Apuntes de
Ficheros y Bases de Datos

Mercedes Marqués

10 de febrero de 2001

Prefacio

El objetivo de este texto es servir como material de apoyo para los estudiantes que cursan la asignatura de *Ficheros y Bases de Datos* de la *Ingeniería Técnica en Informática de Gestión* de la *Universitat Jaume I*.

En esta asignatura se introducen los sistemas de bases de datos como el modo fundamental de organizar los datos en los sistemas de información. Se centra el estudio en los sistemas de bases de datos relacionales, ya que son los más extendidos hoy en día y poseen una sólida base teórica que los sustenta. Además de aprender el uso y funcionamiento de los sistemas relacionales, se estudia una metodología de diseño que es también la más utilizada, tanto en el ámbito profesional como en el académico.

Para la elaboración de estos apuntes se ha utilizado la bibliografía que aparece al final, toda ella disponible en el catálogo de la biblioteca de la propia universidad.

Cualquier sugerencia, comentario o corrección se puede enviar por correo electrónico a `mmarques@inf.uji.es`

Índice General

Capítulo 1

Introducción

En este capítulo se presentan los sistemas de bases de datos, haciendo antes un repaso por sus predecesores, los sistemas de ficheros. Aunque los sistemas de ficheros se han quedado obsoletos, hay dos buenas razones para estudiarlos. En primer lugar, el conocer los problemas de este tipo de sistemas nos previene de volver a cometerlos. En segundo lugar, si en algún momento fuera necesario convertir un sistema de ficheros en un sistema de bases de datos, comprender cómo trabaja este sistema puede ser una ayuda esencial.

1.1 Sistemas de ficheros

Un *sistema de ficheros* es un conjunto de programas que prestan servicio a los usuarios finales. Cada programa define y maneja sus propios datos.

Los sistemas de ficheros surgieron al tratar de informatizar el manejo de los archivadores manuales con objeto de proporcionar un acceso más eficiente a los datos.

En lugar de establecer un sistema centralizado en donde almacenar todos los datos de la organización o empresa, se escogió un modelo descentralizado en el que cada sección o departamento almacena y gestiona sus propios datos. Para comprender esto vamos a utilizar como ejemplo una empresa inmobiliaria cuya descripción completa se encuentra en el último apartado de este capítulo.

En esta inmobiliaria, el *departamento de ventas* se encarga de alquilar inmuebles. Por ejemplo, cuando un propietario pasa por el departamento de ventas para ofrecer en alquiler su piso, se rellena un formulario en donde se recogen los datos del piso, como la dirección y el número de habitaciones, y los datos del propietario. El departamento de ventas también se encarga de atender a los clientes que desean alquilar un inmueble. Cuando un cliente (posible inquilino) pasa por este departamento se rellena un formulario con sus datos y sus preferencias: si quiere un piso o una casa, el importe mensual que está dispuesto a pagar por el alquiler, etc. Para gestionar toda esta información, el departamento de ventas posee un sistema de información. El sistema tiene tres ficheros: fichero de inmuebles, fichero de propietarios y fichero de inquilinos.

INMUEBLE

Inum	Calle	Area	Población	Tipo	Hab	Alquiler	Pnum
IA14	Enmedio, 128	Centro	Castellón	Casa	6	600	P46
IL94	Riu Ebre, 24	Ronda Sur	Castellón	Piso	4	350	P87
IG4	Sorell, 5	Grao	Castellón	Piso	3	300	P40
IG36	Alicante,1		Segorbe	Piso	3	325	P93
IG21	San Francisco, 10		Vinaroz	Casa	5	550	P87
IG16	Capuchinos, 19	Rafalafena	Castellón	Piso	4	400	P93

PROPIETARIO

Pnum	Nombre	Apellido	Dirección	Pref	Teléfono
P46	Amparo	Felip	Asensi 24, Castellón	964	230 680
P87	Manuel	Obiol	Av.Libertad 15, Vinaroz	964	450 760
P40	Alberto	Estrada	Av.del Puerto 52, Castellón	964	200 740
P93	Yolanda	Robles	Purísima 4, Segorbe	964	710 430

INQUILINO

Qnum	Nombre	Apellido	Dirección	Pref	Teléfono	Tipo	Alquiler
Q76	Juan	Felip	Barceló 47, Castellón	964	282 540	Piso	375
Q56	Ana	Grangel	San Rafael 45, Almazora	964	551 110	Piso	300
Q74	Elena	Abaso	Navarra 76, Castellón	964	205 560	Casa	700
Q62	Alicia	Mori	Alloza 45, Castellón	964	229 580	Piso	550

El *departamento de contratos* se ocupa de gestionar los contratos de alquiler de los inmuebles. Cuando un cliente desea formalizar un contrato, un empleado de la empresa

rellena un formulario con los datos del inquilino y los datos del inmueble. Este formulario se pasa al departamento de contratos, que asigna un número al contrato y completa la información sobre el pago y el período del contrato. Para gestionar esta información, el departamento de contratos posee un sistema de información con tres ficheros: el fichero de los contratos, el fichero de los inmuebles alquilados y el fichero de los inquilinos que tienen en vigor un contrato de alquiler.

CONTRATO

Cnum	Inum	Qnum	Importe	Pago	Depósito	Pagado?	Inicio	Fin	Meses
10024	IA14	Q62	600	Visa	1200	S	1/6/99	31/5/00	12
10075	IL94	Q76	350	Efectivo	700	N	1/1/00	30/6/00	6
10012	IG21	Q74	550	Cheque	1100	S	1/7/99	30/6/00	12

INMUEBLE

Inum	Calle	Area	Población	Alquiler
IA14	Enmedio, 128	Centro	Castellón	600
IL94	Riu Ebre, 24	Ronda Sur	Castellón	350
IG21	San Francisco, 10		Vinaroz	550

INQUILINO

Qnum	Nombre	Apellido	Dirección	Población	Teléfono
Q76	Juan	Felip	Barceló, 47	Castellón	964 282 540
Q74	Elena	Abaso	Navarra, 76	Castellón	964 205 560
Q62	Alicia	Mori	Alloza, 45	Castellón	964 229 580

Cada departamento accede a sus propios ficheros mediante una serie de programas de aplicación escritos especialmente para ellos. Estos programas son totalmente independientes entre un departamento y otro, y se utilizan para introducir datos, mantener los ficheros y generar los informes que cada departamento necesita. Es importante destacar que la estructura física de los ficheros de datos y de sus registros está definida dentro de los programas de aplicación.

La situación es muy similar en el resto de departamentos. En el *departamento de nóminas* tienen un fichero con los datos de los salarios de los empleados. Los registros de este fichero tienen los siguientes campos: número de empleado, nombre, apellido, dirección, fecha de nacimiento, salario, DNI y número de la oficina en la que trabaja. El *departamento de personal*

tiene un fichero con los datos de los empleados. Sus registros tienen los siguientes campos: número de empleado, nombre, apellidos, dirección, teléfono, puesto, fecha de nacimiento, salario, DNI y número de la oficina en la que trabaja.

Se puede ver claramente que hay una gran cantidad de datos repetidos en los ficheros de estos departamentos, algo que siempre ocurre en los sistemas de ficheros. A raíz de esto, los sistemas de ficheros presentan una serie de inconvenientes:

- *Separación y aislamiento de los datos.* Cuando los datos se separan en distintos ficheros, es más complicado acceder a ellos, ya que el programador de aplicaciones debe sincronizar el procesamiento de los distintos ficheros implicados para asegurar que se extraen los datos correctos.

- *Duplicación de datos.* La redundancia de datos existente en los sistemas de ficheros hace que se desperdicie espacio de almacenamiento y lo que es más importante: puede llevar a que se pierda la consistencia de los datos. Se produce una inconsistencia cuando copias de los mismos datos no coinciden.

- *Dependencia de datos.* Ya que la estructura física de los datos (la definición de los ficheros y de los registros) se encuentra codificada en los programas de aplicación, cualquier cambio en dicha estructura es difícil de realizar. El programador debe identificar todos los programas afectados por este cambio, modificarlos y volverlos a probar, lo que cuesta mucho tiempo y está sujeto a que se produzcan errores. A este problema, tan característico de los sistemas de ficheros, se le denomina también *falta de independencia de datos lógica–física.*

- *Formatos de ficheros incompatibles.* Ya que la estructura de los ficheros se define en los programas de aplicación, es completamente dependiente del lenguaje de programación. La incompatibilidad entre ficheros generados por distintos lenguajes hace que los ficheros sean difíciles de procesar de modo conjunto.

- *Consultas fijas y proliferación de programas de aplicación.* Desde el punto de vista de los usuarios finales, los sistemas de ficheros fueron un gran avance comparados a los sistemas manuales. A consecuencia de esto, creció la necesidad de realizar distintos tipos de consultas de datos. Sin embargo, los sistemas de ficheros son muy dependientes

del programador de aplicaciones: cualquier consulta o informe que se quiera realizar debe ser programado por él. En algunas organizaciones se conformaron con fijar el tipo de consultas e informes, siendo imposible realizar otro tipo de consultas que no se hubieran tenido en cuenta a la hora de escribir los programas de aplicación.

En otras organizaciones hubo una proliferación de programas de aplicación para resolver todo tipo de consultas, hasta el punto de desbordar al departamento de proceso de datos, que no daba abasto para validar, mantener y documentar dichos programas.

1.2 Sistemas de bases de datos

Los inconvenientes de los sistemas de ficheros se pueden atribuir a dos factores:

- La definición de los datos se encuentra codificada dentro de los programas de aplicación, en lugar de estar almacenada aparte y de forma independiente.
- No hay control sobre el acceso y la manipulación de los datos más allá de lo impuesto por los programas de aplicación.

Para trabajar de un modo más efectivo, surgieron las *bases de datos* y los *sistemas de gestión de bases de datos (SGBD)*.

Una *base de datos* es un conjunto de datos almacenados entre los que existen relaciones lógicas y ha sido diseñada para satisfacer los requerimientos de información de una empresa u organización. En una base de datos, además de los datos, también se almacena su descripción.

La base de datos es un gran almacén de datos que se define una sola vez y que se utiliza al mismo tiempo por muchos departamentos y usuarios. En lugar de trabajar con ficheros desconectados e información redundante, todos los datos se integran con una mínima cantidad de duplicidad. La base de datos no pertenece a un departamento, se comparte por toda la organización. Además, la base de datos no sólo contiene los datos de la organización, también almacena una descripción de dichos datos. Esta descripción es lo que se denomina *metadatos*, se almacena en el *diccionario de datos* o catálogo y es lo que permite que exista independencia de datos lógica–física.

El modelo seguido con los sistemas de bases de datos, en donde se separa la definición de los datos de los programas de aplicación, es muy similar al modelo que se sigue en la actualidad para el desarrollo de programas, en donde se da una definición interna de un objeto y una definición externa separada. Los usuarios del objeto sólo ven la definición externa y no se deben preocupar de cómo se define internamente el objeto y cómo funciona. Una ventaja de este modelo, conocido como abstracción de datos, es que se puede cambiar la definición interna de un objeto sin afectar a sus usuarios ya que la definición externa no se ve alterada. Del mismo modo, los sistemas de bases de datos separan la definición de la estructura de los datos, de los programas de aplicación y almacenan esta definición en la base de datos. Si se añaden nuevas estructuras de datos o se modifican las ya existentes, los programas de aplicación no se ven afectados ya que no dependen directamente de aquello que se ha modificado.

El *sistema de gestión de la base de datos (SGBD)* es una aplicación que permite a los usuarios definir, crear y mantener la base de datos, y proporciona acceso controlado a la misma.

El SGBD es la aplicación que interacciona con los usuarios de los programas de aplicación y la base de datos. En general, un SGBD proporciona los siguientes servicios:

- Permite la definición de la base de datos mediante el *lenguaje de definición de datos*. Este lenguaje permite especificar la estructura y el tipo de los datos, así como las restricciones sobre los datos. Todo esto se almacenará en la base de datos.

- Permite la inserción, actualización, eliminación y consulta de datos mediante el *lenguaje de manejo de datos*. El hecho de disponer de un lenguaje para realizar consultas reduce el problema de los sistemas de ficheros, en los que el usuario tiene que trabajar con un conjunto fijo de consultas, o bien, dispone de un gran número de programas de aplicación costosos de gestionar.

  Hay dos tipos de lenguajes de manejo de datos: los *procedurales* y los *no procedurales*. Estos dos tipos se distinguen por el modo en que acceden a los datos. Los lenguajes procedurales manipulan la base de datos registro a registro, mientras que los no procedurales operan sobre conjuntos de registros. En los lenguajes procedurales se especifica qué operaciones se deben realizar para obtener los datos resultado, mientras

que en los lenguajes no procedurales se especifica qué datos deben obtenerse sin decir cómo hacerlo. El lenguaje no procedural más utilizado es el SQL (Structured Query Language) que, de hecho, es un estándar y es el lenguaje de los SGBD relacionales.

- Proporciona un acceso controlado a la base de datos mediante:
  - un sistema de seguridad, de modo que los usuarios no autorizados no puedan acceder a la base de datos;
  - un sistema de integridad que mantiene la integridad y la consistencia de los datos;
  - un sistema de control de concurrencia que permite el acceso compartido a la base de datos;
  - un sistema de control de recuperación que restablece la base de datos después de que se produzca un fallo del *hardware* o del *software*;
  - un diccionario de datos o catálogo accesible por el usuario que contiene la descripción de los datos de la base de datos.

A diferencia de los sistemas de ficheros, el SGBD gestiona la estructura física de los datos y su almacenamiento. Con esta funcionalidad, el SGBD se convierte en una herramienta de gran utilidad. Sin embargo, desde el punto de vista del usuario, se podría discutir que los SGBD han hecho las cosas más complicadas, ya que ahora los usuarios ven más datos de los que realmente quieren o necesitan, puesto que ven la base de datos completa. Conscientes de este problema, los SGBD proporcionan un mecanismo de *vistas* que permite que cada usuario tenga su propia vista o visión de la base de datos. El lenguaje de definición de datos permite definir vistas como subconjuntos de la base de datos.

Las vistas, además de reducir la complejidad permitiendo que cada usuario vea sólo la parte de la base de datos que necesita, tienen otras ventajas:

- Las vistas proporcionan un nivel de seguridad, ya que permiten excluir datos para que ciertos usuarios no los vean.
- Las vistas proporcionan un mecanismo para que los usuarios vean los datos en el formato que deseen.

- Una vista representa una imagen consistente y permanente de la base de datos, incluso si la base de datos cambia su estructura.

Todos los SGBD no presentan la misma funcionalidad, depende de cada producto. En general, los grandes SGBD multiusuario ofrecen todas las funciones que se acaban de citar y muchas más. Los sistemas modernos son conjuntos de programas extremadamente complejos y sofisticados, con millones de líneas de código y con una documentación consistente en varios volúmenes. Lo que se pretende es proporcionar un sistema que permita gestionar cualquier tipo de requisitos y que tenga un 100% de fiabilidad ante cualquier fallo *hardware* o *software*. Los SGBD están en continua evolución, tratando de satisfacer los requerimientos de todo tipo de usuarios. Por ejemplo, muchas aplicaciones de hoy en día necesitan almacenar imágenes, vídeo, sonido, etc. Para satisfacer a este mercado, los SGBD deben cambiar. Conforme vaya pasando el tiempo irán surgiendo nuevos requisitos, por lo que los SGBD nunca permanecerán estáticos.

1.3 Papeles en el entorno de las bases de datos

Hay cuatro grupos de personas que intervienen en el entorno de una base de datos: el administrador de la base de datos, los diseñadores de la base de datos, los programadores de aplicaciones y los usuarios.

El *administrador de la base de datos* se encarga del diseño físico de la base de datos y de su implementación, realiza el control de la seguridad y de la concurrencia, mantiene el sistema para que siempre se encuentre operativo y se encarga de que los usuarios y las aplicaciones obtengan buenas prestaciones. El administrador debe conocer muy bien el SGBD que se esté utilizando, así como el equipo informático sobre el que esté funcionando.

Los *diseñadores de la base de datos* realizan el diseño lógico de la base de datos, debiendo identificar los datos, las relaciones entre datos y las restricciones sobre los datos y sus relaciones. El diseñador de la base de datos debe tener un profundo conocimiento de los datos de la empresa y también debe conocer sus *reglas de negocio*. Las reglas de negocio describen las características principales de los datos tal y como las ve la empresa.

Para obtener un buen resultado, el diseñador de la base de datos debe implicar en el desarrollo del modelo de datos a todos los usuarios de la base de datos, tan pronto como sea posible. El diseño lógico de la base de datos es independiente del SGBD concreto que se vaya a utilizar, es independiente de los programas de aplicación, de los lenguajes de programación y de cualquier otra consideración física.

Una vez se ha diseñado e implementado la base de datos, los *programadores de aplicaciones* se encargan de implementar los programas de aplicación que servirán a los usuarios finales. Estos programas de aplicación son los que permiten consultar datos, insertarlos, actualizarlos y eliminarlos. Estos programas se escriben mediante lenguajes de tercera generación o de cuarta generación.

Los *usuarios finales* son los "clientes"de la base de datos: la base de datos ha sido diseñada e implementada, y está siendo mantenida, para satisfacer sus requisitos en la gestión de su información.

1.4 Historia de los sistemas de bases de datos

Como se ha visto en este capítulo, los predecesores de los sistemas de bases de datos fueron los sistemas de ficheros. No hay un momento concreto en que los sistemas de ficheros hayan cesado y hayan dado comienzo los sistemas de bases de datos. De hecho, todavía existen sistemas de ficheros en uso.

Se dice que los sistemas de bases de datos tienen sus raíces en el proyecto estadounidense Apolo de mandar al hombre a la luna, en los años sesenta. En aquella época, no había ningún sistema que permitiera gestionar la inmensa cantidad de información que requería el proyecto. La primera empresa encargada del proyecto, NAA (North American Aviation), desarrolló un *software* denominado GUAM (General Update Access Method) que estaba basado en el concepto de que varias piezas pequeñas se unen para formar una pieza más grande, y así sucesivamente hasta que el producto final está ensamblado. Esta estructura, que tiene la forma de un árbol, es lo que se denomina una *estructura jerárquica*. A mediados de los sesenta, IBM se unió a NAA para desarrollar GUAM en lo que ahora se conoce como IMS (Information Management System). El motivo por el cual IBM restringió IMS al manejo de

jerarquías de registros fue el de permitir el uso de dispositivos de almacenamiento serie, más exactamente las cintas magnéticas, ya que era un requisito del mercado por aquella época.

A mitad de los sesenta, se desarrolló IDS (Integrated Data Store), de General Electric. Este trabajo fue dirigido por uno de los pioneros en los sistemas de bases de datos, Charles Bachmann. IDS era un nuevo tipo de sistema de bases de datos conocido como *sistema de red*, que produjo un gran efecto sobre los sistemas de información de aquella generación. El sistema de red se desarrolló, en parte, para satisfacer la necesidad de representar relaciones entre datos más complejas que las que se podían modelar con los sistemas jerárquicos, y, en parte, para imponer un estándar de bases de datos. Para ayudar a establecer dicho estándar, CODASYL (Conference on Data Systems Languages), formado por representantes del gobierno de EEUU y representantes del mundo empresarial, formaron un grupo denominado DBTG (Data Base Task Group), cuyo objetivo era definir unas especificaciones estándar que permitieran la creación de bases de datos y el manejo de los datos. El DBTG presentó su informe final en 1971 y aunque éste no fue formalmente aceptado por ANSI (American National Standards Institute), muchos sistemas se desarrollaron siguiendo la propuesta del DBTG. Estos sistemas son los que se conocen como sistemas de red, o sistemas CODASYL o DBTG.

Los sistemas jerárquico y de red constituyen la primera generación de los SGBD. Pero estos sistemas presentan algunos inconvenientes:

- Es necesario escribir complejos programas de aplicación para responder a cualquier tipo de consulta de datos, por simple que ésta sea.
- La independencia de datos es mínima.
- No tienen un fundamento teórico.

En 1970 Codd, de los laboratorios de investigación de IBM, escribió un artículo presentando el *modelo relacional*. En este artículo, presentaba también los inconvenientes de los sistemas previos, el jerárquico y el de red. Entonces, se comenzaron a desarrollar muchos sistemas relacionales, apareciendo los primeros a finales de los setenta y principios de los ochenta. Uno de los primeros es System R, de IBM, que se desarrolló para probar la funcionalidad del modelo relacional, proporcionando una implementación de sus estructuras de

datos y sus operaciones. Esto condujo a dos grandes desarrollos:

- El desarrollo de un lenguaje de consultas estructurado denominado SQL, que se ha convertido en el lenguaje estándar de los sistemas relacionales.
- La producción de varios SGBD relacionales durante los años ochenta, como DB2 y SLQ/DS de IBM, y ORACLE de ORACLE Corporation.

Hoy en día, existen cientos de SGBD relacionales, tanto para microordenadores como para sistemas multiusuario, aunque muchos no son completamente fieles al modelo relacional.

Otros sistemas relacionales multiusuario son INGRES de Computer Associates, Informix de Informix Software Inc. y Sybase de Sybase Inc. Ejemplos de sistemas relacionales de microordenadores son Paradox y dBase IV de Borland, Access de Microsoft, FoxPro y R:base de Microrim.

Los SGBD relacionales constituyen la segunda generación de los SGBD. Sin embargo, el modelo relacional también tiene sus fallos, siendo uno de ellos su limitada capacidad al modelar los datos. Se ha hecho mucha investigación desde entonces tratando de resolver este problema. En 1976, Chen presentó el modelo entidad–relación, que es la técnica más utilizada en el diseño de bases de datos. En 1979, Codd intentó subsanar algunas de las deficiencias de su modelo relacional con una versión extendida denominada RM/T (1979) y más recientemente RM/V2 (1990). Los intentos de proporcionar un modelo de datos que represente al mundo real de un modo más fiel han dado lugar a los modelos de datos semánticos.

Como respuesta a la creciente complejidad de las aplicaciones que requieren bases de datos, han surgido dos nuevos modelos: el modelo de datos orientado a objetos y el modelo relacional extendido. Sin embargo, a diferencia de los modelos que los preceden, la composición de estos modelos no está clara. Esta evolución representa la tercera generación de los SGBD.

1.5 Ventajas e inconvenientes de los sistemas de bases de datos

Los sistemas de bases de datos presentan numerosas ventajas que se pueden dividir en dos grupos: las que se deben a la integración de datos y las que se deben a la interface común que proporciona el SGBD.

Ventajas por la integración de datos

- *Control sobre la redundancia de datos.* Los sistemas de ficheros almacenan varias copias de los mismos datos en ficheros distintos. Esto hace que se desperdicie espacio de almacenamiento, además de provocar la falta de consistencia de datos. En los sistemas de bases de datos todos estos ficheros están integrados, por lo que no se almacenan varias copias de los mismos datos. Sin embargo, en una base de datos no se puede eliminar la redundancia completamente, ya que en ocasiones es necesaria para modelar las relaciones entre los datos, o bien es necesaria para mejorar las prestaciones.

- *Consistencia de datos.* Eliminando o controlando las redundancias de datos se reduce en gran medida el riesgo de que haya inconsistencias. Si un dato está almacenado una sola vez, cualquier actualización se debe realizar sólo una vez, y está disponible para todos los usuarios inmediatamente. Si un dato está duplicado y el sistema conoce esta redundancia, el propio sistema puede encargarse de garantizar que todas las copias se mantienen consistentes. Desgraciadamente, no todos los SGBD de hoy en día se encargan de mantener automáticamente la consistencia.

- *Más información sobre la misma cantidad de datos.* Al estar todos los datos integrados, se puede extraer información adicional sobre los mismos.

- *Compartición de datos.* En los sistemas de ficheros, los ficheros pertenecen a las personas o a los departamentos que los utilizan. Pero en los sistemas de bases de datos, la base de datos pertenece a la empresa y puede ser compartida por todos los usuarios que estén autorizados. Además, las nuevas aplicaciones que se vayan creando pueden utilizar los datos de la base de datos existente.

- *Mantenimiento de estándares.* Gracias a la integración es más fácil respetar los estándares necesarios, tanto los establecidos a nivel de la empresa como los nacionales e internacionales. Estos estándares pueden establecerse sobre el formato de los datos para facilitar su intercambio, pueden ser estándares de documentación, procedimientos de actualización y también reglas de acceso.

Ventajas por la existencia del SGBD

- *Mejora en la integridad de datos.* La integridad de la base de datos se refiere a la validez y la consistencia de los datos almacenados. Normalmente, la integridad se expresa mediante restricciones o reglas que no se pueden violar. Estas restricciones se pueden aplicar tanto a los datos, como a sus relaciones, y es el SGBD quien se debe encargar de mantenerlas.

- *Mejora en la seguridad.* La seguridad de la base de datos es la protección de la base de datos frente a usuarios no autorizados. Sin unas buenas medidas de seguridad, la integración de datos en los sistemas de bases de datos hace que éstos sean más vulnerables que en los sistemas de ficheros. Sin embargo, los SGBD permiten mantener la seguridad mediante el establecimiento de claves para identificar al personal autorizado a utilizar la base de datos. Las autorizaciones se pueden realizar a nivel de operaciones, de modo que un usuario puede estar autorizado a consultar ciertos datos pero no a actualizarlos, por ejemplo.

- *Mejora en la accesibilidad a los datos.* Muchos SGBD proporcionan lenguajes de consultas o generadores de informes que permiten al usuario hacer cualquier tipo de consulta sobre los datos, sin que sea necesario que un programador escriba una aplicación que realice tal tarea.

- *Mejora en la productividad.* El SGBD proporciona muchas de las funciones estándar que el programador necesita escribir en un sistema de ficheros. A nivel básico, el SGBD proporciona todas las rutinas de manejo de ficheros típicas de los programas de aplicación. El hecho de disponer de estas funciones permite al programador centrarse mejor en la función específica requerida por los usuarios, sin tener que preocuparse de los detalles de implementación de bajo nivel. Muchos SGBD también proporcionan un

entorno de cuarta generación consistente en un conjunto de herramientas que simplifican, en gran medida, el desarrollo de las aplicaciones que acceden a la base de datos. Gracias a estas herramientas, el programador puede ofrecer una mayor productividad en un tiempo menor.

- *Mejora en el mantenimiento gracias a la independencia de datos.* En los sistemas de ficheros, las descripciones de los datos se encuentran inmersas en los programas de aplicación que los manejan. Esto hace que los programas sean dependientes de los datos, de modo que un cambio en su estructura, o un cambio en el modo en que se almacena en disco, requiere cambios importantes en los programas cuyos datos se ven afectados. Sin embargo, los SGBD separan las descripciones de los datos de las aplicaciones. Esto es lo que se conoce como independencia de datos, gracias a la cual se simplifica el mantenimiento de las aplicaciones que acceden a la base de datos.

