengan mejorías.
- Además, la máquina servidor podría ser una máquina construida a la medida y adaptada a la función del DBMS (una "máquina de base de datos") y podría, por lo tanto, proporcionar un mejor desempeño del DBMS.

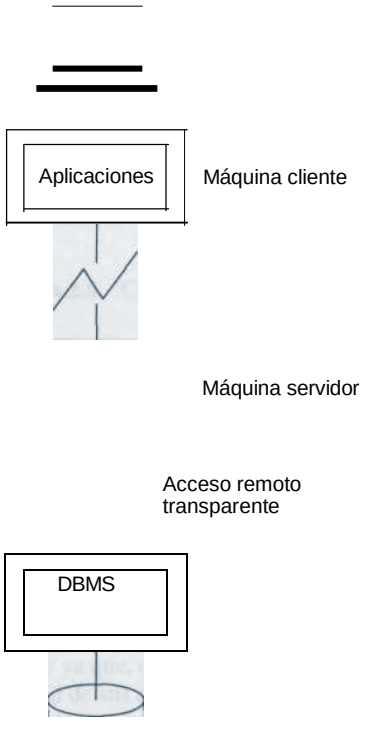

Figura 2.6 Cliente y servidor operando en máquinas diferentes.

- En forma similar, la máquina cliente podría ser una estación de trabajo personal adaptada a las necesidades del usuario final y por lo tanto, capaz de proporcionar mejores interfaces, una alta disponibilidad, respuestas más rápidas y en general una mejor facilidad de uso para el usuario.
- Varias máquinas cliente distintas podrían ser (de hecho serían) capaces de acceder a la misma máquina servidor. Por lo tanto, una sola base de datos podría ser compartida entre varios sistemas cliente distintos (vea la figura 2.7).

Además de los argumentos anteriores, está el punto de que ejecutar los clientes y el servidor en máquinas separadas coincide con la forma en que operan en realidad las empresas. Es muy común que una sola empresa —por ejemplo, un banco— opere muchas computadoras, de tal modo que los datos de una parte de la empresa estén almacenados en una computadora y los datos de otra parte estén almacenados en otra computadora. También es bastante común que los usuarios de una computadora necesiten acceso por lo menos ocasional a los datos almacenados en otra computadora. Siguiendo con el ejemplo del banco, es muy probable que los usuarios de una sucursal necesiten acceder ocasionalmente a los datos almacenados en otra. Observe, por lo tanto, que las máquinas cliente podrían tener almacenados datos propios y que la máquina servidor podría tener sus propias aplicaciones. Por lo tanto, es común que cada máquina actúe como servidor para ciertos usuarios y como cliente para otros (vea la figura 2.8); en otras palabras, cada máquina soportará un *sistema de base de datos completo,* en el sentido al que se refieren las secciones anteriores de este capítulo.

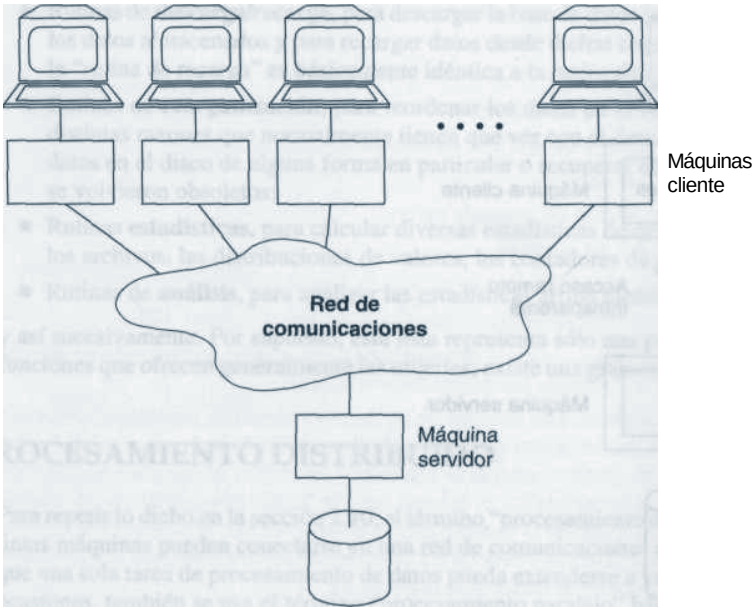

Figura 2.7 Una máquina servidor, varias máquinas cliente.

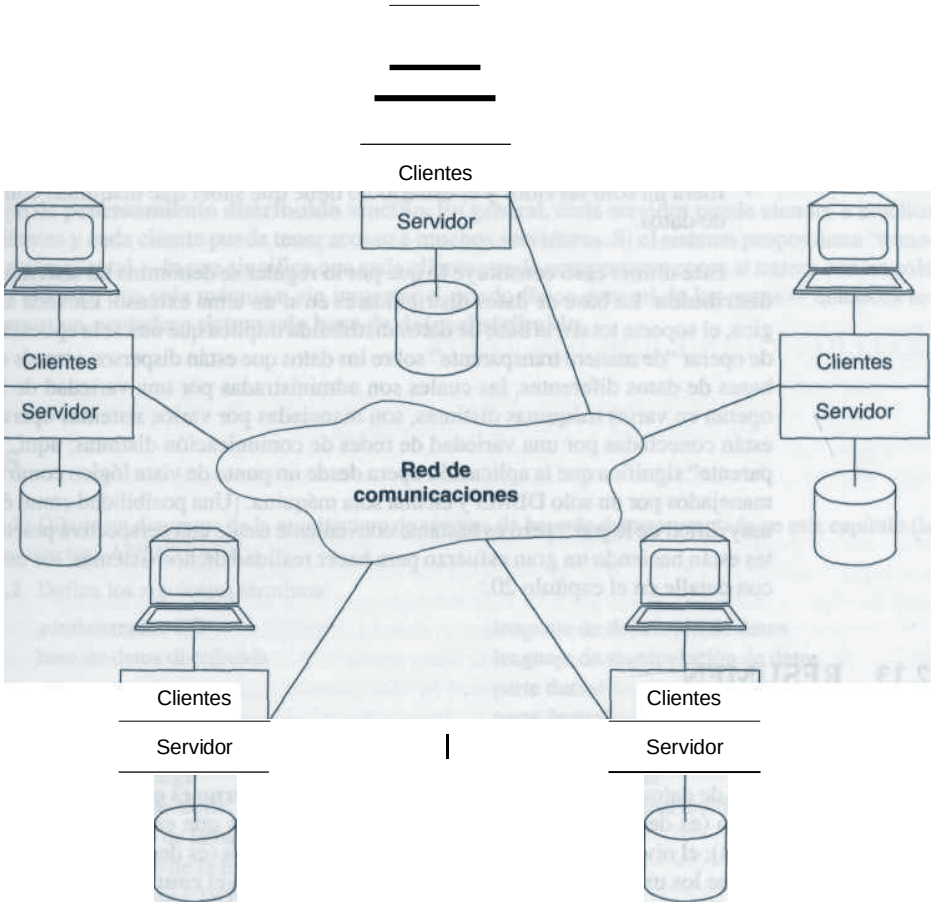

Figura 2.8 Cada máquina opera como clientes y como servidor.

La idea final es que una sola máquina cliente podría ser capaz de acceder a varias máquinas servidor diferentes (lo contrario al caso ilustrado en la figura 2.7). Esta posibilidad es conveniente ya que, como mencioné antes, las empresas operan por lo regular de tal manera que la totalidad de sus datos no están almacenados en una sola máquina, sino más bien están esparcidos a través de muchas máquinas distintas, y las aplicaciones necesitarán a veces tener la posibilidad de acceder a los datos de más de una máquina. Básicamente, este acceso puede ser proporcionado en dos formas:

Una máquina cliente dada podría ser capaz de acceder a cualquier cantidad de servidores, pero sólo uno a la vez (es decir, cada petición individual de base de datos debe ser dirigida

a un solo servidor). En un sistema así, no es posible combinar datos de dos o más servidores con una sola petición. Además, el usuario de dicho sistema debe saber qué máquina en particular contiene qué piezas de datos.