- *Aumento de la concurrencia.* En algunos sistemas de ficheros, si hay varios usuarios que pueden acceder simultáneamente a un mismo fichero, es posible que el acceso interfiera entre ellos de modo que se pierda información o, incluso, que se pierda la integridad. La mayoría de los SGBD gestionan el acceso concurrente a la base de datos y garantizan que no ocurran problemas de este tipo.

- *Mejora en los servicios de copias de seguridad y de recuperación ante fallos.* Muchos sistemas de ficheros dejan que sea el usuario quien proporcione las medidas necesarias para proteger los datos ante fallos en el sistema o en las aplicaciones. Los usuarios tienen que hacer copias de seguridad cada día, y si se produce algún fallo, utilizar estas copias para restaurarlos. En este caso, todo el trabajo realizado sobre los datos desde que se hizo la última copia de seguridad se pierde y se tiene que volver a realizar. Sin embargo, los SGBD actuales funcionan de modo que se minimiza la cantidad de trabajo perdido cuando se produce un fallo.

Inconvenientes de los sistemas de bases de datos

- *Complejidad.* Los SGBD son conjuntos de programas muy complejos con una gran funcionalidad. Es preciso comprender muy bien esta funcionalidad para poder sacar un buen partido de ellos.

- *Tamaño.* Los SGBD son programas complejos y muy extensos que requieren una gran cantidad de espacio en disco y de memoria para trabajar de forma eficiente.

- *Coste económico del SGBD.* El coste de un SGBD varía dependiendo del entorno y de la funcionalidad que ofrece. Por ejemplo, un SGBD para un ordenador personal puede costar 500 euros, mientras que un SGBD para un sistema multiusuario que dé servicio a cientos de usuarios puede costar entre 10.000 y 100.000 euros. Además, hay que pagar una cuota anual de mantenimiento que suele ser un porcentaje del precio del SGBD.

- *Coste del equipamiento adicional.* Tanto el SGBD, como la propia base de datos, pueden hacer que sea necesario adquirir más espacio de almacenamiento. Además, para alcanzar las prestaciones deseadas, es posible que sea necesario adquirir una máquina más grande o una máquina que se dedique solamente al SGBD. Todo esto hará que la implantación de un sistema de bases de datos sea más cara.

- *Coste de la conversión.* En algunas ocasiones, el coste del SGBD y el coste del equipo informático que sea necesario adquirir para su buen funcionamiento, es insignificante comparado al coste de convertir la aplicación actual en un sistema de bases de datos. Este coste incluye el coste de enseñar a la plantilla a utilizar estos sistemas y, probablemente, el coste del personal especializado para ayudar a realizar la conversión y poner en marcha el sistema. Este coste es una de las razones principales por las que algunas empresas y organizaciones se resisten a cambiar su sistema actual de ficheros por un sistema de bases de datos.

- *Prestaciones.* Un sistema de ficheros está escrito para una aplicación específica, por lo que sus prestaciones suelen ser muy buenas. Sin embargo, los SGBD están escritos para ser más generales y ser útiles en muchas aplicaciones, lo que puede hacer que algunas de ellas no sean tan rápidas como antes.

- *Vulnerable a los fallos.* El hecho de que todo esté centralizado en el SGBD hace que el sistema sea más vulnerable ante los fallos que puedan producirse.

1.6 Caso de estudio

En este apartado se describe una empresa inmobiliaria que está especializada en el alquiler de pisos y casas amuebladas.

Esta empresa se encarga de dar publicidad a los inmuebles que ofrece en alquiler, tanto en prensa local como nacional, entrevista a los posibles inquilinos, organiza las visitas a los inmuebles y negocia los contratos de alquiler. Una vez firmado el alquiler, la empresa asume la responsabilidad del inmueble, realizando inspecciones periódicas para comprobar su correcto mantenimiento.

A continuación se describen los datos que se manejan en las oficinas de la empresa para llevar a cabo el trabajo diario.

- *Oficinas.*

  La empresa tiene varias oficinas en todo el país. Cada oficina tiene un código de identificación que es único, tiene una dirección (calle, número y ciudad), un número de teléfono y un número de fax. Cada oficina tiene su propia plantilla.

- *Plantilla.*

  Cada oficina tiene un director que se encarga de supervisar todas sus gestiones. La empresa sigue muy de cerca el trabajo de los directores y tiene registrada la fecha en que cada director empezó en el cargo en su oficina. Cada director tiene un pago anual por gastos de vehículo y una bonificación mensual que depende de los contratos de alquiler que haya realizado su oficina.

  En cada oficina hay varios supervisores. Cada uno es responsable del trabajo diario de un grupo de entre cinco y diez empleados que realizan las gestiones de los alquileres. El trabajo administrativo de cada grupo lo lleva un administrativo.

  Cada miembro de la plantilla tiene un código único que lo identifica en la empresa. De cada uno de ellos se quiere conocer el nombre, la dirección, el número de teléfono, la fecha de nacimiento, el número del DNI, su puesto en la empresa, el salario anual y la fecha en que entró en la empresa. De los administrativos se desea conocer también la velocidad con que escriben a máquina (en pulsaciones por minuto).

Además, de cada empleado se debe guardar información sobre uno de sus parientes más próximos: nombre, relación con el empleado, dirección y número de teléfono.

- *Inmuebles para alquilar.*

  Cada oficina de la empresa tiene una serie de inmuebles para alquilar. Estos inmuebles se identifican por un código que es único dentro de la empresa. Los datos que se guardan de cada inmueble son los siguientes: dirección completa (calle, número y ciudad), tipo de inmueble, número de habitaciones y precio del alquiler en euros (este precio es mensual). El precio del alquiler se revisa de forma anual.

  Cada inmueble se asigna a un empleado que es el responsable de su gestión. Cada miembro de la plantilla puede tener asignados hasta veinte inmuebles para alquilar.

  Si un propietario elimina su oferta de alquiler de la empresa, su información se mantiene durante al menos tres años.

- *Propietarios.*

  Los propietarios de los inmuebles pueden ser particulares o empresas. A cada propietario se le asigna un código que es único en la empresa. De los particulares se guarda el nombre, la dirección y el número de teléfono. De las empresas se guarda el nombre comercial, tipo de empresa, la dirección, el número de teléfono y el nombre de la persona de contacto.

- *Inquilinos (clientes).*

  Cuando un cliente contacta con la empresa por primera vez, se toman sus datos: nombre, dirección, número de teléfono, tipo de inmueble que prefiere e importe máximo que está dispuesto a pagar al mes por el alquiler. Ya que es un posible inquilino, se le asigna un código que es único en toda la empresa. De la entrevista inicial que se realiza con cada cliente se guarda la fecha, el empleado que la realizó y unos comentarios generales sobre el posible inquilino.

- *Visitas a los inmuebles.*

  En la mayoría de los casos, los posibles inquilinos desean ver varios inmuebles antes de alquilar uno. De cada visita que se realiza se guarda la fecha y los comentarios realizados por el cliente respecto al inmueble.

- *Publicidad de los inmuebles.*

  Cuando algún inmueble es difícil de alquilar, la empresa lo anuncia en la prensa local y nacional. De cada anuncio se guarda la fecha de publicación y el coste económico del anuncio. De los periódicos se guarda el nombre, la dirección, el número de teléfono, el número de fax y el nombre de la persona de contacto.

- *Contratos de alquiler.*

  La empresa se encarga de redactar los términos de cada contrato de alquiler. Cada contrato tiene un número, un importe mensual, un método de pago, el importe del depósito, si se ha realizado el depósito, las fechas de inicio y finalización del contrato, la duración del contrato en meses y el miembro de la plantilla que lo formalizó. La duración mínima de un contrato es de tres meses y la duración máxima es de un año. Cada cliente puede tener alquilados uno o varios inmuebles al mismo tiempo.

- *Inspecciones.*

  Como parte del servicio que presta la empresa, ésta se encarga de realizar inspecciones periódicas a los inmuebles para asegurarse de que se mantienen en buen estado. Cada inmueble se inspecciona al menos una vez cada seis meses. Se inspeccionan tanto los inmuebles alquilados, como los que están disponibles para alquilar. De cada inspección se anota la fecha y los comentarios sobre su estado que quiera incluir el empleado que la ha llevado a cabo.

- *Actividades de cada oficina.*

  En cada oficina se llevan a cabo las siguientes actividades para garantizar que cada empleado tenga acceso a la información necesaria para desempeñar su tarea de modo efectivo y eficiente. Cada actividad está relacionada con una función específica de la empresa. Cada una de estas funciones corresponde a uno o varios puestos de los que ocupan los empleados, por lo que éstos se indican entre paréntesis.

  1. Crear y mantener las fichas con los datos de los empleados y su familiar más próximo (director).
  2. Realizar listados de los empleados de cada oficina (director).
  3. Realizar listados del grupo de empleados de un supervisor (director y supervisor).

4. Realizar listados de los supervisores de cada oficina (director y supervisor).
5. Crear y mantener las fichas con los datos de los inmuebles para alquilar (y de sus propietarios) de cada oficina (supervisor).
6. Realizar listados de los inmuebles para alquilar en cada oficina (toda la plantilla).
7. Realizar listados de los inmuebles para alquilar asignados a un determinado miembro de la plantilla (supervisor).
8. Crear y mantener las fichas con los datos de los posibles inquilinos de cada oficina (supervisor).
9. Realizar listados de los posibles inquilinos registrados en cada oficina (toda la plantilla).
10. Buscar inmuebles para alquilar que satisfacen las necesidades de un posible inquilino (toda la plantilla).
11. Crear y mantener las fichas de las visitas realizadas por los posibles inquilinos (toda la plantilla).
12. Realizar listados con los comentarios hechos por los posibles inquilinos respecto a un inmueble concreto (toda la plantilla).
13. Crear y mantener las fichas con los datos de los anuncios insertados en los periódicos (toda la plantilla).
14. Realizar listados de todos los anuncios que se han hecho sobre un determinado inmueble (supervisor).
15. Realizar listados de todos los anuncios realizados en un determinado periódico (supervisor).
16. Crear y mantener las fichas que contienen los datos sobre cada contrato de alquiler (director y supervisor).
17. Realizar listados de los contratos de alquiler de un determinado inmueble (director y supervisor).
18. Crear y mantener las fichas con los datos de cada inspección realizada a los inmuebles en alquiler (toda la plantilla).
19. Realizar listados de todas las inspecciones realizadas a un determinado inmueble (supervisor).

1.7 Resumen

Los predecesores de los sistemas de bases de datos son los sistemas de ficheros. Un sistema de ficheros está formado por un conjunto de programas que dan servicio a los usuarios finales. Cada programa define y gestiona sus propios datos. Aunque los sistemas de ficheros supusieron un gran avance sobre los sistemas manuales, tienen inconvenientes bastante importantes, como la redundancia de datos y la dependencia entre programas y datos.

Los sistemas de bases de datos surgieron con el objetivo de resolver los problemas que planteaban los sistemas de ficheros. Una base de datos es un conjunto de datos relacionados que recogen las necesidades de información de una empresa u organización. Estos datos se comparten por todos los usuarios. El SGBD es un conjunto de programas que permiten a los usuarios definir, crear y mantener la base de datos, además de proporcionar un acceso controlado a dicha base de datos.

La base de datos contiene tanto los datos como su definición. Todos los accesos a la base de datos se realizan a través del SGBD. El SGBD proporciona un lenguaje de definición de datos que permite a los usuarios definir la base de datos, y un lenguaje de manejo de datos que permite a los usuarios la inserción, actualización, eliminación y consulta de datos de la base de datos.

El SGBD proporciona un acceso controlado a la base de datos. Proporciona seguridad, integridad, concurrencia y controla la recuperación ante fallos. Además, proporciona un mecanismo de vistas que permite mostrar a los usuarios sólo aquellos datos que les interesan.

Las personas involucradas en el entorno de una base de datos son: el administrador de la base de datos, los diseñadores de la base de datos, los programadores de aplicaciones y los usuarios finales.

Las raíces de los SGBD se encuentran en los sistemas de ficheros. Los sistemas jerárquico y de red representan la primera generación de los SGBD, surgida hacia los años sesenta. El modelo relacional, propuesto por Codd en 1970, representa la segunda generación de los SGBD. Este modelo es el más extendido en la actualidad. La tercera generación de los SGBD se encuentra representada por el modelo relacional extendido y el modelo orientado a objetos.

Los sistemas de bases de datos presentan una serie de ventajas tanto por el hecho de compartir los datos, como por la existencia del SGBD. Algunas de estas ventajas son el control de la redundancia, la consistencia de datos, la mejora en los aspectos de seguridad y la integridad. Algunos de sus inconvenientes son su elevada complejidad, su coste y su vulnerabilidad ante fallos.

Bibliografía

En la mayoría de los libros sobre sistemas de bases de datos se presentan estos sistemas haciendo un repaso de sus predecesores, los sistemas de ficheros. Para la preparación de este capítulo se han utilizado los textos de Connolly, Begg y Strachan (1996), y Elmasri y Navathe (1997). El caso de estudio es el que presentan Connolly, Begg y Strachan (1996).

Capítulo 2

Organizaciones de ficheros y estructuras de acceso

En este capítulo se presentan los conceptos fundamentales sobre el almacenamiento físico de la base de datos en dispositivos de almacenamiento secundario, como los discos.

2.1 Introducción

Los discos son lentos pero también son maravillas de la tecnología: consiguen incorporar varios gigabytes de capacidad en un ordenador portátil. Hace algunos años, los discos con esa capacidad eran del tamaño de una pequeña lavadora. Pero comparados a otras partes del ordenador, los discos son lentos.

Acceder a memoria RAM cuesta alrededor de 120 nanosegundos, mientras que acceder al disco cuesta unos 30 milisegundos. Para darnos una idea de la diferencia de tiempos (30 mseg / 120 nseg = 250.000), vamos a hacer un símil: buscar algo en memoria RAM sería como buscar algo en el índice de un libro, para lo que empleamos unos 20 segundos. Buscar algo en disco sería como pedir a la biblioteca que localicen un libro y que el libro llegue después de unos 58 días.

Sin embargo, los discos proporcionan gran capacidad a un coste mucho menor que la

memoria RAM y además, mantienen la información aun estando apagados, son no volátiles.

Es fundamental y necesario un buen diseño de la estructura de los ficheros para que nuestras aplicaciones tengan acceso a toda la capacidad del disco sin que tengan que esperar mucho tiempo por los datos. El objetivo es minimizar el número de accesos a disco y maximizar la probabilidad de que la información que el usuario va a necesitar en breve ya esté en memoria RAM.

Los objetivos en el diseño de estructuras de ficheros son los siguientes:

- Obtener la información con un solo acceso a disco. Siguiendo con el símil, no tener que esperar varias veces los 58 días que cuesta obtener el libro. Esto sería el caso ideal.
- Si no se pueden obtener los datos en un solo acceso, hacen falta estructuras que permitan encontrar la información con el mínimo número de accesos posible (dos o tres como mucho). Los algoritmos rápidos de búsqueda no son suficientes: la búsqueda binaria encuentra un registro entre 5.000 con un máximo de 16 comparaciones; 16 accesos son demasiados cuando se accede a disco.
- Hace falta que la estructura del fichero permita agrupar la información de modo que se puedan obtener todos los datos que se necesitan de una sola vez.

Todo esto no es difícil si los ficheros nunca cambian, pero con ficheros que crecen y disminuyen en tamaño a medida que la información se añade o se borra, es mucho más complejo.

Las organizaciones de ficheros han ido evolucionando con el tiempo. En un principio, los ficheros estaban en cinta, con lo que el acceso era secuencial y por lo tanto el coste de acceso aumentaba conforme el fichero crecía en tamaño. Ya que los ficheros de gran tamaño no se podían manejar secuencialmente, la aparición de los discos hizo que se comenzaran a utilizar los *índices*.

Un índice es un fichero pequeño con una lista de claves y punteros donde la búsqueda es más rápida. Si se conoce el valor de la clave, el acceso es directo.

Cuando los índices son demasiado grandes y cambian con frecuencia, son difíciles de

gestionar, ya que son ficheros ordenados en los que la búsqueda se realiza de forma secuencial (búsqueda lineal). Es entonces cuando surge la idea de utilizar estructuras en forma de árbol como una posible solución. Esto ocurría a principios de los años sesenta.

Desafortunadamente, los árboles crecen de un modo muy desigual, con lo que en muchas ocasiones todavía hay que hacer muchos accesos a disco para obtener la información. En 1963 se crean los *árboles AVL* (árbol binario autoajustable), que están equilibrados en altura: entre dos subárboles de la misma raíz sólo puede haber una diferencia en altura de un nivel. Estos árboles son muy buenos como estructuras de datos en memoria RAM, pero muy malos para ficheros en disco, ya que requieren muchos accesos: al ser binarios son demasiado profundos. Incluso los árboles binarios equilibrados requieren demasiados accesos. Hacía falta un modo de tener un árbol equilibrado cuando cada nodo del árbol no era un solo registro (como en un árbol binario), sino un bloque conteniendo decenas o incluso centenas de registros.

En 1972 (casi diez años después), surgen los *árboles B* (*balanced*: equilibrado), en los que todas las hojas están al mismo nivel. Cuesta tanto tiempo encontrarlos porque su filosofía es completamente diferente de la filosofía de los árboles AVL: crecen de abajo hacia arriba conforme se añaden registros. Los árboles B ofrecen muy buenas prestaciones pero presentan un problema: no es posible acceder secuencialmente a los registros del fichero de modo eficiente. Casi de inmediato se crea el *árbol B+*, añadiendo una lista enlazada en el nivel más bajo del árbol B, con lo que también se permite el acceso secuencial. Con los árboles B+ se obtiene la información en tres o cuatro accesos entre millones de registros, y las prestaciones se mantienen aunque se añadan o eliminen registros.

Más tarde, a finales de los años setenta, se empieza a utilizar la *dispersión (hashing)*. Se había utilizado anteriormente para índices, pero no para ficheros dinámicos que cambian con frecuencia. Después de los árboles B, los investigadores se pusieron a trabajar con la dispersión dinámica extensible, que permite acceder a la información en uno o dos accesos a disco, independientemente del tamaño del fichero.

2.2 Conceptos fundamentales de organizaciones de ficheros

En los dispositivos de almacenamiento secundario, la base de datos se organiza en uno o varios *ficheros*. Cada fichero está formado por una serie de *registros*, y cada registro se divide en varios *campos*. Normalmente, los registros corresponden a *entidades*: personas, objetos, eventos, etc., siendo los campos aquellos *atributos* o *propiedades* que se desea conocer sobre las entidades: el nombre de cada persona, el color del objeto, la fecha del evento, etc.

Por ejemplo, el fichero con información sobre la plantilla de la empresa inmobiliaria estará formado por registros de empleados con los siguientes campos: número del empleado, apellido, puesto, DNI y número de la oficina a la que pertenece:

Enum	Apellido	Puesto	DNI	Onum
EL21	Pastor	Director	39432212E	05
EG37	Cubedo	Supervisor	38766623X	03
EG14	Collado	Administrativo	24391223L	03
EA9	Renau	Supervisor	39233190F	07
EG5	Prats	Director	25644309X	03
EL41	Baeza	Supervisor	39552133T	05

Cuando un usuario pide los datos del empleado *EG37*, el SGBD identifica el bloque de disco que contiene dichos datos y lo pide al sistema operativo. Este utilizará sus rutinas de acceso a ficheros para obtener el bloque y lo almacenará en los *buffers* del SGBD en memoria principal. Un *bloque* o *página* es la unidad mínima de transferencia entre disco y memoria principal. Normalmente, caben varios registros de datos en un mismo bloque. Si un registro no cabe en un solo bloque, se repartirá entre varios. Siguiendo con el ejemplo anterior, si caben tres registros de plantilla por bloque, el fichero ocupará dos bloques:

Enum	Apellido	Puesto	DNI	Onum	**bloque**
EL21	Pastor	Director	39432212E	05	
EG37	Cubedo	Supervisor	38766623X	03	**1**
EG14	Collado	Administrativo	24391223L	03	
EA9	Renau	Supervisor	39233190F	07	
EG5	Prats	Director	25644309X	03	**2**
EL41	Baeza	Supervisor	39552133T	05	

El orden en que se colocan los registros en un fichero depende de su estructura. Los principales tipos de estructuras de ficheros son los siguientes:

- *Ficheros desordenados.* En estos ficheros los registros no siguen un orden específico.
- *Ficheros ordenados.* En estos ficheros los registros están ordenados por el valor de un determinado campo.
- *Ficheros dispersos.* En estos ficheros los registros se almacenan en la posición del fichero que indica una función matemática al ser aplicada sobre un determinado campo.
- *Agrupamiento.* Varios ficheros intercalan sus registros de modo que éstos queden agrupados por el valor de algún campo que tienen en común.

Los pasos que se deben llevar a cabo para almacenar y acceder a un registro de un fichero es lo que se denomina un *método de acceso.* Hay distintos métodos de acceso, algunos de los cuales sólo se pueden utilizar con determinadas estructuras de ficheros.

2.3 Dispositivos de almacenamiento secundario

El almacenamiento secundario se realiza generalmente en disco y en cinta. Normalmente los ficheros están almacenados en disco y los datos se transfieren desde el disco a la memoria principal a medida que se necesitan. El almacenamiento en disco es la forma principal de almacenamiento con acceso directo. El almacenamiento en cinta es más lento y el acceso es sólo secuencial, por lo que la función de las cintas está, básicamente, limitada a archivar datos.

2.3.1 Discos

La unidad física en que está contenido el medio de grabación del disco se llama *controlador de disco.* El controlador contiene un paquete de discos, también denominado *volumen.* El paquete de discos está formado por un conjunto de superficies grabables (discos) montados sobre un eje.

En operación, el eje y los discos rotan a una alta velocidad. Los datos se graban sobre las *pistas*, que son coronas circulares que se encuentran sobre cada superficie. Las pistas que se encuentran unas sobre otras forman *cilindros*.

Hay un conjunto de cabezas de lectura/escritura ubicadas al final de un brazo que se mueven como un grupo, de tal forma que éstas pueden ser posicionadas sobre todas las pistas de un mismo cilindro. De este modo, toda la información de un cilindro puede ser accedida sin tener que mover el brazo, que es la operación más lenta.

Las pistas se dividen en porciones, que son las unidades más pequeñas de espacio que se pueden direccionar, y pueden ser *sectores* o *bloques*.

- *Sectores*. Son divisiones físicas y, por tanto, de tamaño fijo. Los sectores van numerados, de modo que sectores consecutivos tienen números consecutivos.

  Un *cluster* es una cantidad fija de sectores contiguos, y constituye la unidad de espacio más pequeña que se puede asignar a un fichero. El gestor de ficheros ve un fichero como un conjunto de *clusters*, información que se encuentra almacenada en la FAT (*File Allocation Table*). Una vez se localiza un *cluster*, no hay que mover las cabezas para leer todos sus sectores. El administrador del sistema es quien define cuántos sectores habrá en un *cluster*; cuantos más haya, menos movimientos de las cabezas hay que realizar, con lo que se mejoran las prestaciones en el acceso secuencial.

  Un *extent* es un trozo de fichero formado por varios *clusters* contiguos. Mediante ellos se pretende enfatizar más la contigüidad física de los sectores. Lo ideal (en acceso secuencial) es tener un solo *extent*. Cuantos más haya, más desperdigado estará el fichero y, por lo tanto, más tiempo se dedicará a mover las cabezas de lectura/escritura.

  Cuando las pistas se encuentran divididas en sectores hay problemas de fragmentación interna debido a que todos los sectores son del mismo tamaño: si en un sector no cabe un múltiplo del número de registros, hay huecos. Cabe la posibilidad de partir los registros entre sectores, pero esto hará que para acceder a ciertos registros, haya que traer dos sectores en lugar de uno. Otro origen del problema de la fragmentación interna viene por el hecho de la asignación de *clusters* enteros a ficheros: puede que sobre espacio en un *cluster*, aunque no es tan malo, ya que se puede necesitar más tarde.

Al principio de cada sector hay información invisible al programador en donde figura la dirección del sector, la dirección de la pista en la que se encuentra y su condición (en buen estado o defectuoso).

- *Bloques.* Son divisiones lógicas y, por lo tanto, su tamaño es variable. Este tamaño puede ser distinto para cada aplicación, y se puede fijar como un múltiplo del tamaño de registro, por lo que no hay problemas de fragmentación interna.

  Los bloques contienen datos e información sobre los datos: tamaño en bytes y, en ocasiones, una clave indicando, por ejemplo, el valor de algún campo clave del último registro del bloque. Además, hay información invisible al programador al principio de cada bloque, más cantidad que con los sectores.

Comparando sectores y bloques, estos últimos ofrecen mayor flexibilidad: se ahorra tiempo y se consigue mayor eficiencia, ya que el programador determina cómo se van a organizar los datos en disco. Los bloques son superiores a los sectores cuando se desea que la localización física de los ficheros corresponda con su organización lógica. Sin embargo, el programador y/o el sistema operativo deben hacer trabajo extra para determinar la organización de los datos.

La dirección de un registro en disco, normalmente necesita información sobre el número de cilindro, la superficie y el sector o bloque.

Coste del acceso a disco

En general hay tres factores que afectan directamente a la velocidad con que los datos se transfieren entre disco y memoria principal. Cada uno de estos factores consume un tiempo y conlleva una operación física.

- *Tiempo de búsqueda (seek):* es el tiempo necesario para mover el brazo con las cabezas de lectura/escritura desde su posición actual hasta el cilindro direccionado. El brazo puede estar sobre el cilindro deseado, tener que ir al siguiente cilindro o al otro extremo del disco; por lo tanto, dependiendo de la distancia a recorrer, será más o menos caro. En un sistema multiusuario este tiempo es más notable ya que cada usuario estará

accediendo a un fichero distinto y lo más probable es que en cada acceso de cada usuario haya que emplear un tiempo de búsqueda que será, por término medio, el tiempo de recorrer un tercio del número de cilindros.

- *Tiempo de rotación (latencia):* el disco debe girar hasta que la cabeza esté situada sobre el sector a leer o escribir. El tiempo medio es el tiempo requerido para dar media vuelta.
- *Tiempo de transferencia:* es el tiempo empleado en leer o escribir los datos. Este tiempo es función del número de bytes transferidos (número de bytes transferidos / número de bytes por pista $*$ tiempo de dar una vuelta).