- El cliente podría ser capaz de acceder a varios servidores en forma simultánea (es decir, una sola petición de base de datos podría combinar datos de varios servidores). En este caso, los servidores ven al cliente —desde un punto de vista lógico— como si en realidad fuera un solo servidor y el usuario no tiene que saber qué máquinas contienen qué piezas de datos.

Este último caso constituye lo que por lo regular se denomina un **sistema de** base de datos **distribuida.** La base de datos distribuida es en sí un tema extenso. Llevada a su conclusión lógica, el soporte total a la base de datos distribuida implica que una sola aplicación debe ser capaz de operar "de manera transparente" sobre los datos que están dispersos a través de una variedad de bases de datos diferentes, las cuales son administradas por una variedad de DBMSs distintos. operan en varias máquinas distintas, son manejadas por varios sistemas operativos diferentes y están conectadas por una variedad de redes de comunicación distintas; aquí, "de manera transparente" significa que la aplicación opera desde un punto de vista lógico como si los datos fueran manejados por un solo DBMS y en una sola máquina. ¡Una posibilidad como ésta podría parecer muy difícil de lograr!; pero es bastante conveniente desde una perspectiva práctica, y los fabricantes están haciendo un gran esfuerzo para hacer realidad dichos sistemas, los cuales explicaremos con detalle en el capítulo 20.

2.13 RESUMEN

En este capítulo hemos visto los sistemas de bases de datos desde el punto de vista de la arquitectura. Primero describimos la **arquitectura ANSI/SPARC,** la cual divide a un sistema de base de datos en tres niveles, como sigue: El nivel **interno** es el más cercano al almacenamiento físico (es decir, es aquel que se ocupa de la forma en que están almacenados físicamente los datos); el nivel **externo** es el más cercano a los usuarios (es decir, es el que se ocupa de la forma en que los usuarios individuales ven los datos); y el nivel **conceptual** es un nivel de indirección entre los otros dos (proporciona una *vista comunitaria* de los datos). Los datos, como se perciben en cada nivel, están descritos por medio de un **esquema** (o por varios esquemas, en el caso del nivel externo). Las **transformaciones** definen la correspondencia entre (a) un esquema externo dado y el esquema conceptual y (b) el esquema conceptual y el esquema interno. Estas transformaciones son la clave para proporcionar la **independencia lógica** y **física de** los datos, respectivamente.

Los usuarios —es decir, los usuarios finales y los programadores de aplicaciones, los cuales operan al nivel externo— interactúan con los datos por medio de un **sublenguaje** de datos, el cual se divide por lo menos en dos componentes: un DDL **(lenguaje de definición de** datos) y un DML **(lenguaje de manipulación de datos).** El sublenguaje de datos está incrustado en un **lenguaje** anfitrión. *Nota:* Los límites entre el lenguaje anfitrión y el sublenguaje de datos, y entre el DDL y el DML, son principalmente de naturaleza conceptual; en forma ideal, deberían ser "transparentes para el usuario".

También vimos más de cerca las funciones del **DBA** y del **DBMS.** Entre otras cosas, el DBA es el responsable de crear el esquema interno (el **diseño físico de la base de datos);** en contraste,

la creación del esquema conceptual (el **diseño lógico** o **conceptual de la base de datos)** es responsabilidad del administrador de *datos.* Y el DBMS es responsable, entre otras cosas, de implementar las peticiones DDL y DML del usuario. El DBMS también es responsable de proporcionar cierto tipo de función de **diccionario de datos.**

Otra forma conveniente de ver a los sistemas de bases de datos es como si consistieran en un **servidor** (el propio DBMS) y un conjunto de **clientes** (las aplicaciones). Los clientes y servidores pueden operar, y a menudo lo harán, en máquinas independientes, proporcionando así un tipo de **procesamiento distribuido** sencillo. En general, cada servidor puede atender a muchos clientes y cada cliente puede tener acceso a muchos servidores. Si el sistema proporciona "transparencia" total —lo que significa que cada cliente puede comportarse como si tratara con un solo servidor en una sola máquina, sin importar el estado físico general de las cosas— entonces tenemos un verdadero **sistema de base de datos distribuida.**

EJERCICIOS

2.1 Dibuje un diagrama de la arquitectura de sistema de base de datos presentada en este capítulo (la arquitectura ANSI/SPARC).