El disco: un cuello de botella

Las prestaciones de los discos crecen constantemente, pero todavía están muy lejos de las velocidades de las redes. Por ejemplo, un disco de 50 Kbytes por pista puede proporcionar datos a una velocidad pico de 5 Mbytes por segundo, mientras que una red local puede tener una velocidad de 100 Mbytes por segundo. Esto hace que la CPU tenga que esperar a los datos debido a la lentitud de los discos, ya que la red es lo suficientemente rápida. Hay una serie de técnicas que se utilizan para resolver este problema:

- *Multiprogramación:* la CPU hace otras tareas mientras espera.
- *Stripping:* el fichero se divide en trozos y se reparte entre varios discos, de modo que todos los discos puedan estar mandando datos a la vez. Esta técnica necesita que la gestión de los accesos a disco sea más sofisticada.
- *Discos RAM:* se tiene memoria RAM configurada para actuar como un disco. El acceso es rápido, lo caro es la memoria RAM. Tiene el inconveniente de que la memoria RAM es volátil.
- *Caché de disco:* se tiene memoria RAM configurada para tener páginas de datos (sectores o bloques) del disco. El gestor de ficheros mira primero en la caché a ver si está el sector que se desea leer o escribir, y sólo si no lo encuentra es cuando va a disco. Por el principio de localidad de las referencias, hay una gran probabilidad de que lo que se necesita esté ya en la caché al ir a buscarlo.

2.3.2 Acceso a los datos

En este apartado se muestra, mediante un ejemplo, qué partes del software y del hardware están involucradas en una operación de acceso a disco. Supongamos que un programa de usuario ha solicitado escribir en el fichero `texto` el contenido de la variable `c`, cuyo tamaño es de 1 byte.

```
write("texto",c,1)                    c: 'P'
```

1. El programa pide al sistema operativo que escriba el contenido de `c` en el fichero `texto`.
2. El sistema operativo pasa el trabajo al gestor de ficheros.
3. El gestor de ficheros comprueba que las características lógicas del fichero son coherentes con lo que se quiere hacer: el fichero está abierto para escritura, el usuario posee los permisos oportunos, etc.
4. El gestor de ficheros busca en la FAT la posición física del sector/bloque que recibirá el byte.
5. El gestor de ficheros se asegura de que el sector/bloque se encuentra en un *buffer* de entrada/salida en memoria RAM, y escribe el byte `'P'` en la posición correspondiente en el *buffer*.
6. El gestor de ficheros dice al procesador de entrada/salida dónde se encuentra el byte en memoria RAM y cuál es su destino en el disco.
7. El procesador de entrada/salida espera a que el controlador esté disponible y pone el dato en el formato del disco.
8. El procesador de entrada/salida envía el dato al controlador del disco.
9. El controlador del disco indica al dispositivo que mueva la cabeza a la pista, espera hasta que el sector/bloque esté bajo ella y le manda el byte, que es escrito bit a bit.

2.4 Ficheros desordenados

En un fichero desordenado, también denominado fichero *heap*, los registros se colocan en el fichero en el orden en que se van insertando. Cuando llega un nuevo registro, se inserta en el último bloque del fichero; si el fichero está lleno, se le añade más espacio. Esta organización hace que la inserción sea muy eficiente. Sin embargo, cuando se trata de acceder a un registro del fichero, es necesario realizar una búsqueda lineal recorriendo todos los bloques del fichero uno tras otro, hasta encontrar el registro deseado. Esto hace que las búsquedas de registros en los ficheros desordenados sean muy lentas.

Si lo que se desea es obtener todos los registros del fichero presentándolos en un determinado orden (según el valor de alguno de sus campos), es necesario realizar una ordenación externa si el fichero completo no cabe en memoria principal: se van leyendo los registros, se escriben de forma ordenada en un fichero temporal en disco y, finalmente, se accede a este último fichero para leer los registros una vez ordenados.

Para eliminar un registro es necesario traer a memoria principal el bloque en el que se encuentra, después se marca el registro como borrado y, por último, se vuelve a escribir el bloque en el mismo lugar en que se encontraba. El espacio que ocupaba el registro borrado no se reutiliza, lo que hace que empeoren las prestaciones, por lo que, cada cierto tiempo, habrá que realizar una reorganización del fichero para recuperar este espacio.

Para modificar un registro también es necesario traerlo a memoria principal y actualizarlo. Si el registro modificado cabe en el bloque en el que se encontraba, se reescribe el bloque en su sitio. Si el registro modificado ha aumentado su tamaño de modo que no cabe en el bloque, hay que borrar el registro original y añadir el registro modificado al final del fichero.

Los ficheros desordenados son los más adecuados cuando se trata de cargar grandes cantidades de datos, ya que la inserción es muy eficiente al no tener que realizarse ningún cálculo para determinar la posición que debe ocupar el registro en el fichero.

2.5 Ficheros ordenados

En los ficheros ordenados, los registros se encuentran ordenados físicamente según el valor de uno o varios campos. A este campo o campos se les denomina *campos de ordenación*.

Cuando se trata de buscar un registro en un fichero ordenado, se puede utilizar una búsqueda binaria sólo cuando se busca por el campo de ordenación. Si la búsqueda se realiza a través de cualquier otro campo, se debe hacer una búsqueda lineal. Si se quiere leer el fichero ordenadamente, la operación será muy eficiente si el orden escogido es el del campo de ordenación. Si el orden de lectura es sobre cualquier otro campo, es preciso realizar una ordenación externa.

Para insertar un registro hay que encontrar su posición en el fichero según el orden establecido, hacer hueco y escribir. Si hay espacio suficiente en el bloque correspondiente, se reordena el bloque y se escribe en el fichero. Si no hay espacio en el bloque, habrá que mover algún registro al siguiente bloque. Si este último bloque no tiene espacio suficiente para estos registros, habrá que mover algunos registros al bloque siguiente, y así sucesivamente. Esto hace que la inserción de un registro pueda ser muy costosa. Una solución consiste en tener un fichero desordenado de desborde, en donde se van realizando todas las inserciones. Cada cierto tiempo, el fichero de desborde se mezcla con el fichero ordenado, de modo que así las inserciones son muy eficientes. Esto perjudica a las búsquedas, ya que si el registro no se encuentra mediante búsqueda binaria en el fichero ordenado, hay que realizar una búsqueda lineal en el fichero de desborde.

Cuando se trata de eliminar un registro, hay que encontrarlo y marcarlo como borrado. Como en los ficheros desordenados, cada cierto tiempo se debe realizar una reorganización del fichero. Para modificar un registro hay que encontrarlo y actualizarlo en caso de que quepa en el bloque en el que se encontraba. Si el registro actualizado ya no cabe en su ubicación, hay que borrarlo, hacer hueco e insertarlo. Si en la actualización se modifica el campo de ordenación, entonces es muy probable que haya que cambiar el registro de lugar para mantener el orden: habrá que borrarlo, hacer hueco e insertarlo de nuevo.

Los ficheros ordenados no se suelen utilizar como ficheros de datos en los SGBD.

2.6 Ficheros dispersos

En los ficheros dispersos, la dirección de cada registro se calcula aplicando cierta función sobre uno o varios de sus campos, denominados *campos de dispersión*. A la función se le denomina *función de dispersión*. Los registros de este tipo de ficheros parece que han sido distribuidos de forma aleatoria por el mismo, por lo que a estos ficheros también se les denomina ficheros aleatorios o ficheros directos. El acceso a los datos es muy rápido sólo si se busca con la condición de igualdad sobre el campo de dispersión.

La función de dispersión se debe escoger de modo que los registros queden distribuidos uniformemente en todo el fichero. La técnica más popular consiste en utilizar el resto de la *división entera*: se toma el valor de un campo, se divide entre un valor determinado y se utiliza el resto de la división entera para obtener la dirección en disco. Otra técnica es el *plegado*. Consiste en tomar distintas partes del campo de dispersión y aplicarles alguna función aritmética. Por ejemplo, si el campo de dispersión es el DNI, se pueden tomar sus dígitos por parejas y sumarlos. Si el campo tiene caracteres, se puede tomar su valor ASCII para aplicar la operación aritmética. En realidad, la función de dispersión produce un número de bloque relativo. En una tabla que se encuentra en la cabecera del fichero se convierte este número en la dirección del bloque en el disco.

El problema que presentan la mayoría de las funciones de dispersión es que no garantizan direcciones únicas, de modo que varios registros se asocian a una misma dirección de bloque. Si a un bloque se asocian más registros de los que realmente caben, se producen *colisiones* que hay que resolver. Los registros que se destinan a un mismo bloque se denominan *sinónimos*.

Hay varias técnicas para gestionar las colisiones:

- *Direccionamiento abierto.* Cuando se produce una colisión, el sistema hace una búsqueda lineal a partir del bloque al que iba destinado el registro para encontrar un hueco donde insertarlo. Si se llega al final del fichero sin encontrar hueco, se continúa la búsqueda desde el principio.
- *Encadenamiento.* En lugar de buscar un hueco libre, lo que se hace es disponer de una serie de bloques como área de desborde. Cuando se produce una colisión, el registro se sitúa en el área de desborde y mediante un puntero en el bloque colisionado, se

apunta a la dirección del bloque de desborde y la posición relativa del registro dentro del bloque. Además, todos los registros que han colisionado en un mismo bloque se van encadenando mediante punteros.

- *Dispersión múltiple.* Esta técnica de resolución de colisiones consiste en utilizar una segunda función de dispersión cuando la primera ha producido una colisión. El objetivo es producir una nueva dirección que no provoque colisión. Normalmente, la segunda función da una dirección de bloque situada en un área de desborde.

Aunque la dispersión es el método de acceso directo más rápido a través del campo de dispersión, no es muy útil cuando también se quiere acceder al fichero a través de otro campo. Ya que la mayoría de las funciones de dispersión que se utilizan no mantienen el orden entre los registros, tampoco es útil cuando se quiere leer los registros ordenadamente.

Buscar un registro a través de un campo que no es el de dispersión es tan caro como buscar un registro en un fichero desordenado: es preciso realizar una búsqueda lineal. Para borrar un registro hay que eliminarlo del bloque en el que se encuentra. Si el bloque tiene una lista de desborde, se puede mover un registro de la lista al bloque. Si el registro a borrar está en la lista de desborde, sólo hay que eliminarlo de ella. En este caso, será necesario mantener una lista enlazada de posiciones de desborde no utilizadas. Si al actualizar un registro se modifica el campo de dispersión, es muy probable que el registro tenga que cambiar de posición, por lo que habrá que borrarlo e insertarlo de nuevo.

Otra desventaja de la dispersión es que el espacio de almacenamiento es fijo, por lo que es difícil que el fichero se expanda o se comprima dinámicamente. Si el fichero tiene m bloques y en cada uno caben n registros, como máximo se podrán almacenar $m * n$ registros. Si hay menos de $m * n$ registros, quedará espacio sin utilizar. Si hay más de $m * n$ registros, habrá muchas colisiones y esto hará que el acceso sea más lento, ya que habrá largas listas de desborde. En este caso, puede ser preferible ampliar el espacio de almacenamiento y cambiar a otra función de dispersión, lo que implica que habrá que redistribuir todos los registros (reorganizar el fichero). Ya que el espacio de almacenamiento es fijo, a este tipo de dispersión se la denomina *dispersión estática*.

Hay varios esquemas que tratan de remediar esta situación: la *dispersión dinámica* y la *dispersión extensible*, que almacenan una estructura de acceso además del fichero, y la

dispersión lineal, que no necesita dicha estructura.

Estas técnicas aprovechan el que las funciones de dispersión dan como resultado un valor entero no negativo que, por tanto, se puede representar como un número binario. Los registros se distribuyen entre los bloques teniendo en cuenta los primeros bits de este número, que se denomina *valor de dispersión*.

2.6.1 Dispersión dinámica

Cuando se utiliza esta técnica, el número de bloques del fichero no es fijo, sino que crece y disminuye conforme es necesario. El fichero puede empezar con un solo bloque y cuando éste se llena, se toma un nuevo bloque. Los registros se distribuyen entre los dos bloques teniendo en cuenta el primer bit de sus valores de dispersión. Los que tienen un 0 van a un bloque y los que tienen un 1 van al otro. Además, se va formando un directorio con estructura de árbol binario que tiene dos tipos de nodos:

- Nodos internos, que guían la búsqueda. Tienen un puntero izquierdo correspondiente al bit 0 y un puntero derecho correspondiente al bit 1.
- Nodos hoja, conteniendo el puntero al bloque de datos.

Para hacer una búsqueda se puede almacenar el directorio en memoria, a menos que sea muy grande. Si no cabe en un bloque, habrá que distribuirlo en dos o más niveles. Cada nodo interno puede tener también un puntero al padre. Se pueden utilizar representaciones especiales de árboles binarios para reducir el espacio necesario para los tres punteros (izquierdo, derecho y padre). En general, un directorio de x niveles puede necesitar hasta $x+1$ accesos a bloque para alcanzar un registro.

Cuando un bloque se llena, se "parte en dos"y los registros se redistribuyen según el siguiente bit de su valor de dispersión. En este caso, el directorio se expande con un nuevo nodo interno. Los niveles del árbol binario se expanden dinámicamente (nunca habrá más niveles que número de bits hay en el valor de dispersión).

Si la función de dispersión distribuye los registros uniformemente, el árbol estará equilibrado. Cuando un bloque se vacía, o si el número de registros en dos bloques vecinos cabe

en un solo bloque, el directorio pierde un nodo interno y los dos nodos hoja se combinan formando uno solo. De este modo los niveles del árbol pueden disminuir dinámicamente.

2.6.2 Dispersión extensible

En la dispersión extensible el directorio es un vector de 2^d direcciones de bloque, donde d es la profundidad global del directorio. El entero formado por los d primeros bits del valor de dispersión es el índice que indica en qué entrada del vector se encuentra la dirección del bloque que contiene al registro. Puede haber varias entradas en el directorio con los primeros d' bits en común ($d' < d$) apuntando al mismo bloque cuando los registros que van a parar a ellas caben en dicho bloque. Cada bloque tiene una profundidad local d' (que se almacena con él), que especifica el número de bits que tienen en común los valores de dispersión de los registros que en él se encuentran.

El valor de d puede incrementarse o disminuirse en una unidad cada vez que se necesita doblar o disminuir a la mitad el número de entradas en el vector directorio. Es necesario doblar el directorio si un bloque con profundidad local d', igual a la profundidad global d, se llena. El directorio se disminuye a la mitad si $d > d'$ para todos los bloques después de borrar algún registro. Como mucho, son necesarios dos accesos a bloque para llegar a cualquier registro (acceso al directorio y acceso al bloque que contiene el registro).

Comparando entre la dispersión dinámica y la extensible, en la dispersión dinámica el directorio es una estructura enlazada, mientras que en la extensible es un árbol perfecto, por lo que se puede expresar como un vector. En la dispersión dinámica el directorio crece más lentamente, mientras que en la extensible va doblando su tamaño. Sin embargo, ya que en la dispersión dinámica los nodos deben almacenar punteros a los hijos, su tamaño debe ser más grande que en la extensible (al menos el doble de grande). Por lo tanto, el directorio de la dinámica ocupará más espacio en memoria. Además, si el directorio se hace lo suficientemente grande como para necesitar memoria virtual, la dispersión extensible ofrece la ventaja de que se puede acceder al directorio con no más de un fallo de página. Ya que la dispersión dinámica utiliza una estructura enlazada para el directorio, puede que se produzca más de un fallo de página al moverse por el mismo.

2.6.3 Dispersión lineal

El objetivo de la dispersión lineal es que el fichero pueda crecer y disminuir sin tener que utilizar un directorio. Así se ahorra el tiempo extra que los anteriores tipos de dispersión necesitan para acceder al directorio.

Supongamos que un fichero empieza con M bloques, numerados $0, 1, 2, \ldots, M-1$, y utiliza la función de dispersión $h(K) = K \bmod M$, a la que se denomina función inicial h_0. Cuando hay colisiones se utilizan listas de desborde, una lista para cada bloque. Además, después de una colisión, uno de los bloques del fichero se parte en dos y no es necesariamente el bloque en el que ha ocurrido la colisión. Por ejemplo, se produce la primera colisión. Se inicia una lista de desborde asociada al bloque colisionado y, además, el primer bloque del fichero, el bloque 0, se parte en dos bloques: el original (bloque 0) y uno nuevo al final del fichero (bloque M). Los registros del bloque 0 original se distribuyen entre los dos bloques utilizando una función de dispersión diferente $h_1(K) = K \bmod 2M$. Una propiedad clave de ambas funciones h_0 y h_1 es que los registros que iban a parar al bloque 0 con la función h_0, irán a parar al bloque 0 o al nuevo bloque M con la función h_1; esto es necesario para que la dispersión lineal funcione.

Conforme se van produciendo más colisiones, se van partiendo bloques en orden lineal $1, 2, 3, \ldots$ Cuando un bloque se parte, los registros de su lista de desborde (si la tiene) se redistribuyen en bloques del fichero utilizando la función h_1. No se utiliza directorio, sólo es necesario un valor n para determinar qué bloques se han partido. Para acceder a un registro con valor de dispersión K, primero se le aplica la función h_0 a K; si $h_0(K) < n$, entonces hay que aplicar la función h_1 a K porque el bloque ha sido partido. Al principio n vale cero, indicando que la función h_0 se aplica a todos los bloques; n crece linealmente conforme se van partiendo los bloques. Cuando $n = M$, todos los bloques originales se han partido y, entonces, la función que se aplica a todos los bloques es h_1. En este momento, n se pone otra vez a cero y si se producen nuevas colisiones, se utilizará una nueva función $h_2(K) = K \bmod 4M$. En general, se utiliza una secuencia de funciones $h_j(K) = K \bmod (2^j M)$, donde $j = 0, 1, 2, \ldots$ Hace falta utilizar una nueva función $h_{j+1}(K)$ cuando todos los bloques $0, 1, \ldots, (2^j M) - 1$, se han partido en dos (entonces n se pone a cero).

Los bloques que se han partido en dos se pueden recombinar si la carga del fichero

disminuye por debajo de un cierto umbral. En general, el factor de carga del fichero se puede definir como $l = r/(b * N)$, donde r es el número de registros en el fichero, b es el número de registros que caben en un bloque y N es el número de bloques que está ocupando el fichero. Los bloques se combinan de forma lineal y N se decrementa de modo apropiado.

2.7 Agrupamiento

Hasta ahora se ha supuesto que todos los registros de un fichero son del mismo tipo. En muchas aplicaciones existen relaciones entre los registros de varios ficheros distintos. Estas relaciones se pueden representar mediante *campos conectores*. Por ejemplo, en el registro de un empleado aparece un campo conector cuyo valor indica el código de la oficina a la que pertenece. En el fichero de las oficinas, los registros tendrán un campo que será el código de oficina. Cuando se quiere acceder a los campos de dos registros relacionados, por ejemplo, a los datos de un empleado y los datos de su oficina, se debe acceder primero al registro del empleado y utilizar el campo conector para obtener el registro con el que se relaciona en el fichero de las oficinas. De este modo, las relaciones se están representando mediante *referencias lógicas* entre registros de distintos ficheros.

Otro modo de representar las relaciones es mediante *referencias físicas*: los registros relacionados se almacenan juntos en el mismo fichero. Esta organización es apropiada cuando se espera utilizar muy a menudo una determinada relación en el acceso a los datos, ya que implementarla físicamente puede aumentar la eficiencia del sistema al acceder a los registros relacionados. Por ejemplo, si se quisiera acceder con frecuencia a una oficina junto con los empleados que trabajan en la misma, sería conveniente poner el registro de cada oficina junto con los registros de todos sus empleados contiguos en el disco, en un mismo fichero. Ya que los registros de todos los empleados de una misma oficina se encuentran físicamente juntos, y a continuación del registro con los datos de su oficina, el campo que hace referencia a la misma en los registros de los empleados no es necesario.

Para poder distinguir los registros de un fichero en el que se ha utilizado el agrupamiento, hay que añadir un campo a cada registro en donde se indique su tipo (suele ser el primer campo).

También es posible utilizar el agrupamiento en ficheros cuyos registros son todos del mismo tipo. Por ejemplo, en un fichero de empleados se puede tener los registros de los empleados agrupados por puesto de trabajo: todos los empleados con un mismo puesto están contiguos físicamente. Del mismo modo que antes, el campo que hace referencia al puesto se puede eliminar, pero hay que incluir un registro con el nombre del puesto delante del grupo de empleados.

Cuando se utiliza el agrupamiento, se favorecen los accesos a través de la relación en la que se basa dicho agrupamiento, pero se penalizan los accesos a través de cualquier otro campo. Ya que éste es un fichero ordenado (aunque de un modo especial), tiene las mismas ventajas e inconvenientes que los ficheros ordenados.

2.8 Índices

Los índices son estructuras de acceso que se utilizan para acelerar el acceso a los registros en respuesta a ciertas condiciones de búsqueda. Algunos tipos de índices, los denominados caminos de acceso secundario, no afectan al emplazamiento físico de los registros en el disco y lo que hacen es proporcionar caminos de acceso alternativos para encontrar los registros de modo eficiente basándose en los campos de indexación. Hay otros tipos de índices que sólo se pueden construir sobre ficheros que tienen una determinada organización.

En general, todas las organizaciones de ficheros descritas en los apartados anteriores se pueden utilizar como caminos de acceso secundarios. Sin embargo, los tipos de índices que más se utilizan son los que se basan en ficheros ordenados (índices de un solo nivel) y las estructuras en forma de árbol (índices multinivel, árboles B y árboles B+). Además, los índices se pueden construir mediante dispersión u otras estructuras de datos.

2.8.1 Índices de un solo nivel

Los índices de un solo nivel están ordenados y son algo similar al índice de palabras de un libro en donde cada palabra viene acompañada de una lista de números de página en donde aparece la palabra. Para hacer una búsqueda se puede recurrir al índice o bien hacer

una búsqueda lineal a lo largo de todo el libro.

Un índice de un solo nivel es un fichero ordenado cuyos registros tienen dos campos: el campo sobre el que se define el índice denominado *campo de indexación* (un campo de los registros del fichero de datos) y una dirección de bloque en disco. Los valores del índice están ordenados, de modo que se puede utilizar la búsqueda binaria. Hay varios tipos de índices de un solo nivel: índices primarios, índices de agrupamiento e índices secundarios.

Índices primarios

Si un fichero está ordenado por un campo clave, el índice que se define sobre este campo es un índice primario.

Un índice primario es un fichero ordenado cuyos registros son de longitud fija y tienen dos campos: el campo de ordenación del fichero de datos (hay un valor único de este campo para cada registro) y un puntero a un bloque del disco (una dirección de bloque). El campo de ordenación se suele denominar *clave*. Hay una *entrada* (registro) en el índice por cada bloque del fichero de datos. Cada entrada tiene el valor de la clave del último registro del bloque y un puntero a dicho bloque.

Un índice primario es un ejemplo de índice no denso: hay una entrada por cada bloque de disco del fichero de datos, y no una entrada por cada registro. Un índice denso contiene una entrada por cada registro del fichero de datos. El fichero en donde se encuentra el índice necesita muchos menos bloques que el fichero de datos porque hay menos entradas que registros en el fichero de datos, y porque sus registros son más pequeños que los del fichero de datos. Por lo tanto, una búsqueda binaria sobre el índice visita menos bloques del disco que una búsqueda binaria sobre el fichero de datos.

Un problema importante con los índices primarios (como con cualquier fichero ordenado) es la inserción y el borrado de registros. De hecho, es incluso más grave, ya que hacer una inserción en el fichero de datos (que es un fichero ordenado) requiere el movimiento de registros en el fichero de datos y, además, hay que tocar el índice, ya que, seguramente, el movimiento de registros en el fichero de datos cambia el último registro de algunos bloques y, por tanto, hay que actualizar el índice. Se puede utilizar un fichero de desborde desordenado para reducir este problema. Otra posibilidad es utilizar una lista enlazada de registros

de desborde, como en la dispersión. Los registros de cada bloque y su lista enlazada de desborde se pueden ordenar para mejorar el tiempo de acceso. El borrado de registros se puede gestionar mediante el uso de marcas de borrado.

Sobre un fichero ordenado por clave sólo puede definirse un índice primario.

Índices de agrupamiento

Si un fichero está ordenado por un campo no clave, un índice que se define sobre este campo es un índice de agrupamiento.

Si los registros de un fichero están ordenados según un campo no clave (no hay un valor único para cada registro), a ese campo se le llama *campo de agrupamiento*. Se puede crear un índice distinto, un índice de agrupamiento, para acceder más rápido al fichero a través de dicho campo. Un índice de agrupamiento es, también, un fichero ordenado con dos campos: el campo de agrupamiento por el que está ordenado el fichero y un puntero a un bloque de disco, por lo que también es un índice no denso. Hay una entrada en el índice por cada valor distinto del campo de agrupamiento, y el puntero señala al primer bloque que contiene un registro con dicho valor.

Nótese que la inserción y el borrado de registros siguen dando problemas, ya que es un fichero ordenado físicamente. Esto se puede subsanar si se reserva un bloque entero para cada valor distinto. Si hacen falta más bloques para un valor, se añaden y se van enlazando. Esto hace que la inserción y el borrado sean más simples.

Sobre un fichero ordenado por un campo no clave (campo de agrupamiento) sólo se puede definir un índice de agrupamiento.

Índices secundarios

Si un fichero está desordenado, cualquier índice de un solo nivel que se defina sobre él es un índice secundario. También es un índice secundario todo aquel que se define sobre un campo de un fichero ordenado que no es el campo de ordenación.

Un índice secundario también es un fichero ordenado con dos campos: el campo de indexación (cualquier campo del fichero de datos que no es el de ordenación) y un puntero a

un bloque o un puntero a un registro. Pueden definirse muchos índices secundarios sobre un mismo fichero. Un índice secundario definido sobre un campo clave (un campo con valores distintos para cada registro) necesita una entrada por cada registro. El puntero puede ser al bloque en el que se encuentra el registro, o bien, al registro en sí. Un índice de este tipo es un índice denso.

Un índice secundario es más grande que un índice primario, ya que tiene mayor número de entradas. Sin embargo, la mejora que se introduce en el tiempo de búsqueda para un registro cualquiera es mayor para el índice secundario que para el primario, ya que hay que hacer una búsqueda lineal en el fichero de datos si el índice secundario no existe. Si un índice primario no existe, se puede hacer una búsqueda binaria sobre el fichero de datos (más rápida que la lineal).