2.2 Defina los siguientes términos:

administrador CD
base de datos distribuida
carga cliente
definición de la estructura de almacenamiento
descarga/recarga diccionario de datos diseño físico de la base de datos diseño lógico de la base de datos esquema, vista y DDL conceptuales esquema, vista y DDL externos esquema, vista y DDL internos interfaz de usuario
lenguaje anfitrión
lenguaje de definición de datos
lenguaje de manipulación de datos
parte dorsal parte frontal
petición no planeada
petición planeada
procesamiento distribuido
reorganización
servidor
sistema BD/CD
sublenguaje de datos
transformación conceptual/interna
transformación externa/conceptual
utilería

2.3 Explique la secuencia de pasos comprendidos en la recuperación de una ocurrencia de un registro externo específico.

2.4 Liste las principales funciones que realiza el DBMS.

2.5 Haga una distinción entre la independencia lógica y física de los datos.

2.6 ¿Qué entiende por el término *metadatos?*

2.7 Liste las principales funciones que realiza el DBA.

2.8 Haga una distinción entre el DBMS y un sistema de administración de archivos.

2.9 Dé algunos ejemplos de herramientas proporcionadas por el fabricante.

2.10 Dé algunos ejemplos de utilerías de base de datos.

2.11 Examine cualquier sistema de base de datos que pudiera tener disponible. Procure transformar ese sistema a la arquitectura ANSI/SPARC tal como se describe en este capítulo. ¿Maneja claramente los tres niveles de la arquitectura? ¿Cómo están definidas las transformaciones entre niveles? ¿Cómo lucen los diversos DDLs (externo, conceptual, interno)? ¿Qué sublenguajes de datos maneja el sistema? ¿Qué lenguajes host? ¿Quién realiza la función de DBA? ¿Existen algunas herramientas de seguri dad y de integridad? ¿Hay un diccionario de datos? ¿Se describe éste a sí mismo? ¿Qué aplicaciones proporcionadas por el fabricante soporta el sistema? ¿Qué utilerías? ¿Hay un administrador indepen diente de comunicaciones de datos ? ¿Existe alguna posibilidad de procesamiento distribuido?

REFERENCIAS Y BIBLIOGRAFÍA

Las referencias siguientes son a la fecha bastante antiguas (con excepción de la última), pero aún son importantes para los conceptos presentados en este capítulo.

2.1 ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report, FD7(bulletin of ACM SIGMOD) 7, No. 2 (1975).

2.2 Dionysios C. Tsichritzis y Anthony Klug (eds.): "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems", *Information Systems 3* (1978).

Las referencias [2.1-2.2] corresponden a los informes provisional y final, respectivamente, del tan mencionado Grupo de Estudio ANSI/SPARC. El Grupo de Estudio sobre Sistemas de Administración de Base de Datos ANSI/X3/SPARC (para dar su nombre completo), fue establecido a finales de 1972 por el SPARC (Comité de Planeación de Estándares y Requerimientos) del Comité sobre Computadoras y Procesamiento de Información (X3) del ANSI (Instituto Nacional Americano de Estándares). (Alrededor de 25 años más tarde, el nombre X3 se cambió por NCITS: Comité Nacional sobre Estándares en Tecnología de la Información.) Los objetivos del Grupo de Estudio fueron determinar qué áreas, si las había, de la tecnología de base de datos eran las adecuadas para estandarizar y producir un conjunto de recomendaciones de acción en cada una de ellas. Al trabajar para alcanzar estos objetivos, el Grupo de Estudio adoptó la posición de que las *interfaces* eran el único aspecto de un sistema de base de datos que podría ser sujeto a la estandarización y de acuerdo con ello, definió una arquitectura generalizada de sistema de base de datos, o infraestructura, que enfatizaba el papel de dichas interfaces. El informe final ofrece una descripción detallada de esa arquitectura y de algunas de las 42 (!) interfaces identificadas. El informe provisional es un documento de trabajo anterior que sigue teniendo cierto interés; proporciona detalles adicionales en algunas áreas.