También se puede crear un índice secundario sobre un campo no clave. En este caso, puede haber varios registros con el mismo valor en el campo de indexación. Hay varios modos de implementar un índice de este tipo:

- Incluir una entrada en el índice por cada registro (varias entradas tendrán el mismo valor), por lo que será un índice denso. El algoritmo de búsqueda binaria debe ser modificado.
- Utilizar registros de longitud variable en el índice, con un campo que contiene una lista de punteros. En el índice hay una entrada por cada valor distinto del campo de indexación, junto a una lista de punteros a todos los bloques que contienen registros con dicho valor. En este caso, también hay que modificar de modo apropiado el algoritmo de búsqueda binaria.
- Utilizar registros de longitud fija en el índice, con una entrada para cada valor del campo de indexación, pero crear un nivel extra de indirección para manejar punteros múltiples. En este índice no denso cada puntero señala a un bloque de punteros a registros; cada puntero a registro de ese bloque apunta a uno de los registros del fichero de datos con el mismo valor del campo de indexación. Si los punteros a registros no caben en un solo bloque, se puede usar una lista enlazada de bloques. El acceso a través del índice requiere un acceso adicional a bloque debido al nivel extra introducido, pero los algoritmos para buscar en el índice y los de inserción de nuevos registros en el

fichero de datos son bastante simples. Además, los accesos que tienen condiciones de búsqueda complicadas (sobre varios campos) se pueden realizar trabajando sobre los punteros y no sobre los registros, siempre que haya un índice secundario por cada uno de los campos implicados en la condición. Esta es la técnica más utilizada.

Nótese que un índice secundario proporciona un *orden lógico* de los registros según el campo de indexación. Si accedemos a los registros en el orden de las entradas del índice secundario, los obtenemos en el orden del campo de indexación. Por lo tanto, no hay que hacer una ordenación externa para leer ordenadamente según dicho campo.

2.8.2 Índices multinivel

Los índices que se han descrito hasta ahora son ficheros ordenados en los que una búsqueda requiere aproximadamente $\log_2 b$ accesos a bloques de disco, siendo b el número de bloques que tiene el índice. Es el logaritmo en base dos porque en cada paso del algoritmo se reduce a la mitad el trozo del índice en el que se va a seguir buscando. Mediante los índices multinivel se pretende reducir mucho más el trozo del índice sobre el que seguir buscando, en concreto, se quiere dividir el tamaño no entre dos, sino entre el número de registros del índice que caben en un bloque. De este modo, se reduce el espacio de búsqueda mucho más rápidamente. Si el número de registros del índice que caben en un bloque es r, entonces el buscar en un índice multinivel requiere aproximadamente $\log_r b$, que es mucho menor si $r > 2$.

Un índice multinivel está formado por un fichero índice, al que se denomina primer nivel o nivel base del índice. Este primer nivel es un fichero ordenado con un valor distinto del campo de indexación en cada entrada. Por tanto, se puede crear un índice primario sobre el primer nivel; este índice constituye el segundo nivel del índice multinivel. Ya que el segundo nivel es un índice primario, en él se tiene una entrada por cada bloque del primer nivel. Como todas las entradas de ambos niveles tienen el mismo tamaño (son registros de tamaño fijo siempre iguales), en un bloque caben siempre el mismo número de entradas: r. Si el primer nivel tiene i_1 entradas, ocupa $\lceil i_1/r \rceil$ bloques, que es el número de entradas i_2 que se necesitan en el segundo nivel. El proceso se puede repetir sobre este nivel, si es necesario. El tercer nivel, que es un índice primario sobre el segundo nivel, tiene una entrada por cada

bloque del segundo nivel, por lo tanto, el número de entradas del tercer nivel será $i_3 = \lceil i_2/r \rceil$. Nótese que se necesita un tercer nivel sólo si el segundo nivel ocupa más de un bloque. Este proceso se puede repetir hasta que todas las entradas de algún nivel n quepan en un solo bloque. Cada bloque reduce el número de entradas del nivel anterior por un factor de r, por tanto, un índice multinivel con i_1 entradas en el primer nivel tiene aproximadamente n niveles, donde $n = \lceil \log_r i_1 \rceil$.

Este esquema multinivel se puede utilizar sobre cualquier tipo de índice, sea primario, de agrupamiento o secundario, siempre que el primer nivel tenga valores distintos en el campo de indexación y entradas de tamaño fijo.

Cuando el índice del primer nivel es denso, se puede saber si existe un registro con un valor dado en el campo de indexación (test de existencia) sin tener que acceder al fichero de datos, cosa que hay que hacer siempre en un índice no denso. Por lo tanto los índices multinivel reducen el número de accesos a bloque cuando se busca un registro a partir del valor de su campo de indexación. Pero todavía siguen los problemas en inserciones y borrados sobre los índices, ya que todos ellos (todos los niveles) son ficheros ordenados. Para mantener las ventajas que presentan los índices multinivel y reducir los problemas de las inserciones y borrados, los diseñadores adoptan a menudo un índice multinivel que deja espacio en cada uno de sus bloques para insertar nuevas entradas. Esto es lo que se denomina un índice multinivel dinámico y se suele implementar mediante estructuras de datos denominadas árboles B y árboles B+.

2.8.3 Árboles B y árboles B+

Los árboles B y los árboles B+ son casos especiales de árboles de búsqueda. Un árbol de búsqueda es un tipo de árbol que sirve para guiar la búsqueda de un registro, dado el valor de uno de sus campos. Los índices multinivel de la sección anterior pueden considerarse como variaciones de los árboles de búsqueda. Cada bloque o nodo del índice multinivel puede tener hasta p valores del campo de indexación y p punteros. Los valores del campo de indexación de cada nodo guían al siguiente nodo (que se encuentra en otro nivel), hasta llegar al bloque del fichero de datos que contiene el registro deseado. Al seguir un puntero, se va restringiendo la búsqueda en cada nivel a un subárbol del árbol de búsqueda, y se ignoran todos los nodos que no estén en dicho subárbol.

Los árboles de búsqueda difieren un poco de los índices multinivel. Un árbol de búsqueda de orden p es un árbol tal que cada nodo contiene como mucho $p-1$ valores del campo de indexación y p punteros:

$$(P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_q)$$

donde $q \leq p$, cada P_i es un puntero a un nodo hijo y cada K_i es un valor de búsqueda proveniente de algún conjunto ordenado de valores. Se supone que todos los valores de búsqueda son únicos. Un árbol de búsqueda debe cumplir, en todo momento, las siguientes restricciones:

1. Dentro de cada nodo: $K_1 < K_2 < \ldots K_{q-1}$.
2. Para todos los valores X del subárbol al que apunta P_i, se tiene: $K_{i-1} < X < K_i$ para $1 < i < q$, $X < K_i$ para $i = 1$ y $K_{i-1} < X$ para $i = q$.

Al buscar un valor X siempre se sigue el puntero P_i apropiado de acuerdo con las condiciones de la segunda restricción. Para insertar valores de búsqueda en el árbol y eliminarlos, sin violar las restricciones anteriores, se utilizan algoritmos que no garantizan que el árbol de búsqueda esté equilibrado (que todas las hojas estén al mismo nivel). Es importante mantener equilibrados los árboles de búsqueda porque esto garantiza que no habrá nodos en niveles muy profundos que requieran muchos accesos a bloques durante una búsqueda. Además, las eliminaciones de registros pueden hacer que queden nodos casi vacíos, con lo que hay un desperdicio de espacio importante que también provoca un aumento en el número de niveles.

El árbol B es un árbol de búsqueda, con algunas restricciones adicionales, que resuelve hasta cierto punto los dos problemas anteriores. Estas restricciones adicionales garantizan que el árbol siempre estará equilibrado y que el espacio desperdiciado por la eliminación, si lo hay, nunca será excesivo. Los algoritmos para insertar y eliminar se hacen más complejos para poder mantener estas restricciones. No obstante, la mayor parte de las inserciones y eliminaciones son procesos simples, se complican sólo en circunstancias especiales: cuando se intenta insertar en un nodo que está lleno o cuando se intenta borrar en un nodo que está ocupado hasta la mitad.

Un árbol B de orden p se define del siguiente modo:

1. La estructura de cada nodo interno tiene la forma:

$$(P_1, (K_1, Pr_1), P_2, (K_2, Pr_2), P_3, (K_3, Pr_3), \ldots P_{q-1}, (K_{q-1}, Pr_{q-1}), P_q)$$

   donde $q \leq p$. Cada P_i es un puntero a un nodo interno del árbol y cada Pr_i es un puntero al registro del fichero de datos que tiene el valor K_i en el campo de búsqueda o de indexación.

2. Dentro de cada nodo se cumple: $K_1 < K_2 < \ldots K_{q-1}$.

3. Para todos los valores X del campo de indexación del subárbol al que apunta P_i, se cumple: $K_{i-1} < X < K_i$ para $1 < i < q$, $X < K_i$ para $i = 1$ y $K_{i-1} < X$, para $i = q$.

4. Cada nodo tiene, como mucho, p punteros a nodos del árbol.

5. Cada nodo, excepto la raíz y las hojas, tiene, al menos, $\lceil p/2 \rceil$ punteros a nodos del árbol. El nodo raíz tiene, como mínimo, dos punteros a nodos del árbol, a menos que sea el único nodo del árbol.

6. Un nodo con q punteros a nodos, $q \leq p$, tiene $q - 1$ valores del campo de indexación.

7. Todos los nodos hoja están al mismo nivel. Los nodos hoja tienen la misma estructura que los nodos internos, pero los punteros a nodos del árbol son nulos.

Como se puede observar, en los árboles B todos los valores del campo de indexación aparecen alguna vez en algún nivel del árbol, junto con un puntero al fichero de datos.

En un árbol B+ los punteros a datos se almacenan sólo en los nodos hoja del árbol, por lo cual, la estructura de los nodos hoja difiere de la de los nodos internos. Los nodos hoja tienen una entrada por cada valor del campo de indexación, junto con un puntero al registro del fichero de datos. Estos nodos están enlazados para ofrecer un acceso ordenado a los registros a través del campo de indexación. Los nodos hoja de un árbol B+ son similares al primer nivel (nivel base) de un índice. Los nodos internos del árbol B+ corresponden a los demás niveles del índice. Algunos valores del campo de indexación se repiten en los nodos internos del árbol B+ con el fin de guiar la búsqueda.

En un árbol B+ de orden p la estructura de un nodo interno es la siguiente:

1. Todo nodo interno es de la forma:

$$(P_1, K_1, P_2, K_2, P_3, K_3, \ldots P_{q-1}, K_{q-1}, P_q)$$

   donde $q \leq p$. Cada P_i es un puntero a un nodo interno del árbol.

2. Dentro de cada nodo interno se cumple: $K_1 < K_2 < \ldots K_{q-1}$.

3. Para todos los valores X del campo de indexación del subárbol al que apunta P_i, se cumple: $K_{i-1} < X \leq K_i$ para $1 < i < q$, $X \leq K_i$ para $i = 1$ y $K_{i-1} < X$ para $i = q$.

4. Cada nodo interno tiene, como mucho, p punteros a nodos del árbol.

5. Cada nodo interno, excepto la raíz, tiene, al menos, $\lceil p/2 \rceil$ punteros a nodos del árbol. El nodo raíz tiene, como mínimo, dos punteros a nodos del árbol si es un nodo interno.

6. Un nodo interno con q punteros a nodos, $q \leq p$, tiene $q - 1$ valores del campo de indexación.

La estructura de los nodos hoja de un árbol B+ de orden p es la siguiente:

1. Todo nodo hoja es de la forma:

$$((K_1, Pr_1), (K_2, Pr_2), (K_3, Pr_3), \ldots (K_{q-1}, Pr_{q-1}), P_{siguiente})$$

   donde $q \leq p$. Cada Pr_i es un puntero al registro de datos que tiene el valor K_i en el campo de indexación, y $P_{siguiente}$ es un puntero al siguiente nodo hoja del árbol.

2. Dentro de cada nodo hoja se cumple: $K_1 < K_2 < \ldots K_{q-1}$, $q \leq p$.

3. Cada nodo hoja tiene, al menos, $\lfloor p/2 \rfloor$ valores.

4. Todos los nodos hoja están al mismo nivel.

Como las entradas en los nodos internos de los árboles B+ contienen valores del campo de indexación y punteros a nodos del árbol, pero no contienen punteros a los registros del fichero de datos, es posible "empaquetar" más entradas en un nodo interno de un árbol B+ que en un nodo similar de un árbol B. Por tanto, si el tamaño de bloque (nodo) es el mismo,

el orden p será mayor para el árbol B+ que para el árbol B. Esto puede reducir el número de niveles del árbol B+, mejorándose así el tiempo de acceso. Como las estructuras de los nodos internos y los nodos hoja de los árboles B+ son diferentes, su orden p puede ser diferente.

Se ha demostrado por análisis y simulación que después de un gran número de inserciones y eliminaciones aleatorias en un árbol B, los nodos están ocupados en un 69% cuando se estabiliza el número de valores del árbol. Esto también es verdadero en el caso de los árboles B+. Si llega a suceder esto, la división y combinación de nodos ocurrirá con muy poca frecuencia, de modo que la inserción y la eliminación se volverán muy eficientes.

Cuando los árboles se definen sobre un campo no clave, los punteros a datos pasan a ser punteros a bloques de punteros a datos (se añade un nivel de indirección).

2.8.4 Ficheros dispersos como índices

También se pueden crear estructuras de acceso similares a los índices basándose en la dispersión. Las entradas del índice (K, Pr) se pueden organizar como un fichero disperso que va cambiando de tamaño mediante dispersión dinámica, extensible o lineal. El algoritmo de búsqueda aplica la función de dispersión sobre K. Una vez se ha encontrado la entrada, el puntero Pr se utiliza para localizar el registro correspondiente en el fichero de datos.

2.9 Resumen

Los discos magnéticos son los dispositivos de almacenamiento secundario más utilizados para almacenar los ficheros de una base de datos. Los datos se almacenan en sectores/bloques en el disco. El acceso a un sector/bloque es costoso debido al tiempo de búsqueda, el tiempo de rotación y el tiempo de transferencia del bloque.

Hay tres organizaciones primarias para los ficheros: desordenados, ordenados y dispersos. Los ficheros desordenados requieren una búsqueda lineal para localizar registros, pero la inserción de éstos es muy sencilla. La eliminación plantea problemas en el sentido de que se desperdicia espacio, recuperable mediante una reorganización del fichero.

Los ficheros ordenados reducen el tiempo requerido para leer registros en orden según el campo de ordenación. El tiempo necesario para buscar un registro cualquiera dado el valor de su campo de ordenación también se reduce, ya que se puede realizar una búsqueda binaria. Sin embargo, la necesidad de mantener los registros en orden hace muy costosa la inserción. Por esta razón, se puede utilizar un fichero de desborde no ordenado para reducir el coste de la inserción. Los registros de desborde se fusionan con el fichero de datos periódicamente.

La dispersión ofrece acceso muy rápido a un registro cualquiera dado el valor de su campo de dispersión. Las colisiones que causan desborde en los bloques se pueden resolver de varios modos, el más frecuente es el encadenamiento. El acceso a un registro a través de un campo que no es el campo de dispersión, es lento, y lo mismo sucede con el acceso secuencial a los registros a través de cualquier campo. Hay varias técnicas que permiten al fichero expandirse y contraerse dinámicamente: la dispersión dinámica, la extensible y la lineal.

El agrupamiento es otro modo de organizar los registros, cuyo resultado es un fichero ordenado de un modo especial. Mediante el agrupamiento se implementan físicamente las relaciones entre los registros de varios ficheros, almacenándolos consecutivos en el mismo fichero.

Los índices son estructuras de acceso adicionales con las que se mejora la eficiencia en la obtención de registros de un fichero de datos. Dichas estructuras de acceso pueden usarse junto con las organizaciones de ficheros vistas anteriormente.

Hay tres tipos de índices ordenados de un solo nivel: primarios, secundarios y de agrupamiento. Cada índice se basa en un campo del fichero. Los índices primarios y de agrupamiento se construyen según el campo de ordenamiento físico del fichero, en tanto que los índices secundarios se basan en campos que no son de ordenamiento. El campo de un índice primario debe ser además un campo con valores únicos (clave), cosa que no sucede con un índice de agrupamiento. Los índices de un solo nivel son ficheros ordenados, y se examinan mediante una búsqueda binaria. Se pueden construir índices de varios niveles (multinivel) para mejorar la eficiencia en las búsquedas.

Los árboles B y los árboles B+ son índices multinivel cuya estructura permite al índice expandirse y contraerse dinámicamente. Los nodos (bloques) de estas estructuras se mantie-

nen ocupados entre el 50% y el 100% de su capacidad gracias a sus algoritmos de inserción y eliminación. Después de cierto tiempo, los nodos se estabilizan en un grado de ocupación medio del 69%, lo que deja espacio para hacer inserciones sin tener que reorganizar el índice con mucha frecuencia. En general, los árboles B+ pueden contener más entradas en sus nodos internos que los árboles B, por lo que es posible que un árbol B+ tenga menos niveles o incluya más entradas que el árbol B equivalente.

También es posible construir índices mediante estructuras de datos que utilizan la técnica de la dispersión.

Bibliografía

Este capítulo se ha elaborado a partir del texto de Elmasri y Navathe (1997) en donde se describen las principales organizaciones de ficheros para el almacenamiento de datos en disco. En el texto se realiza un estudio de este dispositivo de almacenamiento secundario. Estos autores también realizan un análisis de las distintas estructuras de índices para el acceso a estos ficheros.

Folk y Zoellick (1992) presentan con mayor detalle todos los conceptos básicos sobre estructuras de ficheros y estudian los dispositivos de almacenamiento secundario magnéticos más utilizados: discos y cintas, así como el CD–ROM. Además, analizan de modo muy exahustivo los distintos tipos de índices, los árboles B, los árboles B+, y la dispersión, haciendo un extenso estudio de la dispersión extensible.

Capítulo 3

Sistemas de bases de datos

En este capítulo se presenta un análisis de los modelos de datos y se definen los conceptos de esquema y estado de una base de datos. Se trata la arquitectura de los sistemas de bases de datos y la independencia respecto a los datos. También se definen los distintos lenguajes que proporcionan los SGBD y se estudian varias clasificaciones de estos sistemas. Se comentan las funciones de los SGBD y los módulos de software que los componen.

3.1 Modelos de datos

Una de las características fundamentales de los sistemas de bases de datos es que proporcionan cierto nivel de abstracción de datos, al ocultar las características sobre el almacenamiento físico que la mayoría de usuarios no necesita conocer. Los modelos de datos son el instrumento principal para ofrecer dicha abstracción. Un *modelo de datos* es un conjunto de conceptos que sirven para describir la estructura de una base de datos: los datos, las relaciones entre los datos y las restricciones que deben cumplirse sobre los datos. Los modelos de datos contienen también un conjunto de operaciones básicas para la realización de consultas (lecturas) y actualizaciones de datos. Además, los modelos de datos más modernos incluyen conceptos para especificar comportamiento, permitiendo especificar un conjunto de operaciones definidas por el usuario.

Los modelos de datos se pueden clasificar dependiendo de los tipos de conceptos que

ofrecen para describir la estructura de la base de datos. Los modelos de datos de alto nivel, o *modelos conceptuales*, disponen de conceptos muy cercanos al modo en que la mayoría de los usuarios percibe los datos, mientras que los modelos de datos de bajo nivel, o *modelos físicos*, proporcionan conceptos que describen los detalles de cómo se almacenan los datos en el ordenador. Los conceptos de los modelos físicos están dirigidos al personal informático, no a los usuarios finales. Entre estos dos extremos se encuentran los *modelos lógicos*, cuyos conceptos pueden ser entendidos por los usuarios finales, aunque no están demasiado alejados de la forma en que los datos se organizan físicamente. Los modelos lógicos ocultan algunos detalles de cómo se almacenan los datos, pero pueden implementarse de manera directa en un ordenador.

Los modelos conceptuales utilizan conceptos como entidades, atributos y relaciones. Una *entidad* representa un objeto o concepto del mundo real como, por ejemplo, un empleado de la empresa inmobiliaria o una oficina. Un *atributo* representa alguna propiedad de interés de una entidad como, por ejemplo, el nombre o el salario del empleado. Una *relación* describe una interacción entre dos o más entidades, por ejemplo, la relación de trabajo entre un empleado y su oficina.

Cada SGBD soporta un modelo lógico, siendo los más comunes el relacional, el de red y el jerárquico. Estos modelos representan los datos valiéndose de estructuras de registros, por lo que también se denominan *modelos orientados a registros*. Hay una nueva familia de modelos lógicos, son los *modelos orientados a objetos*, que están más próximos a los modelos conceptuales.

Los modelos físicos describen cómo se almacenan los datos en el ordenador: el formato de los registros, la estructura de los ficheros (desordenados, ordenados, etc.) y los métodos de acceso utilizados (índices, etc.).

A la descripción de una base de datos mediante un modelo de datos se le denomina *esquema de la base de datos*. Este esquema se especifica durante el diseño, y no es de esperar que se modifique a menudo. Sin embargo, los datos que se almacenan en la base de datos pueden cambiar con mucha frecuencia: se insertan datos, se actualizan, etc. Los datos que la base de datos contiene en un determinado momento se denominan *estado de la base de datos* u *ocurrencia de la base de datos*.

La distinción entre el esquema y el estado de la base de datos es muy importante. Cuando definimos una nueva base de datos, sólo especificamos su esquema al SGBD. En ese momento, el estado de la base de datos es el "estado vacío", sin datos. Cuando se cargan datos por primera vez, la base datos pasa al "estado inicial". De ahí en adelante, siempre que se realice una operación de actualización de la base de datos, se tendrá un nuevo estado. El SGBD se encarga, en parte, de garantizar que todos los estados de la base de datos sean estados válidos que satisfagan la estructura y las restricciones especificadas en el esquema. Por lo tanto, es muy importante que el esquema que se especifique al SGBD sea correcto y se debe tener muchísimo cuidado al diseñarlo. El SGBD almacena el esquema en su catálogo o diccionario de datos, de modo que se pueda consultar siempre que sea necesario.

3.2 Arquitectura de los sistemas de bases de datos

Hay tres características importantes inherentes a los sistemas de bases de datos: la separación entre los programas de aplicación y los datos, el manejo de múltiples vistas por parte de los usuarios y el uso de un catálogo para almacenar el esquema de la base de datos. En 1975, el comité ANSI–SPARC (American National Standard Institute – Standards Planning and Requirements Committee) propuso una arquitectura de tres niveles para los sistemas de bases de datos, que resulta muy útil a la hora de conseguir estas tres características.

El objetivo de la arquitectura de tres niveles es el de separar los programas de aplicación de la base de datos física. En esta arquitectura, el esquema de una base de datos se define en tres niveles de abstracción distintos:

1. En el *nivel interno* se describe la estructura física de la base de datos mediante un *esquema interno*. Este esquema se especifica mediante un modelo físico y describe todos los detalles para el almacenamiento de la base de datos, así como los métodos de acceso.

2. En el *nivel conceptual* se describe la estructura de toda la base de datos para una comunidad de usuarios (todos los de una empresa u organización), mediante un *esquema conceptual*. Este esquema oculta los detalles de las estructuras de almacenamiento y se concentra en describir entidades, atributos, relaciones, operaciones de los usuarios y

restricciones. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar el esquema.

3. En el *nivel externo* se describen varios *esquemas externos* o *vistas de usuario*. Cada esquema externo describe la parte de la base de datos que interesa a un grupo de usuarios determinado y oculta a ese grupo el resto de la base de datos. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar los esquemas.

La mayoría de los SGBD no distinguen del todo los tres niveles. Algunos incluyen detalles del nivel físico en el esquema conceptual. En casi todos los SGBD que se manejan vistas de usuario, los esquemas externos se especifican con el mismo modelo de datos que describe la información a nivel conceptual, aunque en algunos se pueden utilizar diferentes modelos de datos en los niveles conceptual y externo.

Hay que destacar que los tres esquemas no son más que descripciones de los mismos datos pero con distintos niveles de abstracción. Los únicos datos que existen realmente están a nivel físico, almacenados en un dispositivo como puede ser un disco. En un SGBD basado en la arquitectura de tres niveles, cada grupo de usuarios hace referencia exclusivamente a su propio esquema externo. Por lo tanto, el SGBD debe transformar cualquier petición expresada en términos de un esquema externo a una petición expresada en términos del esquema conceptual, y luego, a una petición en el esquema interno, que se procesará sobre la base de datos almacenada. Si la petición es de una obtención (consulta) de datos, será preciso modificar el formato de la información extraída de la base de datos almacenada, para que coincida con la vista externa del usuario. El proceso de transformar peticiones y resultados de un nivel a otro se denomina *correspondencia* o *transformación*. Estas correspondencias pueden requerir bastante tiempo, por lo que algunos SGBD no cuentan con vistas externas.

La arquitectura de tres niveles es útil para explicar el concepto de *independencia de datos* que podemos definir como la capacidad para modificar el esquema en un nivel del sistema sin tener que modificar el esquema del nivel inmediato superior. Se pueden definir dos tipos de independencia de datos:

- La *independencia lógica* es la capacidad de modificar el esquema conceptual sin tener que alterar los esquemas externos ni los programas de aplicación. Se puede modificar

el esquema conceptual para ampliar la base de datos o para reducirla. Si, por ejemplo, se reduce la base de datos eliminando una entidad, los esquemas externos que no se refieran a ella no deberán verse afectados.

- La *independencia física* es la capacidad de modificar el esquema interno sin tener que alterar el esquema conceptual (o los externos). Por ejemplo, puede ser necesario reorganizar ciertos ficheros físicos con el fin de mejorar el rendimiento de las operaciones de consulta o de actualización de datos. Dado que la independencia física se refiere sólo a la separación entre las aplicaciones y las estructuras físicas de almacenamiento, es más fácil de conseguir que la independencia lógica.

En los SGBD que tienen la arquitectura de varios niveles es necesario ampliar el catálogo o diccionario, de modo que incluya información sobre cómo establecer la correspondencia entre las peticiones de los usuarios y los datos, entre los diversos niveles. El SGBD utiliza una serie de procedimientos adicionales para realizar estas correspondencias haciendo referencia a la información de correspondencia que se encuentra en el catálogo. La independencia de datos se consigue porque al modificarse el esquema en algún nivel, el esquema del nivel inmediato superior permanece sin cambios, sólo se modifica la correspondencia entre los dos niveles. No es preciso modificar los programas de aplicación que hacen referencia al esquema del nivel superior.

Por lo tanto, la arquitectura de tres niveles puede facilitar la obtención de la verdadera independencia de datos, tanto física como lógica. Sin embargo, los dos niveles de correspondencia implican un gasto extra durante la ejecución de una consulta o de un programa, lo cual reduce la eficiencia del SGBD. Es por esto que muy pocos SGBD han implementado esta arquitectura completa.

3.3 Lenguajes de los sistemas de gestión de bases de datos

Los SGBD deben ofrecer lenguajes e interfaces apropiadas para cada tipo de usuario: administradores de la base de datos, diseñadores, programadores de aplicaciones y usuarios finales.

3.3.1 Lenguaje de definición de datos

Una vez finalizado el diseño de una base de datos y escogido un SGBD para su implementación, el primer paso consiste en especificar el esquema conceptual y el esquema interno de la base de datos, y la correspondencia entre ambos. En muchos SGBD no se mantiene una separación estricta de niveles, por lo que el administrador de la base de datos y los diseñadores utilizan el mismo lenguaje para definir ambos esquemas, es el *lenguaje de definición de datos* (LDD). El SGBD posee un compilador de LDD cuya función consiste en procesar las sentencias del lenguaje para identificar las descripciones de los distintos elementos de los esquemas y almacenar la descripción del esquema en el catálogo o diccionario de datos. Se dice que el diccionario contiene *metadatos*: describe los objetos de la base de datos.

Cuando en un SGBD hay una clara separación entre los niveles conceptual e interno, el LDD sólo sirve para especificar el esquema conceptual. Para especificar el esquema interno se utiliza un *lenguaje de definición de almacenamiento* (LDA). Las correspondencias entre ambos esquemas se pueden especificar en cualquiera de los dos lenguajes. Para tener una verdadera arquitectura de tres niveles sería necesario disponer de un tercer lenguaje, el *lenguaje de definición de vistas* (LDV), que se utilizaría para especificar las vistas de los usuarios y su correspondencia con el esquema conceptual.

3.3.2 Lenguaje de manejo de datos

Una vez creados los esquemas de la base de datos, los usuarios necesitan un lenguaje que les permita manipular los datos de la base de datos: realizar consultas, inserciones, eliminaciones y modificaciones. Este lenguaje es el que se denomina *lenguaje de manejo de datos* (LMD).

Hay dos tipos de LMD: los procedurales y los no procedurales. Con un *LMD procedural* el usuario (normalmente será un programador) especifica qué datos se necesitan y cómo hay que obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la información requerida. Estos lenguajes acceden a un registro, lo procesan y basándose en los resultados obtenidos, acceden a otro registro, que también deben procesar. Así se va accediendo a

registros y se van procesando hasta que se obtienen los datos deseados. Las sentencias de un LMD procedural deben estar embebidas en un lenguaje de alto nivel, ya que se necesitan sus estructuras (bucles, condicionales, etc.) para obtener y procesar cada registro individual. A este lenguaje se le denomina *lenguaje anfitrión*. Las bases de datos jerárquicas y de red utilizan LMD procedurales.

Un *LMD no procedural* se puede utilizar de manera independiente para especificar operaciones complejas sobre la base de datos de forma concisa. En muchos SGBD se pueden introducir interactivamente instrucciones del LMD desde un terminal o bien embeberlas en un lenguaje de programación de alto nivel. Los LMD no procedurales permiten especificar los datos a obtener en una consulta o los datos que se deben actualizar, mediante una sola y sencilla sentencia. El usuario o programador especifica qué datos quiere obtener sin decir cómo se debe acceder a ellos. El SGBD traduce las sentencias del LMD en uno o varios procedimientos que manipulan los conjuntos de registros necesarios. Esto libera al usuario de tener que conocer cuál es la estructura física de los datos y qué algoritmos se deben utilizar para acceder a ellos. A los LMD no procedurales también se les denomina *declarativos*. Las bases de datos relacionales utilizan LMD no procedurales, como SQL (Structured Query Language) o QBE (Query–By–Example). Los lenguajes no procedurales son más fáciles de aprender y de usar que los procedurales, y el usuario debe realizar menos trabajo, siendo el SGBD quien hace la mayor parte.

La parte de los LMD no procedurales que realiza la obtención de datos es lo que se denomina un *lenguaje de consultas*. En general, las órdenes tanto de obtención como de actualización de datos de un LMD no procedural se pueden utilizar interactivamente, por lo que al conjunto completo de sentencias del LMD se le denomina lenguaje de consultas, aunque es técnicamente incorrecto.

3.3.3 Lenguajes de cuarta generación

No existe consenso sobre lo que es un *lenguaje de cuarta generación* (4GL). Lo que en un lenguaje de tercera generación (3GL) como COBOL requiere cientos de líneas de código, tan solo necesita diez o veinte líneas en un 4GL. Comparado con un 3GL, que es procedural, un 4GL es un lenguaje no procedural: el usuario define qué se debe hacer, no cómo debe

hacerse. Los 4GL se apoyan en unas herramientas de mucho más alto nivel denominadas herramientas de cuarta generación. El usuario no debe definir los pasos a seguir en un programa para realizar una determinada tarea, tan sólo debe definir una serie de parámetros que estas herramientas utilizarán para generar un programa de aplicación. Se dice que los 4GL pueden mejorar la productividad de los programadores en un factor de 10, aunque se limita el tipo de problemas que pueden resolver. Los 4GL abarcan:

- Lenguajes de presentación, como lenguajes de consultas y generadores de informes.
- Lenguajes especializados, como hojas de cálculo y lenguajes de bases de datos.
- Generadores de aplicaciones que definen, insertan, actualizan y obtienen datos de la base de datos.
- Lenguajes de muy alto nivel que se utilizan para generar el código de la aplicación.

Los lenguajes SQL y QBE son ejemplos de 4GL. Hay otros tipos de 4GL:

- Un *generador de formularios* es una herramienta interactiva que permite crear rápidamente formularios de pantalla para introducir o visualizar datos. Los generadores de formularios permiten que el usuario defina el aspecto de la pantalla, qué información se debe visualizar y en qué lugar de la pantalla debe visualizarse. Algunos generadores de formularios permiten la creación de atributos derivados utilizando operadores aritméticos y también permiten especificar controles para la validación de los datos de entrada.
- Un *generador de informes* es una herramienta para crear informes a partir de los datos almacenados en la base de datos. Se parece a un lenguaje de consultas en que permite al usuario hacer preguntas sobre la base de datos y obtener información de ella para un informe. Sin embargo, en el generador de informes se tiene un mayor control sobre el aspecto de la salida. Se puede dejar que el generador determine automáticamente el aspecto de la salida o se puede diseñar ésta para que tenga el aspecto que desee el usuario final.

- Un *generador de gráficos* es una herramienta para obtener datos de la base de datos y visualizarlos en un gráfico mostrando tendencias y relaciones entre datos. Normalmente se pueden diseñar distintos tipos de gráficos: barras, líneas, etc.

- Un *generador de aplicaciones* es una herramienta para crear programas que hagan de interface entre el usuario y la base de datos. El uso de un generador de aplicaciones puede reducir el tiempo que se necesita para diseñar un programa de aplicación. Los generadores de aplicaciones constan de procedimientos que realizan las funciones fundamentales que se utilizan en la mayoría de los programas. Estos procedimientos están escritos en un lenguaje de programación de alto nivel y forman una librería de funciones entre las que escoger. El usuario especifica qué debe hacer el programa y el generador de aplicaciones es quien determina cómo realizar la tarea.

3.4 Clasificación de los sistemas de gestión de bases de datos

El criterio principal que se utiliza para clasificar los SGBD es el modelo lógico en que se basan. Los modelos lógicos empleados con mayor frecuencia en los SGBD comerciales actuales son el relacional, el de red y el jerárquico. Algunos SGBD más modernos se basan en modelos orientados a objetos.

El *modelo relacional* se basa en el concepto matemático denominado "relación", que gráficamente se puede representar como una tabla. En el modelo relacional, los datos y las relaciones existentes entre los datos se representan mediante estas relaciones matemáticas, cada una con un nombre que es único y con un conjunto de columnas.

En el modelo relacional la base de datos es percibida por el usuario como un conjunto de tablas. Esta percepción es sólo a nivel lógico (en los niveles externo y conceptual de la arquitectura de tres niveles), ya que a nivel físico puede estar implementada mediante distintas estructuras de almacenamiento.

En el *modelo de red* los datos se representan como colecciones de registros y las relaciones entre los datos se representan mediante conjuntos, que son punteros en la implementación física. Los registros se organizan como un grafo: los registros son los nodos y los arcos son los conjuntos. El SGBD de red más popular es el sistema IDMS.

El *modelo jerárquico* es un tipo de modelo de red con algunas restricciones. De nuevo los datos se representan como colecciones de registros y las relaciones entre los datos se representan mediante conjuntos. Sin embargo, en el modelo jerárquico cada nodo puede tener un solo padre. Una base de datos jerárquica puede representarse mediante un árbol: los registros son los nodos, también denominados segmentos, y los arcos son los conjuntos. El SGBD jerárquico más importante es el sistema IMS.

La mayoría de los SGBD comerciales actuales están basados en el modelo relacional, mientras que los sistemas más antiguos estaban basados en el modelo de red o el modelo jerárquico. Estos dos últimos modelos requieren que el usuario tenga conocimiento de la estructura física de la base de datos a la que se accede, mientras que el modelo relacional proporciona una mayor independencia de datos. Se dice que el modelo relacional es declarativo (se especifica qué datos se han de obtener) y los modelos de red y jerárquico son navegacionales (se especifica cómo se deben obtener los datos).

El *modelo orientado a objetos* define una base de datos en términos de objetos, sus propiedades y sus operaciones. Los objetos con la misma estructura y comportamiento pertenecen a una clase, y las clases se organizan en jerarquías o grafos acíclicos. Las operaciones de cada clase se especifican en términos de procedimientos predefinidos denominados métodos. Algunos SGBD relacionales existentes en el mercado han estado extendiendo sus modelos para incorporar conceptos orientados a objetos. A estos SGBD se les conoce como sistemas *objeto–relacionales*

Un segundo criterio para clasificar los SGBD es el número de usuarios a los que da servicio el sistema. Los sistemas *monousuario* sólo atienden a un usuario a la vez, y su principal uso se da en los ordenadores personales. Los sistemas *multiusuario*, entre los que se encuentran la mayor parte de los SGBD, atienden a varios usuarios al mismo tiempo.

Un tercer criterio es el número de sitios en los que está distribuida la base de datos. Casi todos los SGBD son *centralizados*: sus datos se almacenan en un solo computador. Los SGBD centralizados pueden atender a varios usuarios, pero el SGBD y la base de datos en sí residen por completo en una sola máquina. En los SGBD *distribuidos* la base de datos real y el propio software del SGBD pueden estar distribuidos en varios sitios conectados por una red. Los SGBD *distribuidos homogéneos* utilizan el mismo SGBD en múltiples sitios.

Una tendencia reciente consiste en crear software para tener acceso a varias bases de datos autónomas preexistentes almacenadas en SGBD *distribuidos heterogéneos*. Esto da lugar a los SGBD *federados* o *sistemas multibase de datos* en los que los SGBD participantes tienen cierto grado de autonomía local. Muchos SGBD distribuidos emplean una arquitectura cliente–servidor.

Un cuarto criterio es el coste del SGBD. La mayor parte de los paquetes de SGBD cuestan entre 10.000 y 100.000 euros. Los sistemas monousuario más económicos para microcomputadores cuestan entre 100 y 3.000 euros. En el otro extremo, los paquetes más completos cuestan más de 100.000 euros.

Por último, los SGBD pueden ser de *propósito general* o de *propósito específico*. Cuando el rendimiento es fundamental, se puede diseñar y construir un SGBD de propósito especial para una aplicación específica, y este sistema no sirve para otras aplicaciones. Muchos sistemas de reservas de líneas aéreas son SGBD de propósito especial y pertenecen a la categoría de *sistemas de procesamiento de transacciones en línea* (OLTP), que deben atender un gran número de transacciones concurrentes sin imponer excesivos retrasos.

3.5 Funciones de los sistemas de gestión de bases de datos

Codd, el creador del modelo relacional, ha establecido una lista con los ocho servicios que debe ofrecer todo SGBD.

1. Un SGBD debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos. Esta es la función fundamental de un SGBD y por supuesto, el SGBD debe ocultar al usuario la estructura física interna (la organización de los ficheros y las estructuras de almacenamiento).

2. Un SGBD debe proporcionar un catálogo en el que se almacenen las descripciones de los datos y que sea accesible por los usuarios. Este catálogo es lo que se denomina diccionario de datos y contiene información que describe los datos de la base de datos (metadatos). Normalmente, un diccionario de datos almacena:

   - Nombre, tipo y tamaño de los datos.

- Nombre de las relaciones entre los datos.
- Restricciones de integridad sobre los datos.
- Nombre de los usuarios autorizados a acceder a la base de datos.
- Esquemas externos, conceptual e interno, y correspondencia entre los esquemas.
- Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

Algunos de los beneficios que reporta el diccionario de datos son los siguientes:

- La información sobre los datos se puede almacenar de un modo centralizado. Esto ayuda a mantener el control sobre los datos, como un recurso que son.
- El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.
- La comunicación se simplifica ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen los datos o que los acceden.
- Las redundancias y las inconsistencias se pueden identificar más fácilmente ya que los datos están centralizados.
- Se puede tener un historial de los cambios realizados sobre la base de datos.
- El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.
- Se puede hacer respetar la seguridad.
- Se puede garantizar la integridad.
- Se puede proporcionar información para auditorías.

3. Un SGBD debe proporcionar un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una *transacción* es un conjunto de acciones que cambian el contenido de la base de datos. Una transacción en el sistema informático de la empresa inmobiliaria sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más

complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización, por ejemplo porque falla el hardware, la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente.

4. Un SGBD debe proporcionar un mecanismo que asegure que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir unos con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pueden interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo.

5. Un SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Este fallo puede ser a causa de un fallo en algún dispositivo hardware o un error del software, que hagan que el SGBD aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte antes de que finalice. En todos estos casos, el SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado consistente.

6. Un SGBD debe proporcionar un mecanismo que garantice que sólo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.

7. Un SGBD debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera

de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea comercialmente viable.

8. Un SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar. Por ejemplo, se puede establecer la restricción de que cada empleado no puede tener asignados más de diez inmuebles. En este caso sería deseable que el SGBD controlara que no se sobrepase este límite cada vez que se asigne un inmueble a un empleado.

Además, de estos ocho servicios, es razonable esperar que los SGBD proporcionen un par de servicios más:

9. Un SGBD debe permitir que se mantenga la independencia entre los programas y la estructura de la base de datos. La independencia de datos se alcanza mediante las vistas o subesquemas. La independencia de datos física es más fácil de alcanzar, de hecho hay varios tipos de cambios que se pueden realizar sobre la estructura física de la base de datos sin afectar a las vistas. Sin embargo, lograr una completa independencia de datos lógica es más difícil. Añadir una nueva entidad, un atributo o una relación puede ser sencillo, pero no es tan sencillo eliminarlos.

10. Un SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo. Algunas herramientas trabajan a nivel externo, por lo que habrán sido producidas por el administrador de la base de datos. Las herramientas que trabajan a nivel interno deben ser proporcionadas por el distribuidor del SGBD. Algunas de ellas son:

    - Herramientas para importar y exportar datos.

- Herramientas para monitorizar el uso y el funcionamiento de la base de datos.
- Programas de análisis estadístico para examinar las prestaciones o las estadísticas de utilización.
- Herramientas para reorganización de índices.
- Herramientas para aprovechar el espacio dejado en el almacenamiento físico por los registros borrados y que consoliden el espacio liberado para reutilizarlo cuando sea necesario.

3.6 Componentes de un sistema de gestión de bases de datos

Los SGBD son paquetes de software muy complejos y sofisticados que deben proporcionar los servicios comentados en la sección anterior. No se puede generalizar sobre los elementos que componen un SGBD ya que varían mucho unos de otros. Sin embargo, es muy útil conocer sus componentes y cómo se relacionan cuando se trata de comprender lo que es un sistema de bases de datos.

Un SGBD tiene varios módulos, cada uno de los cuales realiza una función específica. El sistema operativo proporciona servicios básicos al SGBD, que es construido sobre él.

- El *procesador de consultas* es el componente principal de un SGBD. Transforma las consultas en un conjunto de instrucciones de bajo nivel que se dirigen al gestor de la base de datos.
- El *gestor de la base de datos* es el interface con los programas de aplicación y las consultas de los usuarios. El gestor de la base de datos acepta consultas y examina los esquemas externo y conceptual para determinar qué registros se requieren para satisfacer la petición. Entonces el gestor de la base de datos realiza una llamada al gestor de ficheros para ejecutar la petición.
- El *gestor de ficheros* maneja los ficheros en disco en donde se almacena la base de datos. Este gestor establece y mantiene la lista de estructuras e índices definidos en el esquema interno. Si se utilizan ficheros dispersos, llama a la función de dispersión para generar la dirección de los registros. Pero el gestor de ficheros no realiza directamente la

entrada y salida de datos. Lo que hace es pasar la petición a los métodos de acceso del sistema operativo que se encargan de leer o escribir los datos en el buffer del sistema.

- El *preprocesador del LMD* convierte las sentencias del LMD embebidas en los programas de aplicación, en llamadas a funciones estándar escritas en el lenguaje anfitrión. El preprocesador del LMD debe trabajar con el procesador de consultas para generar el código apropiado.
- El *compilador del LDD* convierte las sentencias del LDD en un conjunto de tablas que contienen metadatos. Estas tablas se almacenan en el diccionario de datos.
- El *gestor del diccionario* controla los accesos al diccionario de datos y se encarga de mantenerlo. La mayoría de los componentes del SGBD acceden al diccionario de datos.

Los principales componentes del gestor de la base de datos son los siguientes:

- *Control de autorización.* Este módulo comprueba que el usuario tiene los permisos necesarios para llevar a cabo la operación que solicita.
- *Procesador de comandos.* Una vez que el sistema ha comprobado los permisos del usuario, se pasa el control al procesador de comandos.
- *Control de la integridad.* Cuando una operación cambia los datos de la base de datos, este módulo debe comprobar que la operación a realizar satisface todas las restricciones de integridad necesarias.
- *Optimizador de consultas.* Este módulo determina la estrategia óptima para la ejecución de las consultas.
- *Gestor de transacciones.* Este módulo realiza el procesamiento de las transacciones.
- *Planificador (scheduler).* Este módulo es el responsable de asegurar que las operaciones que se realizan concurrentemente sobre la base de datos tienen lugar sin conflictos.
- *Gestor de recuperación.* Este módulo garantiza que la base de datos permanece en un estado consistente en caso de que se produzca algún fallo.

- *Gestor de buffers.* Este módulo es el responsable de transferir los datos entre memoria principal y los dispositivos de almacenamiento secundario. A este módulo también se le denomina *gestor de datos.*

3.7 Resumen

Un modelo de datos es un conjunto de conceptos que se utilizan para describir el esquema de una base de datos, la operaciones para manejar los datos y el conjunto de reglas de integridad. Hay tres categorías principales de modelos de datos: modelos conceptuales, modelos lógicos y modelos físicos.

Es importante distinguir entre el esquema (descripción de una base de datos) y la base de datos en sí misma. El esquema no cambia a menudo, en tanto que la base de datos cambia cada vez que se insertan, eliminan o modifican datos.

La arquitectura de los sistemas de bases de datos establecida por ANSI–SPARC utiliza tres niveles de abstracción: externo, conceptual e interno. En el nivel externo, el esquema consta de las distintas visiones que tienen los usuarios de la base de datos. En el nivel conceptual, el esquema es la visión común de la base de datos. Especifica el contenido de información de la base de datos independientemente de las consideraciones de almacenamiento. En el nivel interno, el esquema es la visión que el ordenador tiene de la base de datos. Especifica cómo se representan los datos, en qué orden se almacenan los registros, qué índices y punteros se han creado y qué esquema de dispersión se ha utilizado, si es el caso.

Todo SGBD que separe los tres niveles deberá tener correspondencias entre los esquemas para transformar las peticiones de los usuarios y los resultados, de un nivel al siguiente. La mayoría de los SGBD no separan los tres niveles por completo.

La independencia de datos hace que cada nivel de la arquitectura sea inmune a los cambios en los niveles de debajo. La independencia de datos lógica se refiere a la inmunidad de los esquemas externos frente a los cambios en el esquema conceptual. La independencia de datos física se refiere a la inmunidad del esquema conceptual frente a los cambios en el esquema interno.

Un lenguaje de base de datos consta de dos partes: un lenguaje de definición de datos (LDD) y un lenguaje de manejo de datos (LMD). El LDD se utiliza para especificar el esquema de la base de datos, las vistas de los usuarios y las estructuras de almacenamiento, mientras que el LMD se utiliza para leer y actualizar los datos de la base de datos.

Los SGBD se pueden clasificar de acuerdo con el modelo lógico que soportan, el número de usuarios, el número de puestos, el coste y la generalidad. La clasificación más importante de los SGBD se basa en el modelo lógico, siendo los principales modelos que se utilizan en el mercado el relacional, el de red, el jerárquico y el orientado a objetos.

Los SGBD son sistemas informáticos muy complejos formados por una serie de componentes, cada uno de ellos con una función específica. Además de estos módulos, los SGBD cuentan con una serie de herramientas que ayudan al administrador de la base de datos a manejar el sistema.

Bibliografía

La mayoría de los libros de texto sobre bases de datos analizan los diversos conceptos que se han presentado en este capítulo. Concretamente, para su elaboración se han utilizado los textos de Elmasri y Navathe (1997), Connolly, Begg y Strachan (1996) y Date (1993).

Capítulo 4

El modelo relacional

En este capítulo se presenta el modelo relacional, que es el modelo lógico en el que se basan la mayoría de los SGBD comerciales en uso hoy en día. En primer lugar, se trata la descripción de los principios básicos del modelo relacional: la estructura de datos relacional y las reglas de integridad. A continuación, se presenta un tratamiento detallado del álgebra relacional, que es un conjunto de operaciones para manipular la estructura de datos relacional y especificar consultas de datos. El álgebra relacional es un lenguaje procedural, mientras que el cálculo relacional, que también se estudia en este capítulo, es un lenguaje equivalente no procedural.

4.1 Introducción

Es una buena justificación para estudiar la teoría que hay tras el modelo relacional, la que da Hernández (1997):

"Muchas disciplinas (y sus metodologías de diseño asociadas) tienen algún tipo de base teórica. Los ingenieros industriales diseñan estructuras utilizando teorías de la física. Los compositores crean sinfonías utilizando conceptos de teoría de la música. La industria del automóvil utiliza teorías de la aerodinámica para diseñar automóviles con menor consumo. La industria aeronáutica utiliza las mismas teorías para diseñar alas de aviones que reduzcan la resistencia al viento.

Estos ejemplos demuestran que la teoría es muy importante. La ventaja principal de la teoría es que hace que las cosas sean predecibles: nos permite predecir qué ocurrirá si realizamos una determinada acción. Por ejemplo, sabemos que si soltamos una piedra, caerá al suelo. Si somos rápidos, podemos apartar nuestros pies del camino de la teoría de la gravedad de Newton. Lo importante es que siempre funciona. Si ponemos una piedra plana encima de otra piedra plana, podemos predecir que se quedarán tal y como las hemos puesto. Esta teoría permite diseñar pirámides, catedrales y casas de ladrillos. Consideremos ahora el ejemplo de una base de datos relacional. Sabemos que si un par de tablas están relacionadas, podemos extraer datos de las dos a la vez, simplemente por el modo en que funciona la teoría de las bases de datos relacionales. Los datos que se saquen de las dos tablas se basarán en los valores coincidentes del campo que ambas tienen en común. Una vez más, nuestras acciones tienen un resultado predecible.

El modelo relacional se basa en dos ramas de las matemáticas: la teoría de conjuntos y la lógica de predicados de primer orden. El hecho de que el modelo relacional esté basado en la teoría de las matemáticas es lo que lo hace tan seguro y robusto. Al mismo tiempo, estas ramas de las matemáticas proporcionan los elementos básicos necesarios para crear una base de datos relacional con una buena estructura, y proporcionan las líneas que se utilizan para formular buenas metodologías de diseño.

Hay quien ofrece una cierta resistencia a estudiar complicados conceptos matemáticos para tan sólo llevar a cabo una tarea bastante concreta. Es habitual escuchar quejas sobre que las teorías matemáticas en las que se basa el modelo relacional y sus metodologías de diseño, no tienen relevancia en el mundo real o que no son prácticas. No es cierto: las matemáticas son básicas en el modelo relacional. Pero, por fortuna, no hay que aprender teoría de conjuntos o lógica de predicados de primer orden para utilizar el modelo relacional. Sería como decir que hay que saber todos los detalles de la aerodinámica para poder conducir un automóvil. Las teorías de la aerodinámica ayudan a entender cómo un automóvil puede ahorrar combustible, pero desde luego no son necesarias para manejarlo.

La teoría matemática proporciona la base para el modelo relacional y, por lo tanto, hace que el modelo sea predecible, fiable y seguro. La teoría describe los elementos básicos que se utilizan para crear una base de datos relacional y proporciona las líneas a seguir para construirla. El organizar estos elementos para conseguir el resultado deseado es lo que se

denomina diseño."

4.2 El modelo relacional

En 1970, el modo en que se veían las bases de datos cambió por completo cuando E. F. Codd introdujo el modelo relacional. En aquellos momentos, el enfoque existente para la estructura de las bases de datos utilizaba punteros físicos (direcciones de disco) para relacionar registros de distintos ficheros. Si, por ejemplo, se quería relacionar un registro A con un registro B, se debía añadir al registro A un campo conteniendo la dirección en disco del registro B. Este campo añadido, un puntero físico, siempre señalaría desde el registro A al registro B. Codd demostró que estas bases de datos limitaban en gran medida los tipos de operaciones que los usuarios podían realizar sobre los datos. Además, estas bases de datos eran muy vulnerables a cambios en el entorno físico. Si se añadían los controladores de un nuevo disco al sistema y los datos se movían de una localización física a otra, se requería una conversión de los ficheros de datos. Estos sistemas se basaban en el modelo de red y el modelo jerárquico, los dos modelos lógicos que constituyeron la primera generación de los SGBD.

El modelo relacional representa la segunda generación de los SGBD. En él, todos los datos están estructurados a nivel lógico como tablas formadas por filas y columnas, aunque a nivel físico pueden tener una estructura completamente distinta. Un punto fuerte del modelo relacional es la sencillez de su estructura lógica. Pero detrás de esa simple estructura hay un fundamento teórico importante del que carecen los SGBD de la primera generación, lo que constituye otro punto a su favor.

Dada la popularidad del modelo relacional, muchos sistemas de la primera generación se han modificado para proporcionar una interfaz de usuario relacional, con independencia del modelo lógico que soportan (de red o jerárquico). Por ejemplo, el sistema de red IDMS ha evolucionado a IDMS/R e IDMS/SQL, ofreciendo una visión relacional de los datos.