2.3 J. J. van Griethuysen (ed.): "Concepts and Terminology for the Conceptual Schema and the In formation Base", International Organization for Standardization (ISO) Technical Report ISO/TR 9007:1987(E) (marzo, 1982; revisado en julio, 1987).

Este documento es un informe de un grupo de trabajo de la ISO (Organización Internacional de Estándares) cuyos objetivos comprendían "la definición de conceptos para lenguajes de esquema conceptual". Incluye una introducción a los tres candidatos en competencia (para ser más precisos, a los tres *conjuntos* de candidatos) para un conjunto apropiado de formalismos, y aplica cada uno de los tres a un ejemplo común que comprende las actividades de una autoridad ficticia de registro de automóviles. Los tres conjuntos de competidores son (1) esquemas de "entidad-atribu-

to-vínculo", (2) esquemas de "vínculo binario" y (3) esquemas de "lógica de predicados interpretados". El informe también incluye una exposición de los conceptos fundamentales subyacentes a la noción del esquema conceptual y sugiere algunos principios como base para la implementatión de un sistema que soporte en forma adecuada dicha noción. Aunque en ocasiones resulta denso, es un documento importante para todo el que se interese seriamente en el nivel conceptual del sistema.

2.4 William Kent: *Data and Reality.* Amsterdam, Netherlands: North-Holland/NuevaYork, N.Y.: Elsevier Science (1978).

Una explicación estimulante e incitante a la reflexión sobre la naturaleza de la información y en particular, sobre el esquema conceptual. "Este libro proyecta una filosofía de que la vida y la realidad son en el fondo amorfas, desordenadas, contradictorias, inconsistentes, irracionales y no objetivas" (extracto del último capítulo). Se puede ver al libro en gran parte como un compendio de problemas reales con los que (se sugiere) los formalismos de las bases de datos existentes tienen dificultades para enfrentar; en particular, los formalismos que se basan en estructuras del tipo de registro convencional, que incluyen al modelo relational. Recomendado.

2.5 Odysseas G. Tsatalos, Marvin H. Solomon y Yannis E. Ioannidis: "The GMAP: A Versatile Tool for Physical Data Independence", Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile (septiembre, 1994).

Las siglas GMAP corresponden a *Ruta de Acceso Generalizada de Nivel Múltiple.* Los autores de este artículo señalan correctamente que los productos de base de datos actuales "obligan a los usuarios a encasillar sus consultas en términos de un esquema lógico que está ligado en forma directa a las estructuras físicas", de ahí que sean más bien débiles en cuanto a la independencia de los datos. Por lo tanto, en el artículo proponen un lenguaje de transformación conceptual/interna (para emplear la terminología del presente capítulo), que puede ser usada para especificar muchas más clases de transformaciones que las que se manejan generalmente en los productos actuales. Dado un "esquema lógico, en particular el lenguaje (que está basado en el álgebra relational —vea el capítulo 6— y por lo tanto es de naturaleza declarativa, no de procedimientos) permite la especificación de numerosos esquemas físicos diferentes, todos ellos derivados formalmente de un esquema lógico. Entre otras cosas, dichos esquemas físicos pueden incluir particiones verticales y horizontales (vea el capítulo 20), cualquier cantidad de rutas de acceso físico, agrupamiento y redundancia controlada.

El artículo también ofrece un algoritmo para transformar las operaciones del usuario, frente al esquema lógico, en operaciones equivalentes, frente al esquema físico. Una implementatión prototipo muestra que el DBA puede ajustar el esquema físico para lograr "un rendimiento significativamente mejor al que es posible obtener con las técnicas convencionales".