En los últimos años, se han propuesto algunas extensiones al modelo relacional para capturar mejor el significado de los datos, para disponer de los conceptos de la orientación a objetos y para disponer de capacidad deductiva.

El modelo relacional, como todo modelo de datos, tiene que ver con tres aspectos de los datos:

- Estructura de datos.
- Integridad de datos.
- Manejo de datos.

4.3 Estructura de datos relacional

En este apartado se presenta la estructura de datos del modelo relacional: la *relación*.

4.3.1 Relaciones

Definiciones informales

El modelo relacional se basa en el concepto matemático de *relación*, que gráficamente se representa mediante una tabla. Codd, que era un experto matemático, utilizó una terminología perteneciente a las matemáticas, en concreto de la teoría de conjuntos y de la lógica de predicados.

Una relación es una tabla con columnas y filas. Un SGBD sólo necesita que el usuario pueda percibir la base de datos como un conjunto de tablas. Esta percepción sólo se aplica a la estructura lógica de la base de datos (en el nivel externo y conceptual de la arquitectura de tres niveles ANSI–SPARC). No se aplica a la estructura física de la base de datos, que se puede implementar con distintas estructuras de almacenamiento.

Un atributo es el nombre de una columna de una relación. En el modelo relacional, las relaciones se utilizan para almacenar información sobre los objetos que se representan en la base de datos. Una relación se representa gráficamente como una tabla bidimensional en la que las filas corresponden a registros individuales y las columnas corresponden a los campos o atributos de esos registros. Los atributos pueden aparecer en la relación en cualquier orden.

Por ejemplo, la información de las oficinas de la empresa inmobiliaria se representa mediante la relación `OFICINA`, que tiene columnas para los atributos `Onum` (número de oficina), `Calle`, `Area`, `Población`, `Teléfono` y `Fax`. La información sobre la plantilla se representa mediante la relación `PLANTILLA`, que tiene columnas para los atributos `Enum` (número de empleado), `Nombre`, `Apellido`, `Dirección`, `Teléfono`, `Puesto`, `Fecha_nac`, `Salario`, `DNI`, `Onum` (número de la oficina a la que pertenece el empleado). A continuación se muestra una instancia de la relación `OFICINA` y una instancia de la relación `PLANTILLA`. Como se puede observar, cada columna contiene valores de un solo atributo. Por ejemplo, la columna `Onum` sólo contiene números de oficinas que existen.

OFICINA

Onum	Calle	Area	Población	Teléfono	Fax
05	Enmedio, 8	Centro	Castellón	964 201 240	964 201 340
07	Moyano, s/n	Centro	Castellón	964 215 760	964 215 670
03	San Miguel, 1		Villarreal	964 520 250	964 520 255
04	Trafalgar, 23	Grao	Castellón	964 284 440	964 284 420
02	Cedre, 26		Villarreal	964 525 810	964 252 811

PLANTILLA

Enum	Nombre	Apellido	Dirección	Teléfono	Puesto	Fecha_nac	Salario	DNI	Onum
EL21	Amelia	Pastor	Magallanes, 15 Castellón	964 284 560	Director	12/10/62	30000	39432212E	05
EG37	Pedro	Cubedo	Bayarri, 11 Villarreal	964 535 690	Supervisor	24/3/57	18000	38766623X	03
EG14	Luis	Collado	Borriol, 35 Villarreal	964 522 230	Administ.	9/5/70	12000	24391223L	03
EA9	Rita	Renau	Casalduch, 32 Castellón	964 257 550	Supervisor	19/5/60	18000	39233190F	07
EG5	Julio	Prats	Melilla, 23 Villarreal	964 524 590	Director	19/12/50	24000	25644309X	03
EL41	Carlos	Baeza	Herrero, 51 Castellón	964 247 250	Supervisor	29/2/67	18000	39552133T	05

Un dominio es el conjunto de valores legales de uno o varios atributos. Los dominios constituyen una poderosa característica del modelo relacional. Cada atributo de una base de datos relacional se define sobre un dominio, pudiendo haber varios atributos definidos sobre el mismo dominio. La siguiente tabla muestra los dominios de los atributos de la relación `OFICINA`. Nótese que en esta relación hay dos atributos que están definidos sobre el mismo

dominio, `Teléfono` y `Fax`.

Atributo	*Nombre del Dominio*	*Descripción*	*Definición*
`Onum`	`NUM_OFICINA`	Posibles valores de número de oficina	3 caracteres; rango O1–O99
`Calle`	`NOM_CALLE`	Nombres de calles de España	25 caracteres
`Area`	`NOM_AREA`	Nombres de áreas de las poblaciones de España	20 caracteres
`Población`	`NOM_POBLACION`	Nombres de las poblaciones de España	15 caracteres
`Teléfono`	`NUM_TEL_FAX`	Números de teléfono de España	9 caracteres
`Fax`	`NUM_TEL_FAX`	Números de teléfono de España	9 caracteres

El concepto de dominio es importante porque permite que el usuario defina, en un lugar común, el significado y la fuente de los valores que los atributos pueden tomar. Esto hace que haya más información disponible para el sistema cuando éste va a ejecutar una operación relacional, de modo que las operaciones que son semánticamente incorrectas, se pueden evitar. Por ejemplo, no tiene sentido comparar el nombre de una calle con un número de teléfono, aunque los dos atributos sean cadenas de caracteres. Sin embargo, el importe mensual del alquiler de un inmueble no estará definido sobre el mismo dominio que el número de meses que dura el alquiler, pero sí tiene sentido multiplicar los valores de ambos dominios para averiguar el importe total al que asciende el alquiler. Los SGBD relacionales no ofrecen un soporte completo de los dominios ya que su implementación es extremadamente compleja.

Una tupla es una fila de una relación. Los elementos de una relación son las tuplas o filas de la tabla. En la relación `OFICINA`, cada tupla tiene seis valores, uno para cada atributo. Las tuplas de una relación no siguen ningún orden.

El grado de una relación es el número de atributos que contiene. La relación `OFICINA` es de grado seis porque tiene seis atributos. Esto quiere decir que cada fila de la tabla es una tupla con seis valores. El grado de una relación no cambia con frecuencia.

La cardinalidad de una relación es el número de tuplas que contiene. Ya que en las relaciones se van insertando y borrando tuplas a menudo, la cardinalidad de las mismas varía constantemente.

Una base de datos relacional es un conjunto de relaciones normalizadas.

Definiciones formales

Una *relación* R definida sobre un conjunto de *dominios* $D_1, D_2, \ldots, D_n$ consta de:

- *Cabecera:* conjunto fijo de pares *atributo:dominio*

  $$\{(A_1 : D_1), (A_2 : D_2), \ldots (A_n : D_n)\}$$

  donde cada atributo A_j corresponde a un único dominio D_j y todos los A_j son distintos, es decir, no hay dos atributos que se llamen igual. El grado de la relación R es n.

- *Cuerpo:* conjunto variable de *tuplas*. Cada tupla es un conjunto de pares *atributo:valor*:

  $$\{(A_1 : v_{i1}), (A_2 : v_{i2}), \ldots (A_n : v_{in})\}$$

  con $i = 1, 2, \ldots m$, donde m es la cardinalidad de la relación R. En cada par $(A_j : v_{ij})$ se tiene que $v_{ij} \in D_j$.

La relación `OFICINA` tiene la siguiente cabecera:

{`(Onum:NUM_OFICINA), (Calle:NOM_CALLE), (Area:NOM_AREA),`
`(Población:NOM_POBLACION), (Teléfono:NUM_TEL_FAX), (Fax:NUM_TEL_FAX)`}.

Siendo la siguiente una de sus tuplas:

{`(Onum:05), (Calle:Enmedio,8), (Area:Centro),`
`(Población:Castellón), (Teléfono:964 201 240), (Fax:964 201 340)`}.

Este conjunto de pares no está ordenado, por lo que esta tupla y la siguiente, son la misma:

{`(Calle:Enmedio,8), (Fax:964 201 340), (Población:Castellón),`
`(Onum:05), (Teléfono:964 201 240), (Area:Centro)`}

Gráficamente se suelen representar las relaciones mediante tablas. Los nombres de las columnas corresponden a los nombres de los atributos y las filas son cada una de las tuplas de la relación. Los valores que aparecen en cada una de las columnas pertenecen al conjunto de valores del dominio sobre el que está definido el atributo correspondiente.

4.3.2 Propiedades de las relaciones

Las relaciones tienen las siguientes características:

- Cada relación tiene un nombre y éste es distinto del nombre de todas las demás.
- Los valores de los atributos son atómicos: en cada tupla, cada atributo toma un solo valor. Se dice que las relaciones están *normalizadas*.
- No hay dos atributos que se llamen igual.
- El orden de los atributos no importa: los atributos no están ordenados.
- Cada tupla es distinta de las demás: no hay tuplas duplicadas.
- El orden de las tuplas no importa: las tuplas no están ordenadas.

4.3.3 Tipos de relaciones

En un SGBD relacional pueden existir varios tipos de relaciones, aunque no todos manejan todos los tipos.

- *Relaciones base.* Son relaciones reales que tienen nombre y forman parte directa de la base de datos almacenada (son autónomas).
- *Vistas.* También denominadas relaciones virtuales, son relaciones con nombre y derivadas: se representan mediante su definición en términos de otras relaciones con nombre, no poseen datos almacenados propios.
- *Instantáneas.* Son relaciones con nombre y derivadas. Pero a diferencia de las vistas, son reales, no virtuales: están representadas no sólo por su definición en términos de otras relaciones con nombre, sino también por sus propios datos almacenados. Son relaciones de sólo de lectura y se refrescan periódicamente.
- *Resultados de consultas.* Son las relaciones resultantes de alguna consulta especificada. Pueden o no tener nombre y no persisten en la base de datos.

- *Resultados intermedios.* Son las relaciones que contienen los resultados de las subconsultas. Normalmente no tienen nombre y tampoco persisten en la base de datos.
- *Resultados temporales.* Son relaciones con nombre, similares a las relaciones base o a las instantáneas, pero la diferencia es que se destruyen automáticamente en algún momento apropiado.

4.3.4 Claves

Ya que en una relación no hay tuplas repetidas, éstas se pueden distinguir unas de otras, es decir, se pueden identificar de modo único. La forma de identificarlas es mediante los valores de sus atributos.

Una *superclave* es un atributo o un conjunto de atributos que identifican de modo único las tuplas de una relación.

Una *clave candidata* es una superclave en la que ninguno de sus subconjuntos es una superclave de la relación. El atributo o conjunto de atributos K de la relación R es una clave candidata para R si y sólo si satisface las siguientes propiedades:

- *Unicidad:* nunca hay dos tuplas en la relación R con el mismo valor de K.
- *Irreducibilidad (minimalidad):* ningún subconjunto de K tiene la propiedad de unicidad, es decir, no se pueden eliminar componentes de K sin destruir la unicidad.

Cuando una clave candidata está formada por más de un atributo, se dice que es una *clave compuesta*. Una relación puede tener varias claves candidatas. Por ejemplo, en la relación `OFICINA`, el atributo `Población` no es una clave candidata ya que puede haber varias oficinas en una misma población. Sin embargo, ya que la empresa asigna un código único a cada oficina, el atributo `Onum` sí es una clave candidata de la relación `OFICINA`. También son claves candidatas de esta relación los atributos `Teléfono` y `Fax`.

En la base de datos de la inmobiliaria hay una relación denominada `VISITA` que contiene información sobre las visitas que los clientes han realizado a los inmuebles. Esta relación contiene el número del cliente `Qnum`, el número del inmueble `Inum`, la fecha de la visita `Fecha`

y un comentario opcional. Para un determinado número de cliente `Qnum`, se pueden encontrar varias visitas a varios inmuebles. Del mismo modo, dado un número de inmueble `Inum`, puede que haya varios clientes que lo hayan visitado. Por lo tanto, el atributo `Qnum` no es una clave candidata para la relación `VISITA`, como tampoco lo es el atributo `Inum`. Sin embargo, la combinación de los dos atributos sí identifica a una sola tupla, por lo que los dos juntos son una clave candidata de `VISITA`. Si se desea considerar la posibilidad de que un mismo cliente pueda visitar un mismo inmueble en varias ocasiones, habría que incluir el atributo `Fecha` para identificar las tuplas de modo único (aunque éste no es el caso de la empresa que nos ocupa).

Para identificar las claves candidatas de una relación no hay que fijarse en un estado o instancia de la base de datos. El hecho de que en un momento dado no haya duplicados para un atributo o conjunto de atributos, no garantiza que los duplicados no sean posibles. Sin embargo, la presencia de duplicados en un estado de la base de datos sí es útil para demostrar que cierta combinación de atributos no es una clave candidata. El único modo de identificar las claves candidatas es conociendo el significado real de los atributos, ya que esto permite saber si es posible que aparezcan duplicados. Sólo usando esta información semántica se puede saber con certeza si un conjunto de atributos forman una clave candidata. Por ejemplo, viendo la instancia anterior de la relación `PLANTILLA` se podría pensar que el atributo `Apellido` es una clave candidata. Pero ya que este atributo es el apellido de un empleado y es posible que haya dos empleados con el mismo apellido, el atributo no es una clave candidata.

La *clave primaria* de un relación es aquella clave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria. En el peor caso, la clave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función.

Las claves candidatas que no son escogidas como clave primaria son denominadas *claves alternativas*. Por ejemplo, la clave primaria de la relación `OFICINA` es el atributo `Onum`, siendo `Teléfono` y `Fax` dos claves alternativas. En la relación `VISITA` sólo hay una clave candidata formada por los atributos `Qnum` e `Inum`, por lo que esta clave candidata es la clave primaria.

Una *clave ajena* es un atributo o un conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (puede ser la misma). Las claves ajenas representan *relaciones entre datos*. El atributo `Onum` de `PLANTILLA` relaciona a cada empleado con la oficina a la que pertenece. Este atributo es una clave ajena cuyos valores hacen referencia al atributo `Onum`, clave primaria de `OFICINA`. Se dice que un valor de clave ajena representa una *referencia* a la tupla que contiene el mismo valor en su clave primaria (*tupla referenciada*).

4.3.5 Esquema de una base de datos relacional

Una base de datos relacional es un conjunto de relaciones normalizadas. Para representar el esquema de una base de datos relacional se debe dar el nombre de sus relaciones, los atributos de éstas, los dominios sobre los que se definen estos atributos, las claves primarias y las claves ajenas.

El esquema de la base de datos de la empresa inmobiliaria es el siguiente:

OFICINA	(Onum, Calle, Area, Población, Teléfono, Fax)
PLANTILLA	(Enum, Nombre, Apellido, Dirección, Teléfono, Puesto, Fecha_nac, Salario, DNI, Onum)
INMUEBLE	(Inum, Calle, Area, Población, Tipo, Hab, Alquiler, Pnum, Enum, Onum)
INQUILINO	(Qnum, Nombre, Apellido, Dirección, Teléfono, Tipo_pref, Alquiler_max)
PROPIETARIO	(Pnum, Nombre, Apellido, Dirección, Teléfono)
VISITA	(Qnum, Inum, Fecha, Comentario)

En el esquema, los nombres de las relaciones aparecen seguidos de los nombres de los atributos encerrados entre paréntesis. Las claves primarias son los atributos subrayados. Las claves ajenas se representan mediante los siguientes *diagramas referenciales*.

PLANTILLA	$\xrightarrow{\text{Onum}}$	OFICINA	:	*Oficina a la que pertenece el empleado.*
INMUEBLE	$\xrightarrow{\text{Pnum}}$	PROPIETARIO	:	*Propietario del inmueble.*
INMUEBLE	$\xrightarrow{\text{Enum}}$	PLANTILLA	:	*Empleado encargado del inmueble.*
INMUEBLE	$\xrightarrow{\text{Onum}}$	OFICINA	:	*Oficina a la que pertenece el inmueble.*
VISITA	$\xrightarrow{\text{Qnum}}$	INQUILINO	:	*Inquilino que ha visitado el inmueble.*
VISITA	$\xrightarrow{\text{Inum}}$	INMUEBLE	:	*Inmueble que ha sido visitado.*

A continuación se muestra un estado (instancia) de la base de datos cuyo esquema se acaba de definir.

OFICINA

Onum	Calle	Area	Población	Teléfono	Fax
05	Enmedio, 8	Centro	Castellón	964 201 240	964 201 340
07	Moyano, s/n	Centro	Castellón	964 215 760	964 215 670
03	San Miguel, 1		Villarreal	964 520 250	964 520 255
04	Trafalgar, 23	Grao	Castellón	964 284 440	964 284 420
02	Cedre, 26		Villarreal	964 525 810	964 252 811

PLANTILLA

Enum	Nombre	Apellido	Dirección	Teléfono	Puesto	Fecha_nac	Salario	DNI	Onum
EL21	Amelia	Pastor	Magallanes, 15 Castellón	964 284 560	Director	12/10/62	30000	39432212E	05
EG37	Pedro	Cubedo	Bayarri, 11 Villarreal	964 535 690	Supervisor	24/3/57	18000	38766623X	03
EG14	Luis	Collado	Borriol, 35 Villarreal	964 522 230	Administ.	9/5/70	12000	24391223L	03
EA9	Rita	Renau	Casalduch, 32 Castellón	964 257 550	Supervisor	19/5/60	18000	39233190F	07
EG5	Julio	Prats	Melilla, 23 Villarreal	964 524 590	Director	19/12/50	24000	25644309X	03
EL41	Carlos	Baeza	Herrero, 51 Castellón	964 247 250	Supervisor	29/2/67	18000	39552133T	05

INMUEBLE

Inum	Calle	Area	Población	Tipo	Hab	Alquiler	Pnum
IA14	Enmedio, 128	Centro	Castellón	Casa	6	600	P46
IL94	Riu Ebre, 24	Ronda Sur	Castellón	Piso	4	350	P87
IG4	Sorell, 5	Grao	Castellón	Piso	3	300	P40
IG36	Alicante,1		Segorbe	Casa	3	325	P93
IG21	San Francisco, 10		Vinaroz	Piso	5	550	P87
IG16	Capuchinos, 19	Rafalafena	Castellón	Piso	4	400	P93

PROPIETARIO

Pnum	Nombre	Apellido	Dirección	Teléfono
P46	Amparo	Felip	Asensi 24, Castellón	964 230 680
P87	Manuel	Obiol	Av. Libertad 15, Vinaroz	964 450 760
P40	Alberto	Estrada	Av. del Puerto 52, Castellón	964 200 740
P93	Yolanda	Robles	Purísima 4, Segorbe	964 710 430

INQUILINO

Qnum	Nombre	Apellido	Dirección	Teléfono	Tipo	Alquiler
Q76	Juan	Felip	Barceló 47, Castellón	964 282 540	Piso	375
Q56	Ana	Grangel	San Rafael 45, Almazora	964 551 110	Piso	300
Q74	Elena	Abaso	Navarra 76, Castellón	964 205 560	Casa	700
Q62	Alicia	Mori	Alloza 45, Castellón	964 229 580	Piso	550

VISITA

Qnum	Inum	Fecha	Comentario
Q56	IA14	24/11/99	muy pequeño
Q76	IG4	20/10/99	muy lejos
Q56	IG4	26/11/99	
Q62	IA14	14/11/99	no tiene salón
Q56	IG36	28/10/99	

4.4 Reglas de integridad

Una vez definida la estructura de datos del modelo relacional, pasamos a estudiar las reglas de integridad que los datos almacenados en dicha estructura deben cumplir para garantizar que son correctos.

Al definir cada atributo sobre un dominio se impone una restricción sobre el conjunto de valores permitidos para cada atributo. A este tipo de restricciones se les denomina *restricciones de dominios*. Hay además dos reglas de integridad muy importantes que son restricciones que se deben cumplir en todas las bases de datos relacionales y en todos sus estados o instancias (las reglas se deben cumplir todo el tiempo). Estas reglas son la *regla de integridad de entidades* y la *regla de integridad referencial*. Antes de definirlas, es preciso conocer el concepto de *nulo*.

4.4.1 Nulos

Cuando en una tupla un atributo es desconocido, se dice que es *nulo*. Un nulo no representa el valor cero ni la cadena vacía, éstos son valores que tienen significado. El nulo implica ausencia de información, bien porque al insertar la tupla se desconocía el valor del atributo, o bien porque para dicha tupla el atributo no tiene sentido.

Ya que los nulos no son valores, deben tratarse de modo diferente, lo que causa problemas de implementación. De hecho, no todos los SGBD relacionales soportan los nulos.

4.4.2 Regla de integridad de entidades

La primera regla de integridad se aplica a las claves primarias de las relaciones base: *ninguno de los atributos que componen la clave primaria puede ser nulo.*

Por definición, una clave primaria es un identificador irreducible que se utiliza para identificar de modo único las tuplas. Que es irreducible significa que ningún subconjunto de la clave primaria sirve para identificar las tuplas de modo único. Si se permite que parte de la clave primaria sea nula, se está diciendo que no todos sus atributos son necesarios para distinguir las tuplas, con lo que se contradice la irreducibilidad.

Nótese que esta regla sólo se aplica a las relaciones base y a las claves primarias, no a las claves alternativas.

4.4.3 Regla de integridad referencial

La segunda regla de integridad se aplica a las claves ajenas: *si en una relación hay alguna clave ajena, sus valores deben coincidir con valores de la clave primaria a la que hace referencia, o bien, deben ser completamente nulos.*

La regla de integridad referencial se enmarca en términos de estados de la base de datos: indica lo que es un estado ilegal, pero no dice cómo puede evitarse. La cuestión es ¿qué hacer si estando en un estado legal, llega una petición para realizar una operación que conduce a un estado ilegal? Existen dos opciones: *rechazar* la operación, o bien *aceptar* la operación y realizar operaciones adicionales compensatorias que conduzcan a un estado legal.

Por lo tanto, para cada clave ajena de la base de datos habrá que contestar a tres preguntas:

- *Regla de los nulos:* ¿Tiene sentido que la clave ajena acepte nulos?
- *Regla de borrado:* ¿Qué ocurre si se intenta borrar la tupla referenciada por la clave ajena?
  - *Restringir:* no se permite borrar la tupla referenciada.
  - *Propagar:* se borra la tupla referenciada y se propaga el borrado a las tuplas que la referencian mediante la clave ajena.
  - *Anular:* se borra la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (sólo si acepta nulos).
- *Regla de modificación:* ¿Qué ocurre si se intenta modificar el valor de la clave primaria de la tupla referenciada por la clave ajena?
  - *Restringir:* no se permite modificar el valor de la clave primaria de la tupla referenciada.
  - *Propagar:* se modifica el valor de la clave primaria de la tupla referenciada y se propaga la modificación a las tuplas que la referencian mediante la clave ajena.
  - *Anular:* se modifica la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (sólo si acepta nulos).

4.4.4 Reglas de negocio

Además de las dos reglas de integridad anteriores, los usuarios o los administradores de la base de datos pueden imponer ciertas restricciones específicas sobre los datos, denominadas *reglas de negocio*.

Por ejemplo, si en una oficina de la empresa inmobiliaria sólo puede haber hasta veinte empleados, el SGBD debe dar la posibilidad al usuario de definir una regla al respecto y debe hacerla respetar. En este caso, no debería permitir dar de alta un empleado en una oficina que ya tiene los veinte permitidos.

Hoy en día aún existen SGBD relacionales que no permiten definir este tipo de restricciones ni las hacen respetar.

4.5 Lenguajes relacionales

La tercera parte de un modelo de datos es la de la manipulación. Son varios los lenguajes utilizados por los SGBD relacionales para manejar las relaciones. Algunos de ellos son *procedurales*, lo que quiere decir que el usuario dice al sistema exactamente cómo debe manipular los datos. Otros son *no procedurales*, que significa que el usuario dice qué datos necesita, en lugar de decir cómo deben obtenerse.

En este apartado se presentan el álgebra relacional y el cálculo relacional, definidos por Codd como la base de los lenguajes relacionales. Se puede decir que el álgebra es un lenguaje procedural (de alto nivel), mientras que el cálculo relacional es un lenguaje no procedural. Sin embargo, ambos lenguajes son equivalentes: para cada expresión del álgebra, se puede encontrar una expresión equivalente en el cálculo, y viceversa.

El álgebra relacional (o el cálculo relacional) se utilizan para medir la potencia de los lenguajes relacionales. Si un lenguaje permite obtener cualquier relación que se pueda derivar mediante el álgebra relacional, se dice que es *relacionalmente completo*. La mayoría de los lenguajes relacionales son relacionalmente completos, pero tienen más potencia que el álgebra o el cálculo porque se les han añadido operadores especiales.

Tanto el álgebra como el cálculo son lenguajes formales no muy "amigables". Pero se deben estudiar porque sirven para ilustrar las operaciones básicas que todo lenguaje de manejo datos debe ofrecer. Además, han sido la base para otros lenguajes relacionales de manejo de datos de más alto nivel.

4.5.1 Álgebra relacional

El álgebra relacional es un lenguaje formal con una serie de operadores que trabajan sobre una o varias relaciones para obtener otra relación resultado, sin que cambien las relaciones originales. Tanto los operandos como los resultados son relaciones, por lo que la salida de una operación puede ser la entrada de otra operación. Esto permite anidar expresiones del álgebra, del mismo modo que se pueden anidar las expresiones aritméticas. A esta propiedad se le denomina *clausura*: las relaciones son cerradas bajo el álgebra, del mismo modo que los números son cerrados bajo las operaciones aritméticas.

En este apartado se presentan los operadores del álgebra relacional de un modo informal. Las definiciones formales pueden encontrarse en la bibliografía que se comenta al final del capítulo. Primero se describen los ocho operadores originalmente propuestos por Codd y después se estudian algunos operadores adicionales que añaden potencia al lenguaje.

De los ocho operadores, sólo hay cinco que son fundamentales: *restricción*, *proyección*, *producto cartesiano*, *unión* y *diferencia*, que permiten realizar la mayoría de las operaciones de obtención de datos. Los operadores no fundamentales son la *concatenación (join)*, la *intersección* y la *división*, que se pueden expresar a partir de los cinco operadores fundamentales.

La restricción y la proyección son operaciones *unarias* porque operan sobre una sola relación. El resto de las operaciones son *binarias* porque trabajan sobre pares de relaciones. En las definiciones que se presentan a continuación, se supone que `R` y `S` son dos relaciones cuyos atributos son `A=(`a_1`, `a_2`, ..., `a_N`)` y `B=(`b_1`, `b_2`, ..., `b_M`)` respectivamente.

Restricción : `R WHERE condición`

La restricción, también denominada *selección*, opera sobre una sola relación `R` y da como resultado otra relación cuyas tuplas son las tuplas de `R` que satisfacen la condición

especificada. Esta condición es una comparación en la que aparece al menos un atributo de R, o una combinación booleana de varias de estas comparaciones.

Ejemplo 4.1 *Obtener todos los empleados con un salario anual superior a 15.000 euros.*

```
PLANTILLA WHERE salario>15000
```

Enum	Nombre	Apellido	Dirección	Teléfono	Puesto	Fecha_nac	Salario	DNI	Onum
EL21	Amelia	Pastor	Magallanes, 15 Castellón	964 284 560	Director	12/10/62	30000	39432212E	05
EG37	Pedro	Cubedo	Bayarri, 11 Villarreal	964 535 690	Supervisor	24/3/57	18000	38766623X	03
EA9	Rita	Renau	Casalduch, 32 Castellón	964 257 550	Supervisor	19/5/60	18000	39233190F	07
EG5	Julio	Prats	Melilla, 23 Villarreal	964 524 590	Director	19/12/50	24000	25644309X	03
EL41	Carlos	Baeza	Herrero, 51 Castellón	964 247 250	Supervisor	29/2/67	18000	39552133T	05

Ejemplo 4.2 *Obtener todos los inmuebles de Castellón con un alquiler mensual de hasta 350 euros.*

```
INMUEBLE WHERE población='Castellón' AND alquiler<=350
```

Inum	Calle	Area	Población	Tipo	Hab	Alquiler	Pnum
IL94	Riu Ebre, 24	Ronda Sur	Castellón	Piso	4	350	P87
IG4	Sorell, 5	Grao	Castellón	Piso	3	300	P40
IG36	Alicante,1		Segorbe	Piso	3	325	P93

Proyección : R[a_i, ..., a_k]

La proyección opera sobre una sola relación R y da como resultado otra relación que contiene un subconjunto vertical de R, extrayendo los valores de los atributos especificados y eliminando duplicados.

Ejemplo 4.3 *Obtener un listado de empleados mostrando su número, nombre, apellido y salario.*

```
PLANTILLA [enum,nombre,apellido,salario]
```

Enum	Nombre	Apellido	Salario
EL21	Amelia	Pastor	30000
EG37	Pedro	Cubedo	18000
EG14	Luis	Collado	12000
EA9	Rita	Renau	18000
EG5	Julio	Prats	24000
EL41	Carlos	Baeza	18000

Ejemplo 4.4 *Obtener los distintos puestos que pueden ocupar los empleados.*

```
PLANTILLA [puesto]
```

Puesto
Director
Supervisor
Administ.

Producto cartesiano : `R TIMES S`

El producto cartesiano obtiene una relación cuyas tuplas están formadas por la concatenación de todas las tuplas de `R` con todas las tuplas de `S`.

La restricción y la proyección son operaciones que permiten extraer información de una sola relación. Habrá casos en que sea necesario combinar la información de varias relaciones. El producto cartesiano "multiplica" dos relaciones, definiendo una nueva relación que tiene todos los pares posibles de tuplas de las dos relaciones. Si la relación `R` tiene P tuplas y N atributos y la relación `S` tiene Q tuplas y M atributos, la relación resultado tendrá $P * Q$ tuplas y $N + M$ atributos. Ya que es posible que haya atributos con el mismo nombre en

las dos relaciones, el nombre de la relación se antepondrá al del atributo en este caso para que los nombres de los atributos sigan siendo únicos en la relación resultado.

Ejemplo 4.5 *Obtener los nombres de los inquilinos y los comentarios que éstos han realizado cuando han visto algún inmueble.*

```
INQUILINO[qnum,nombre,apellido] TIMES VISITA[qnum,inum,comentario]
```

INQUILINO.Qnum	Nombre	Apellido	VISITA.Qnum	Inum	Comentario
Q76	Juan	Felip	Q56	IA14	muy pequeño
Q76	Juan	Felip	Q76	IG4	muy lejos
Q76	Juan	Felip	Q56	IG4	
Q76	Juan	Felip	Q62	IA14	no tiene salón
Q76	Juan	Felip	Q56	IG36	
Q56	Ana	Grangel	Q56	IA14	muy pequeño
Q56	Ana	Grangel	Q76	IG4	muy lejos
Q56	Ana	Grangel	Q56	IG4	
Q56	Ana	Grangel	Q62	IA14	no tiene salón
Q56	Ana	Grangel	Q56	IG36	
Q74	Elena	Abaso	Q56	IA14	muy pequeño
Q74	Elena	Abaso	Q76	IG4	muy lejos
Q74	Elena	Abaso	Q56	IG4	
Q74	Elena	Abaso	Q62	IA14	no tiene salón
Q74	Elena	Abaso	Q56	IG36	
Q62	Alicia	Mori	Q56	IA14	muy pequeño
Q62	Alicia	Mori	Q76	IG4	muy lejos
Q62	Alicia	Mori	Q56	IG4	
Q62	Alicia	Mori	Q62	IA14	no tiene salón
Q62	Alicia	Mori	Q56	IG36	

Como se puede observar, la relación resultado contiene más información de la que se necesita. Por ejemplo, la primera tupla tiene distintos números de inquilino: el comentario realizado en la visita no corresponde al inquilino cuyo nombre y apellido se muestra. Para obtener el listado que se pide en el ejemplo, es necesario realizar una restricción para quedarse solamente con las tuplas en donde `INQUILINO.Qnum = VISITA.Qnum`.

```
(INQUILINO[qnum,nombre,apellido] TIMES VISITA[qnum,inum,comentario])
WHERE inquilino.qnum=visita.qnum
```

El resultado de esta operación se muestra a continuación.

INQUILINO.Qnum	Nombre	Apellido	VISITA.Qnum	Inum	Comentario
Q76	Juan	Felip	Q76	IG4	muy lejos
Q56	Ana	Grangel	Q56	IA14	muy pequeño
Q56	Ana	Grangel	Q56	IG4	
Q56	Ana	Grangel	Q56	IG36	
Q62	Alicia	Mori	Q62	IA14	no tiene salón

La combinación del producto cartesiano y la restricción del modo en que se acaba de realizar, se puede reducir a la operación de *concatenación* (*join*) que se presenta más adelante.

Unión : `R UNION S`

La unión de dos relaciones `R` y `S`, con P y Q tuplas respectivamente, es otra relación que tiene como mucho $P + Q$ tuplas siendo éstas las tuplas que se encuentran en `R` o en `S` o en ambas relaciones a la vez. Para poder realizar esta operación, `R` y `S` deben ser compatibles para la unión.

Se dice que dos relaciones son *compatibles para la unión* si ambas tienen la misma cabecera, es decir, si tienen el mismo número de atributos y éstos se encuentran definidos sobre los mismos dominios. En muchas ocasiones será necesario realizar proyecciones para hacer que dos relaciones sean compatibles para la unión.

Ejemplo 4.6 *Obtener un listado de las áreas en las que hay oficinas o inmuebles para alquilar.*

```
OFICINA[área] UNION INMUEBLE[área]
```

Area
Centro
Grao
Ronda Sur
Rafalafena

Diferencia : `R MINUS S`

La diferencia obtiene una relación que tiene las tuplas que se encuentran en `R` y no se encuentran en `S`. Para realizar esta operación, `R` y `S` deben ser compatibles para la unión.

Ejemplo 4.7 *Obtener un listado de todas las poblaciones en donde hay una oficina y no hay inmuebles para alquilar.*

```
OFICINA[población] MINUS INMUEBLE[población]
```

Población
Villarreal

Concatenación (Join) : `R JOIN S`

La concatenación de dos relaciones `R` y `S` obtiene como resultado una relación cuyas tuplas son todas las tuplas de `R` concatenadas con todas las tuplas de `S` que en los atributos comunes (que se llaman igual) tienen los mismos valores. Estos atributos comunes aparecen una sola vez en el resultado.

Ejemplo 4.8 *Obtener los nombres y los comentarios que los inquilinos han realizado cuando han visto algún inmueble.*

```
INQUILINO JOIN VISITA
```

Esta expresión obtiene el mismo resultado que la expresión final del ejemplo 4.5, ya que la concatenación es, en realidad, un producto cartesiano y una restricción de igualdad sobre los atributos comunes.

Concatenación externa (Outer–join) : `R JOIN S (+)`

La concatenación externa es una concatenación en la que las tuplas de `R` que no tienen valores en común con ninguna tupla de `S`, también aparecen en el resultado.

Ejemplo 4.9 *Obtener un listado de todos los inmuebles y las visitas que han tenido.*

`INMUEBLE JOIN VISITA (+)`

Inum	Calle	Población	Qnum	Fecha	Comentario
IA14	Enmedio, 128	Castellón	Q56	24/11/99	muy pequeño
IA14	Enmedio, 128	Castellón	Q62	14/11/99	no tiene salón
IL94	Riu Ebre, 24	Castellón			
IG4	Sorell, 5	Castellón	Q76	20/10/99	muy lejos
IG4	Sorell, 5	Castellón	Q56	26/11/99	
IG36	Alicante,1	Segorbe	Q56	28/10/99	
IG21	San Francisco, 10	Vinaroz			
IG16	Capuchinos, 19	Castellón			

La expresión `S (+) JOIN R` es equivalente a `R JOIN S (+)`. Cuando en ambas relaciones hay tuplas que no se pueden concatenar y se desea que en el resultado aparezcan también todas estas tuplas (tanto las de una relación como las de la otra), se utiliza la *concatenación externa completa*: `R (+) JOIN S (+)`

Intersección : `R INTERSECT S`

La intersección obtiene como resultado una relación que contiene las tuplas de `R` que también se encuentran en `S`. Para realizar esta operación, `R` y `S` deben ser compatibles para la unión.

La intersección se puede expresar en términos de diferencias:

$$\texttt{R INTERSECT S} = \texttt{R MINUS (R MINUS S)}$$

División : `R DIVIDEBY S`

Suponiendo que la cabecera de `R` es el conjunto de atributos `A` y que la cabecera de `S`

es el conjunto de atributos `B`, tales que `B` es un subconjunto de `A`, y si `C` = `A - B` (los atributos de `R` que no están en `S`), la división obtiene una relación cuya cabecera es el conjunto de atributos `C` y que contiene las tuplas de `R` que están acompañadas de todas las tuplas de `S`.

Ejemplo 4.10 *Obtener los inquilinos que han visitado todos los inmuebles de tres habitaciones.*

```
VISITA[qnum,inum] DIVIDEBY (INMUEBLE WHERE hab=3)[inum]
```

Qnum
Q56

Además de las operaciones que Codd incluyó en el álgebra relacional, otros autores han aportado otras operaciones para dar más potencia al lenguaje. Es de especial interés la *agrupación*, también denominada *resumen*, que añade capacidad computacional al álgebra.

Agrupación : `SUMMARIZE R GROUPBY(`a_i`,...,`a_k`) ADD cálculo AS atributo`

Esta operación agrupa las tuplas de `R` que tienen los mismos valores en los atributos especificados y realiza un cálculo sobre los grupos obtenidos. La relación resultado tiene como cabecera los atributos por los que se ha agrupado y el cálculo realizado, al que se da el nombre especificado en `atributo`.

Los cálculos que se pueden realizar sobre los grupos de filas son: suma de los valores de un atributo (`SUM(`a_p`)`), media de los valores de un atributo (`AVG(`a_p`)`), máximo y mínimo de los valores de un atributo (`MAX(`a_p`)`, `MIN(`a_p`)`) y número de tuplas en el grupo (`COUNT(*)`). La relación resultado tendrá tantas filas como grupos se hayan obtenido.

Ejemplo 4.11 *Obtener el salario total que se gasta en los empleados de cada oficina.*

```
SUMMARIZE PLANTILLA GROUPBY(oficina) ADD SUM(salario) AS salario_total
```

Oficina	Salario_total
05	48000
03	54000
07	18000

4.5.2 Cálculo relacional

El álgebra relacional y el cálculo relacional son formalismos diferentes que representan distintos estilos de expresión del manejo de datos en el ámbito del modelo relacional. El álgebra relacional proporciona una serie de operaciones que se pueden usar para decir al sistema cómo *construir* la relación deseada a partir de las relaciones de la base de datos. El cálculo relacional proporciona una notación para formular la *definición* de la relación deseada en términos de las relaciones de la base de datos.

El cálculo relacional toma su nombre del *cálculo de predicados*, que es una rama de la lógica. Hay dos tipos de cálculo relacional, el *orientado a tuplas*, propuesto por Codd, y el *orientado a dominios*, propuesto por otros autores. El estudio del cálculo relacional se hará mediante definiciones informales. Las definiciones formales se pueden encontrar en la bibliografía que se comenta al final del capítulo.

En el cálculo de predicados (lógica de primer orden), un *predicado* es una función con argumentos que se puede evaluar a verdadero o falso. Cuando los argumentos se sustituyen por valores, la función lleva a una expresión denominada *proposición*, que puede ser verdadera o falsa. Por ejemplo, las frases 'Carlos Baeza es un miembro de la plantilla' y 'Carlos Baeza gana más que Amelia Pastor' son proposiciones, ya que se puede determinar si son verdaderas o falsas. En el primer caso, la función 'es un miembro de la plantilla' tiene un argumento (Carlos Baeza) y en el segundo caso, la función 'gana más que' tiene dos argumentos (Carlos Baeza y Amelia Pastor).

Si un predicado tiene una variable, como en '`x` es un miembro de la plantilla', esta variable debe tener un *rango* asociado. Cuando la variable se sustituye por alguno de los valores de su rango, la proposición puede ser cierta; para otros valores puede ser falsa. Por ejemplo, si el rango de `x` es el conjunto de todas las personas y reemplazamos `x` por Carlos Baeza, la proposición 'Carlos Baeza es un miembro de la plantilla' es cierta. Pero si reemplazamos `x`

por el nombre de una persona que no es miembro de la plantilla, la proposición es falsa.

Si `F` es un predicado, la siguiente expresión corresponde al conjunto de todos los valores de `x` para los que `F` es cierto:

```
x WHERE F(x)
```

Los predicados se pueden conectar mediante AND, OR y NOT para formar *predicados compuestos.*

Cálculo orientado a tuplas

En el cálculo relacional orientado a tuplas, lo que interesa es encontrar tuplas para las que se cumple cierto predicado. El cálculo orientado a tuplas se basa en el uso de *variables tupla.* Una variable tupla es una variable cuyo rango de valores son las tuplas de una relación.

Por ejemplo, para especificar el rango de la variable tupla `PX` sobre la relación `PLANTILLA` se utiliza la siguiente expresión:

```
RANGE OF PX IS PLANTILLA
```

Para expresar la consulta 'obtener todas las tuplas `PX` para las que `F(PX)` es cierto', se escribe la siguiente expresión:

```
PX WHERE F(PX)
```

donde `F` es lo que se denomina una *fórmula bien formada (fbf).* Por ejemplo, para expresar la consulta 'obtener todos los datos de los empleados que ganan más de 10.000 euros' se puede escribir:

```
RANGE OF PX IS PLANTILLA
PX WHERE PX.salario > 10000
```

`PX.salario` se refiere al valor del atributo `salario` para la tupla `PX`. Para que se muestren solamente algunos atributos, por ejemplo, `apellido` y `salario`, en lugar de todos los atributos de la relación, se escribe:

```
RANGE OF PX IS PLANTILLA
PX.apellido, PX.salario WHERE PX.salario > 10000
```

Hay dos *cuantificadores* que se utilizan en las fórmulas bien formadas para decir a cuántas instancias se aplica el predicado. El *cuantificador existencial* $\exists$ ('existe') se utiliza en las fórmulas bien formadas que deben ser ciertas para al menos una instancia.

```
RANGE OF OX IS OFICINA
∃OX (OX.onum = PX.onum AND OX.población = 'Castellón')
```

Esta fórmula bien formada dice que 'existe una oficina que tiene el mismo número que el número de oficina de la tupla que ahora se encuentra en la variable de `PLANTILLA`, `PX`, y que está en Castellón'. El *cuantificador universal* $\forall$ ('para todo') se utiliza en las fórmulas bien formadas que deben ser ciertas para todas las instancias.

```
∀PX (PX.población ≠ 'Castellón')
```

Esta fórmula bien formada dice que 'para todas las tuplas de `PLANTILLA`, la población no es Castellón'. Utilizando las reglas de las operaciones lógicas, esta fórmula bien formada se puede escribir también del siguiente modo:

```
NOT ∃PX (PX.población = 'Castellón')
```

que dice que 'no hay ningún miembro de la plantilla cuya población sea Castellón'.

Las variables tupla que no están cuantificadas por $\forall$ o $\exists$ se denominan *variables libres*. Si están cuantificadas, se denominan *variables ligadas*. El cálculo, al igual que cualquier lenguaje, tiene una sintaxis que permite construir expresiones válidas. Para que una expresión no sea ambigua y tenga sentido, debe seguir esta sintaxis:

- Si P es una fórmula bien formada n–ária (un predicado con n argumentos) y $t_1, t_2, \ldots, t_n$ son constantes o variables, entonces $P(t_1, t_2, \ldots, t_n)$ es también una fórmula bien formada.
- Si t_1 y t_2 son constantes o variables del mismo dominio y θ es un operador de comparación $(<, <=, >, >=, =, \neq)$, entonces $t_1 \theta t_2$ es una fórmula bien formada.
- Si P_1 y P_2 son fórmulas bien formadas, también lo son su conjunción P_1 *AND* P_2, su disyunción P_1 *OR* P_2 y la negación *NOT* P_1. Además, si P es una fórmula bien formada que tiene una variable libre X, entonces $\exists X(P)$ y $\forall X(P)$ también son fórmulas bien formadas.

Ejemplo 4.12 *Obtener un listado de los empleados que llevan inmuebles de Almazora.*

```
RANGE OF PX IS PLANTILLA
RANGE OF IX IS INMUEBLE
PX WHERE ∃IX (IX.enum = PX.enum AND IX.población = 'Almazora')
```

Esta petición se puede escribir en términos del cálculo: 'un miembro de la plantilla debe salir en el listado si existe una tupla en `INMUEBLE` que tenga asignado a ese empleado y que esté en Almazora (`población`)'. Nótese que formulando la consulta de este modo no se indica la estrategia a seguir para ejecutarla, por lo que el SGBD tiene libertad para decidir qué operaciones hacer y en qué orden. En el álgebra relacional se hubiera formulado así: 'Hacer una restricción sobre `INMUEBLE` para quedarse con las tuplas que tienen como `población` Almazora, y hacer después una concatenación con `PLANTILLA`.

Ejemplo 4.13 *Obtener las oficinas cuyos empleados (todos) nacieron de 1965 en adelante.*

```
RANGE OF PX IS PLANTILLA
RANGE OF OX IS OFICINA
OX WHERE ∀PX (PX.onum ≠OX.onum OR PX.fecha_nac >= '1/1/65')
```

La expresión anterior es equivalente a esta otra:

```
OX WHERE NOT ∃PX (PX.onum = OX.onum AND PX.fecha_nac < '1/1/65')
```

Cálculo orientado a dominios

En el cálculo relacional orientado a dominios las variables toman sus valores en dominios, en lugar de tomar valores de tuplas de relaciones. Otra diferencia con el cálculo orientado a tuplas es que en el cálculo orientado a dominios hay un tipo de comparación adicional, a la que se denomina *ser miembro de*. Esta condición tiene la forma:

$$\texttt{R(}a_1\texttt{:}v_1\texttt{, }a_2\texttt{:}v_2\texttt{, ...)}$$

donde los a_i son atributos de la relación R y los v_i son variables dominio o constantes. La condición se evalúa a verdadero si existe alguna tupla en R que tiene los valores especificados en los atributos especificados. Por ejemplo, la siguiente condición:

```
PLANTILLA(puesto:'Supervisor', onum:'O3')
```

se evaluará a verdadero si hay algún empleado que sea supervisor en la oficina O3. Y la condición

```
PLANTILLA(puesto:px, onum:ox)
```

será cierta si hay alguna tupla en PLANTILLA que tenga en puesto el valor actual de la variable dominio px y que tenga en onum el valor actual de la variable dominio ox.

Ejemplo 4.14 *Obtener los apellidos de los empleados que no siendo directores, tienen un salario mayor de 10.000 euros.*

```
ax WHERE ∃px ∃sx (px ≠ 'Director' AND sx > 10000
AND PLANTILLA(apellido:ax, puesto:px, salario:sx))
```

4.5.3 Otros lenguajes

Aunque el cálculo relacional es difícil de entender y de usar, tiene una propiedad muy atractiva: es un lenguaje no procedural. Esto ha hecho que se busquen técnicas no procedurales algo más sencillas, resultando en dos nuevas categorías de lenguajes relacionales: orientados a transformaciones y gráficos.

Los *lenguajes orientados a transformaciones* son lenguajes no procedurales que utilizan relaciones para transformar los datos de entrada en la salida deseada. Estos lenguajes tienen estructuras que son fáciles de utilizar y que permiten expresar lo que se desea en términos de lo que se conoce. Uno de estos lenguajes es SQL (Structured Query Language).

Los *lenguajes gráficos* visualizan en pantalla una fila vacía de cada una de las tablas que indica el usuario. El usuario rellena estas filas con un 'ejemplo' de lo que desea y el sistema devuelve los datos que siguen tal ejemplo. Uno de estos lenguajes es QBE (Query–by–Example).

Otra categoría son los *lenguajes de cuarta generación (4GL)*, que permiten diseñar una aplicación a medida utilizando un conjunto limitado de órdenes en un entorno amigable (normalmente un entorno de menús). Algunos sistemas aceptan cierto lenguaje natural, una versión restringida del idioma inglés, al que algunos llaman *lenguaje de quinta generación (5GL)*, aunque todavía se encuentra en desarrollo.

4.6 Vistas

En la arquitectura de tres niveles estudiada se describe una vista externa como la estructura de la base de datos tal y como la ve un usuario en particular. En el modelo relacional, el término 'vista' tiene un significado un tanto diferente. En lugar de ser todo el esquema externo de un usuario, una vista es una *relación virtual*, una relación que en realidad no existe como tal. Una vista se puede construir realizando operaciones como las del álgebra relacional: restricciones, proyecciones, concatenaciones, etc. a partir de las relaciones base de la base de datos. Las relaciones base son aquellas que forman parte directa de la base de datos, las que se encuentran almacenadas físicamente. Un esquema externo puede tener

tanto relaciones base como vistas derivadas de las relaciones base de la base de datos.

Una vista es el resultado dinámico de una o varias operaciones relacionales realizadas sobre las relaciones base. Una vista es una relación virtual que se produce cuando un usuario la consulta. Al usuario le parece que la vista es una relación que existe y la puede manipular como si se tratara de una relación base, pero la vista no está almacenada físicamente. El contenido de una vista está definido como una consulta sobre una o varias relaciones base. Cualquier operación que se realice sobre la vista se traduce automáticamente a operaciones sobre las relaciones de las que se deriva. Las vistas son *dinámicas* porque los cambios que se realizan sobre las tablas base que afectan a una vista se reflejan inmediatamente sobre ella. Cuando un usuario realiza un cambio sobre la vista (no todo tipo de cambios están permitidos), este cambio se realiza sobre las relaciones de las que se deriva.

Las vistas son útiles por varias razones:

- Proporcionan un poderoso mecanismo de seguridad, ocultando partes de la base de datos a ciertos usuarios. El usuario no sabrá que existen aquellos atributos que se han omitido al definir una vista.
- Permiten que los usuarios accedan a los datos en el formato que ellos desean o necesitan, de modo que los mismos datos pueden ser vistos con formatos distintos por distintos usuarios.
- Se pueden simplificar operaciones sobre las relaciones base que son complejas. Por ejemplo, se puede definir una vista como la concatenación de dos relaciones. El usuario puede hacer restricciones y proyecciones sobre la vista, que el SGBD traducirá en las operaciones equivalentes sobre la concatenación.

Se puede utilizar una vista para ofrecer un esquema externo a un usuario de modo que éste lo encuentre 'familiar'. Por ejemplo:

- Un usuario puede necesitar los datos de los directores junto con los de las oficinas. Esta vista se crea haciendo una concatenación de las relaciones `PLANTILLA` y `OFICINA`, y proyectando sobre los atributos que se desee mantener.

- Otro usuario puede que necesite ver los datos de los empleados sin ver el salario. Para este usuario se realiza una proyección para crear una vista sin el atributo `salario`.
- Los atributos se pueden renombrar, de modo que cada usuario los vea del modo en que esté acostumbrado. También se puede cambiar el orden en que se visualizan las columnas.
- Un miembro de la plantilla puede querer ver sólo los datos de aquellos inmuebles que tiene asignados. En este caso, se debe hacer una restricción para que sólo se vea el subconjunto horizontal deseado de la relación `INMUEBLE`.

Como se ve, las vistas proporcionan independencia de datos a nivel lógico, que también se da cuando se reorganiza el nivel conceptual. Si se añade un atributo a una relación, los usuarios no se percatan de su existencia si sus vistas no lo incluyen. Si una relación existente se reorganiza o se divide en varias relaciones, se pueden crear vistas para que los usuarios la sigan viendo como al principio.

Cuando se actualiza una relación base, el cambio se refleja automáticamente en todas las vistas que la referencian. Del mismo modo, si se actualiza una vista, las relaciones base de las que se deriva deberían reflejar el cambio. Sin embargo, hay algunas restricciones respecto a los tipos de modificaciones que se pueden realizar sobre las vistas. A continuación, se resumen las condiciones bajo las cuales la mayoría de los sistemas determinan si se permite realizar una actualización:

- Se permiten las actualizaciones de vistas que se definen mediante una consulta simple sobre una sola relación base y que contienen la clave primaria de la relación base.
- No se permiten las actualizaciones de vistas que se definen sobre varias relaciones base.
- No se permiten las actualizaciones de vistas definidas con operaciones de agrupamiento (`GROUPBY`).

4.7 Resumen

La relación es la estructura de datos del modelo relacional. Las relaciones se representan gráficamente como tablas, donde las filas corresponden a las tuplas y las columnas corresponden a los atributos. Los atributos se definen sobre dominios.

Las relaciones de una base de datos tienen una serie de propiedades: en la intersección de cada fila con cada columna hay un solo valor (valor atómico), los nombres de los atributos de una relación son todos distintos entre sí, los atributos no están ordenados, las tuplas no están ordenadas y no hay tuplas repetidas. El grado de una relación es el número de atributos y la cardinalidad es el número de tuplas.

Una superclave es un conjunto de atributos que identifica las tuplas de una relación de modo único. Una clave candidata es una superclave minimal o irreducible. La clave primaria es la clave candidata que se escoge para identificar las tuplas de una relación. Toda relación tiene siempre clave primaria. Una clave ajena es un atributo o un conjunto de atributos que hacen referencia a la clave primaria de otra relación.

Cuando un atributo no tiene valor para una determinada tupla, bien porque se desconoce o bien porque no tiene sentido para dicha tupla, se dice que es nulo.

La regla de integridad de entidades es una restricción que dice que ninguno de los atributos que forman la clave primaria puede ser nulo. La regla de integridad referencial dice que los valores de las claves ajenas deben coincidir con alguno de los valores de la clave primaria a la que hacen referencia, o bien ser completamente nulos.

Los lenguajes relacionales de manejo de datos se pueden clasificar como procedurales, no procedurales, orientados a transformaciones, gráficos, de cuarta generación o de quinta generación. El álgebra relacional es un lenguaje procedural formal. Sus operaciones son: restricción, proyección, producto cartesiano, unión, intersección, diferencia, división y varios tipos de concatenación (join). El cálculo relacional es un lenguaje no procedural formal que utiliza predicados. El álgebra relacional y el cálculo relacional son equivalentes.

Una vista es una relación virtual. Las vistas proporcionan seguridad y permiten que el diseñador haga esquemas a medida de cada usuario. Las vistas se generan dinámicamente y

no todas son actualizables.

Bibliografía

Este capítulo ha sido elaborado fundamentalmente a partir de los capítulos sobre el modelo relacional contenidos en los textos de Connolly, Begg y Strachan (1996) y Date (1993). Es en éste último en donde se encuentran todas las definiciones formales que en este capítulo se han omitido. La sintaxis del álgebra y del cálculo relacional se ha tomado de Date (1993), ya que se ha considerado que es la más sencilla y fácil de recordar.

Capítulo 5

Planificación, diseño y administración de bases de datos

En este capítulo se comentan las fases principales del ciclo de vida de un sistema de información y se ve cómo se relaciona esto con el desarrollo de aplicaciones de bases de datos. Después, se describen las tareas que se deben realizar en cada etapa del ciclo de vida de una aplicación de bases de datos para que se produzca un sistema que funcione correctamente. Por último, se presenta todo el personal responsable de la planificación, diseño y administración de una base de datos.

5.1 Introducción

Se inicia este capítulo con algunos párrafos de Hernández (1997), en donde se justifica la importancia del diseño de bases de datos:

"Algunas de las personas que trabajan con SGBD relacionales parecen preguntarse porqué deberían preocuparse del diseño de las bases de datos que utilizan. Después de todo, la mayoría de los SGBD vienen con bases de datos de ejemplo que se pueden copiar y después modificar, para que se adecuen a cada caso particular, e incluso las tablas de esas bases de datos de ejemplo se pueden cortar y pegar en una nueva base de datos. Algunos SGBD

tienen "asistentes", herramientas que guían al usuario a través del proceso de definición y creación de tablas. Sin embargo, esas herramientas no sirven para *diseñar* una base de datos, tan solo ayudan a crear las tablas físicas que se incluirán en la base de datos.

Lo que la mayoría de la gente no parece entender es que esas herramientas se deben utilizar *después de que se haya realizado el diseño lógico de la base de datos.* Los asistentes y las bases de datos de ejemplo se suministran para minimizar el tiempo que lleva *implementar* la estructura física de la base de datos. La idea es que si se ahorra tiempo en la implementación de la estructura de la base de datos una vez se ha realizado el diseño lógico, habrá más tiempo para centrarse en la creación y construcción de las aplicaciones que se utilizarán para trabajar con los datos de la base de datos.

Por lo tanto, la razón para preocuparse por el diseño de las bases de datos es que es crucial para la consistencia, integridad y precisión de los datos. Si una base de datos está mal diseñada, los usuarios tendrán dificultades a la hora de acceder a ciertos tipos de información y existe el riesgo añadido de que ciertas búsquedas puedan producir información errónea. *La información errónea es, probablemente, el peor de los resultados de un mal diseño de la base de datos. Puede repercutir muy negativamente a la empresa u organización propietaria de los datos.* De hecho, si los datos de una base de datos van a influir en la gestión del negocio, si van a servir para decidir las actuaciones de la empresa, la base de datos *debe* ser una preocupación.

Viéndolo desde una perspectiva diferente, la base de datos es como una casa que queremos que nos construyan. ¿Qué es lo primero que hay que hacer? Desde luego, lo que no vamos a hacer es buscar a un constructor que haga la casa sobre la marcha y como él quiera. Seguramente, buscaremos primero a un arquitecto que diseñe nuestra nueva casa y después haremos que el constructor la edifique. El arquitecto expresará nuestras necesidades en una serie de planos, anotando todos los requisitos de los diversos sistemas (estructural, mecánico y eléctrico). Después, el constructor pondrá los materiales necesarios, tal y como se indica en los planos y en las especificaciones.

Volviendo a la perspectiva de las bases de datos, el diseño lógico corresponde con la fase de elaboración de los planos arquitectónicos, y la implementación física de la base de datos es la casa ya construida. El diseño lógico describe el tamaño, la forma y los sistemas

necesarios para la base de datos: contiene las necesidades en cuanto a información y modo de operación del negocio. Después, se construye la implementación física del diseño lógico de la base de datos mediante el SGBD. Si pensamos en un sistema relacional, una vez creadas las tablas, establecidas las relaciones y los niveles de integridad necesarios, la base de datos está finalizada. Ahora ya se pueden crear las aplicaciones que permiten interactuar con los datos de la base de datos, y podemos estar seguros de que estas aplicaciones proporcionarán la información oportuna y, sobre todo, la información correcta.

Se pueden hacer malos diseños, pero una base de datos bien diseñada contendrá información correcta, almacenará los datos más eficientemente y será más fácil de gestionar y de mantener."

5.2 Ciclo de vida de los sistemas de información

Un *sistema de información* es el conjunto de recursos que permiten recoger, gestionar, controlar y difundir la información de toda una empresa u organización.

Desde los años setenta, los sistemas de bases de datos han ido reemplazando a los sistemas de ficheros en los sistemas de información de las empresas. Al mismo tiempo, se ha ido reconociendo la gran importancia que tienen los datos que éstas manejan, convirtiéndose en uno de sus recursos más importantes. Esto ha hecho que muchas empresas tengan departamentos que se encarguen de gestionar toda su información, que estará almacenada en una base de datos. Aparecen los papeles de *administrador de datos* y *administrador de la base de datos*, que son las personas encargadas de supervisar y controlar todas las actividades relacionadas con los datos de la empresa y con el ciclo de vida de las aplicaciones de bases de datos, respectivamente.

Un sistema de información está formado por los siguientes componentes:

- La base de datos.
- El SGBD.
- Los programas de aplicación.

- Los dispositivos físicos (ordenadores, dispositivos de almacenamiento, etc.).
- El personal que utiliza y que desarrolla el sistema.

La base de datos es un componente fundamental de un sistema de información. El ciclo de vida de un sistema de información está ligado al ciclo de vida del sistema de base de datos sobre el que se apoya. Al ciclo de vida de los sistemas de información también se le denomina *ciclo de vida de desarrollo del software*. Las etapas típicas del ciclo de vida de desarrollo del software son: planificación, recolección y análisis de los requisitos, diseño (incluyendo el diseño de la base de datos), creación de prototipos, implementación, prueba, conversión y mantenimiento. Este ciclo de vida hace énfasis en la identificación de las funciones que realiza la empresa y en el desarrollo de las aplicaciones que lleven a cabo estas funciones. Se dice que el ciclo de vida de desarrollo del software sigue un enfoque orientado a funciones, ya que los sistemas se ven desde el punto de vista de las funciones que llevan a cabo. Por esta razón, el análisis estructurado hace énfasis en los diagramas de flujo de datos, siguiendo el movimiento de los datos a través de una secuencia de transformaciones, y refinando éstas a través de una serie de niveles. Lo mismo ocurre en el diseño estructurado, que ve a un sistema como una función que se descompone sucesivamente en niveles o subfunciones.

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

En contraste, el enfoque orientado a datos centra el foco de atención en el análisis de los datos utilizados por las funciones. Esto tiene dos ventajas. La primera es que los datos son una parte considerablemente más estable que las funciones. La segunda ventaja es que la propia estructura de un esquema de base de datos requiere de un análisis sofisticado de los datos y de sus relaciones. Una vez que se haya construido un esquema para la base de

datos que sea lógico, podrían diseñarse tantas funciones como fuera necesario para sacar provecho del mismo. Sin embargo, sin un esquema tal, la base de datos sólo podría ser útil para una única aplicación. Por lo tanto, el enfoque orientado a funciones puede ser bueno para el desarrollo a corto plazo, pero pierde su valor real a largo plazo. Usando un enfoque orientado a datos, los datos pasan a ser los cimientos sobre los cuales se puede construir una gran variedad de funciones diferentes.

Por lo tanto, en este capítulo se van a estudiar cada una de las etapas del ciclo de vida de desarrollo del software desde la perspectiva del desarrollo de una aplicación de bases de datos, siguiendo un enfoque orientado a datos.

5.3 Ciclo de vida de las aplicaciones de bases de datos

Las etapas del ciclo de vida de una aplicación de bases de datos son las siguientes:

1. Planificación del proyecto.
2. Definición del sistema.
3. Recolección y análisis de los requisitos.
4. Diseño de la base de datos.
5. Selección del SGBD.
6. Diseño de la aplicación.
7. Prototipado.
8. Implementación.
9. Conversión y carga de datos.
10. Prueba.
11. Mantenimiento.

Estas etapas no son estrictamente secuenciales. De hecho hay que repetir algunas de las etapas varias veces, haciendo lo que se conocen como *ciclos de realimentación*. Por ejemplo, los problemas que se encuentran en la etapa del diseño de la base de datos pueden requerir una recolección de requisitos adicional y su posterior análisis.

A continuación, se muestran las tareas más importantes que se realizan en cada etapa.

1. Planificación del proyecto

Esta etapa conlleva la planificación de cómo se pueden llevar a cabo las etapas del ciclo de vida de la manera más eficiente. Hay tres componentes principales: el trabajo que se ha de realizar, los recursos para llevarlo a cabo y el dinero para pagar por todo ello. Como apoyo a esta etapa, se necesitará un *modelo de datos corporativo* en donde se muestren las entidades principales de la empresa y sus relaciones, y en donde se identifiquen las principales áreas funcionales. Normalmente, este modelo de datos se representa mediante un diagrama entidad–relación. En este modelo se tiene que mostrar también qué datos comparten las distintas áreas funcionales de la empresa.

La planificación de la base de datos también incluye el desarrollo de estándares que especifiquen cómo realizar la recolección de datos, cómo especificar su formato, qué documentación será necesaria y cómo se va a llevar a cabo el diseño y la implementación. El desarrollo y el mantenimiento de los estándares puede llevar bastante tiempo, pero si están bien diseñados, son una base para el personal informático en formación y para medir la calidad, además, garantizan que el trabajo se ajusta a unos patrones, independientemente de las habilidades y la experiencia del diseñador. Por ejemplo, se pueden establecer reglas sobre cómo dar nombres a los datos, lo que evitará redundancias e inconsistencias. Se deben documentar todos los aspectos legales sobre los datos y los establecidos por la empresa como, por ejemplo, qué datos deben tratarse de modo confidencial.

2. Definición del sistema

En esta etapa se especifica el ámbito y los límites de la aplicación de bases de datos, así como con qué otros sistemas interactúa. También hay que determinar quienes son los usuarios y las áreas de aplicación.

3. Recolección y análisis de los requisitos

En esta etapa se recogen y analizan los requerimientos de los usuarios y de las áreas de aplicación. Esta información se puede recoger de varias formas:

- Entrevistando al personal de la empresa, concretamente, a aquellos que son considerados expertos en las áreas de interés.
- Observando el funcionamiento de la empresa.
- Examinando documentos, sobre todo aquellos que se utilizan para recoger o visualizar información.
- Utilizando cuestionarios para recoger información de grandes grupos de usuarios.
- Utilizando la experiencia adquirida en el diseño de sistemas similares.

La información recogida debe incluir las principales áreas de aplicación y los grupos de usuarios, la documentación utilizada o generada por estas áreas de aplicación o grupos de usuarios, las transacciones requeridas por cada área de aplicación o grupo de usuarios y una lista priorizada de los requerimientos de cada área de aplicación o grupo de usuarios.

Esta etapa tiene como resultado un conjunto de documentos con las especificaciones de requisitos de los usuarios, en donde se describen las operaciones que se realizan en la empresa desde distintos puntos de vista.

La información recogida se debe estructurar utilizando *técnicas de especificación de requisitos*, como por ejemplo técnicas de análisis y diseño estructurado y diagramas de flujo de datos. También las herramientas CASE (*Computer–Aided Software Engineering*) pueden proporcionar una asistencia automatizada que garantice que los requisitos son completos y consistentes.

4. Diseño de la base de datos

Esta etapa consta de tres fases: diseño conceptual, diseño lógico y diseño físico de la base de datos. La primera fase consiste en la producción de un esquema conceptual, que

es independiente de todas las consideraciones físicas. Este modelo se refina después en un esquema lógico eliminando las construcciones que no se pueden representar en el modelo de base de datos escogido (relacional, orientado a objetos, etc.). En la tercera fase, el esquema lógico se traduce en un esquema físico para el SGBD escogido. La fase de diseño físico considera las estructuras de almacenamiento y los métodos de acceso necesarios para proporcionar un acceso eficiente a la base de datos en memoria secundaria.

Los objetivos del diseño de la base de datos son:

- Representar los datos que requieren las principales áreas de aplicación y los grupos de usuarios, y representar las relaciones entre dichos datos.
- Proporcionar un modelo de datos que soporte las transacciones que se vayan a realizar sobre los datos.
- Especificar un esquema que alcance las prestaciones requeridas para el sistema.

Hay varias estrategias a seguir para realizar el diseño: de abajo a arriba, de arriba a abajo, de dentro a fuera y la estrategia mixta. La estrategia *de abajo a arriba* parte de todos los atributos y los va agrupando en entidades y relaciones. Es apropiada cuando la base de datos es simple, con pocos atributos. La estrategia *de arriba a abajo* es más apropiada cuando se trata de bases de datos complejas. Se comienza con un esquema con entidades de alto nivel, que se van refinando para obtener entidades de bajo nivel, atributos y relaciones. La estrategia *de dentro a fuera* es similar a la estrategia de abajo a arriba, pero difiere en que se parte de los conceptos principales y se va extendiendo el esquema para considerar también otros conceptos, asociados con los que se han identificado en primer lugar. La estrategia *mixta* utiliza ambas estrategias, de abajo a arriba y de arriba a abajo, con un esquema de divide y vencerás. Se obtiene un esquema inicial de alto nivel, se divide en partes, y de cada parte se obtiene un subesquema. Estos subesquemas se integran después para obtener el modelo final.

5. Selección del SGBD

Si no se dispone de un SGBD, o el que hay se encuentra obsoleto, se debe escoger un SGBD que sea adecuado para el sistema de información. Esta elección se debe hacer en

cualquier momento antes del diseño lógico.

6. Diseño de la aplicación

En esta etapa se diseñan los programas de aplicación que usarán y procesarán la base de datos. Esta etapa y el diseño de la base de datos, son paralelas. En la mayor parte de los casos no se puede finalizar el diseño de las aplicaciones hasta que se ha terminado con el diseño de la base de datos. Por otro lado, la base de datos existe para dar soporte a las aplicaciones, por lo que habrá una realimentación desde el diseño de las aplicaciones al diseño de la base de datos.

En esta etapa hay que asegurarse de que toda la funcionalidad especificada en los requisitos de usuario se encuentra en el diseño de la aplicación. Habrá algunos programas que utilicen y procesen los datos de la base de datos.

Además, habrá que diseñar las interfaces de usuario, aspecto muy importante que se suele ignorar. El sistema debe ser fácil de aprender, fácil de usar, ser directo y estar "dispuesto a perdonar". Si la interface no tiene estas características, el sistema dará problemas, sin lugar a dudas.

7. Prototipado

Esta etapa, que es opcional, es para construir prototipos de la aplicación que permitan a los diseñadores y a los usuarios probar el sistema. Un prototipo es un modelo de trabajo de las aplicaciones del sistema. El prototipo no tiene toda la funcionalidad del sistema final, pero es suficiente para que los usuarios puedan utilizar el sistema e identificar qué aspectos están bien y cuáles no son adecuados, además de poder sugerir mejoras o la inclusión de nuevos elementos. Este proceso permite que quienes diseñan e implementan el sistema sepan si han interpretado correctamente los requisitos de los usuarios. Otra ventaja de los prototipos es que se construyen rápidamente.

Esta etapa es imprescindible cuando el sistema que se va a implementar tiene un gran coste, alto riesgo o utiliza nuevas tecnologías.

8. Implementación

En esta etapa se crean las definiciones de la base de datos a nivel conceptual, externo e interno, así como los programas de aplicación. La implementación de la base de datos se realiza mediante las sentencias del lenguaje de definición de datos (LDD) del SGBD escogido. Estas sentencias se encargan de crear el esquema de la base de datos, los ficheros en donde se almacenarán los datos y las vistas de los usuarios.

Los programas de aplicación se implementan utilizando lenguajes de tercera o cuarta generación. Partes de estas aplicaciones son transacciones sobre la base de datos, que se implementan mediante el lenguaje de manejo de datos (LMD) del SGBD. Las sentencias de este lenguaje se pueden embeber en un lenguaje de programación anfitrión como Visual Basic, Delphi, C, C++, Java, COBOL, Fortran, Ada o Pascal. En esta etapa, también se implementan los menús, los formularios para la introducción de datos y los informes de visualización de datos. Para ello, el SGBD puede disponer de lenguajes de cuarta generación que permiten el desarrollo rápido de aplicaciones mediante lenguajes de consultas no procedurales, generadores de informes, generadores de formularios, generadores de gráficos y generadores de aplicaciones.

También se implementan en esta etapa todos los controles de seguridad e integridad. Algunos de estos controles se pueden implementar mediante el LDD y otros puede que haya que implementarlos mediante utilidades del SGBD o mediante programas de aplicación.

9. Conversión y carga de datos

Esta etapa es necesaria cuando se está reemplazando un sistema antiguo por uno nuevo. Los datos se cargan desde el sistema viejo al nuevo directamente o, si es necesario, se convierten al formato que requiera el nuevo SGBD y luego se cargan. Si es posible, los programas de aplicación del sistema antiguo también se convierten para que se puedan utilizar en el

sistema nuevo.

10. Prueba

En esta etapa se prueba y valida el sistema con los requisitos especificados por los usuarios. Para ello, se debe diseñar una batería de tests con datos reales, que se deben llevar a cabo de manera metódica y rigurosa. Es importante darse cuenta de que la fase de prueba no sirve para demostrar que no hay fallos, sirve para encontrarlos. Si la fase de prueba se lleva a cabo correctamente, descubrirá los errores en los programas de aplicación y en la estructura de la base de datos. Además, demostrará que los programas "parecen" trabajar tal y como se especificaba en los requisitos y que las prestaciones deseadas "parecen" obtenerse. Por último, en las pruebas se podrá hacer una medida de la fiabilidad y la calidad del software desarrollado.

11. Mantenimiento

Una vez que el sistema está completamente implementado y probado, se pone en marcha. El sistema está ahora en la fase de mantenimiento en la que se llevan a cabo las siguientes tareas:

- Monitorización de las prestaciones del sistema. Si las prestaciones caen por debajo de un determinado nivel, puede ser necesario reorganizar la base de datos.
- Mantenimiento y actualización del sistema. Cuando sea necesario, los nuevos requisitos que vayan surgiendo se incorporarán al sistema, siguiendo de nuevo las etapas del ciclo de vida que se acaban de presentar.

5.4 Diseño de bases de datos

En este apartado se describen con más detalle los objetivos de cada una de las etapas del diseño de bases de datos: diseño conceptual, diseño lógico y diseño físico. La metodología a seguir en cada una de estas etapas se describe en los tres capítulos que siguen a éste.

5.4.1 Diseño conceptual

En esta etapa se debe construir un esquema de la información que se usa en la empresa, independientemente de cualquier consideración física. A este esquema se le denomina *esquema conceptual*. Al construir el esquema, los diseñadores descubren la semántica (significado) de los datos de la empresa: encuentran entidades, atributos y relaciones. El objetivo es comprender:

- La perspectiva que cada usuario tiene de los datos.
- La naturaleza de los datos, independientemente de su representación física.
- El uso de los datos a través de las áreas de aplicación.

El esquema conceptual se puede utilizar para que el diseñador transmita a la empresa lo que ha entendido sobre la información que ésta maneja. Para ello, ambas partes deben estar familiarizadas con la notación utilizada en el esquema. La más popular es la notación del modelo entidad–relación, que se describirá en el capítulo dedicado al diseño conceptual.

El esquema conceptual se construye utilizando la información que se encuentra en la especificación de los requisitos de usuario. El diseño conceptual es completamente independiente de los aspectos de implementación, como puede ser el SGBD que se vaya a usar, los programas de aplicación, los lenguajes de programación, el hardware disponible o cualquier otra consideración física. Durante todo el proceso de desarrollo del esquema conceptual éste se prueba y se valida con los requisitos de los usuarios. El esquema conceptual es una fuente de información para el diseño lógico de la base de datos.

5.4.2 Diseño lógico

El diseño lógico es el proceso de construir un esquema de la información que utiliza la empresa, basándose en un modelo de base de datos específico, independiente del SGBD concreto que se vaya a utilizar y de cualquier otra consideración física.

En esta etapa, se transforma el esquema conceptual en un esquema lógico que utilizará las estructuras de datos del modelo de base de datos en el que se basa el SGBD que se vaya

a utilizar, como puede ser el modelo relacional, el modelo de red, el modelo jerárquico o el modelo orientado a objetos. Conforme se va desarrollando el esquema lógico, éste se va probando y validando con los requisitos de usuario.

La *normalización* es una técnica que se utiliza para comprobar la validez de los esquemas lógicos basados en el modelo relacional, ya que asegura que las relaciones (tablas) obtenidas no tienen datos redundantes. Esta técnica se presenta en el capítulo dedicado al diseño lógico de bases de datos.

El esquema lógico es una fuente de información para el diseño físico. Además, juega un papel importante durante la etapa de mantenimiento del sistema, ya que permite que los futuros cambios que se realicen sobre los programas de aplicación o sobre los datos, se representen correctamente en la base de datos.

Tanto el diseño conceptual, como el diseño lógico, son procesos iterativos, tienen un punto de inicio y se van refinando continuamente. Ambos se deben ver como un proceso de aprendizaje en el que el diseñador va comprendiendo el funcionamiento de la empresa y el significado de los datos que maneja. El diseño conceptual y el diseño lógico son etapas clave para conseguir un sistema que funcione correctamente. Si el esquema no es una representación fiel de la empresa, será difícil, sino imposible, definir todas las vistas de usuario (esquemas externos), o mantener la integridad de la base de datos. También puede ser difícil definir la implementación física o el mantener unas prestaciones aceptables del sistema. Además, hay que tener en cuenta que la capacidad de ajustarse a futuros cambios es un sello que identifica a los buenos diseños de bases de datos. Por todo esto, es fundamental dedicar el tiempo y las energías necesarias para producir el mejor esquema que sea posible.

5.4.3 Diseño físico

El diseño físico es el proceso de producir la descripción de la implementación de la base de datos en memoria secundaria: estructuras de almacenamiento y métodos de acceso que garanticen un acceso eficiente a los datos.

Para llevar a cabo esta etapa, se debe haber decidido cuál es el SGBD que se va a utilizar, ya que el esquema físico se adapta a él. Entre el diseño físico y el diseño lógico hay

una realimentación, ya que algunas de las decisiones que se tomen durante el diseño físico para mejorar las prestaciones, pueden afectar a la estructura del esquema lógico.

En general, el propósito del diseño físico es describir cómo se va a implementar físicamente el esquema lógico obtenido en la fase anterior. Concretamente, en el modelo relacional, esto consiste en:

- Obtener un conjunto de relaciones (tablas) y las restricciones que se deben cumplir sobre ellas.
- Determinar las estructuras de almacenamiento y los métodos de acceso que se van a utilizar para conseguir unas prestaciones óptimas.
- Diseñar el modelo de seguridad del sistema.

5.5 Diseño de aplicaciones

En este apartado se examinan los dos aspectos del diseño de las aplicaciones: el diseño de las transacciones y el diseño de las interfaces de usuario.

5.5.1 Diseño de transacciones

Una transacción es un conjunto de acciones llevadas a cabo por un usuario o un programa de aplicación, que acceden o cambian el contenido de la base de datos. Las transacciones representan eventos del mundo real, como registrar un inmueble para ponerlo en alquiler, concertar una visita con un cliente a un inmueble, dar de alta un nuevo empleado o registrar un nuevo cliente. Estas transacciones se deben realizar sobre la base de datos para que ésta siga siendo un fiel reflejo de la realidad.

Una transacción puede estar compuesta por varias operaciones, como la transferencia de dinero de una cuenta bancaria a otra. Sin embargo, desde el punto de vista del usuario, estas operaciones conforman una sola tarea. Desde el punto de vista del SGBD, una transacción lleva a la base de datos de un estado consistente a otro estado consistente. El SGBD garantiza la consistencia de la base de datos incluso si se produce algún fallo, y también garantiza

que una vez se ha finalizado una transacción, los cambios realizados por ésta quedan permanentemente en la base de datos, no se pueden perder ni deshacer (a menos que se realice otra transacción que compense el efecto de la primera). Si la transacción no se puede finalizar por cualquier motivo, el SGBD garantiza que los cambios realizados por esta transacción son deshechos. En el ejemplo de la transferencia de fondos entre dos cuentas bancarias, si el dinero se extrae de una cuenta y la transacción falla antes de que el dinero se ingrese en la otra cuenta, el SGBD deshará la extracción de fondos.

El objetivo del diseño de las transacciones es definir y documentar las características de alto nivel de las transacciones que requiere el sistema. Esta tarea se debe llevar a cabo al principio del proceso de diseño para garantizar que el esquema lógico es capaz de soportar todas las transacciones necesarias. Las características que se deben recoger de cada transacción son las siguientes:

- Datos que utiliza la transacción.
- Características funcionales de la transacción.
- Salida de la transacción.
- Importancia para los usuarios.
- Frecuencia de utilización.

Hay tres tipos de transacciones:

- En las *transacciones de recuperación* se accede a los datos para visualizarlos en la pantalla a modo de informe.
- En las *transacciones de actualización* se insertan, borran o actualizan datos de la base de datos.
- En las *transacciones mixtas* se mezclan operaciones de recuperación de datos y de actualización.

El diseño de las transacciones utiliza la información dada en las especificaciones de requisitos de usuario.

5.5.2 Diseño de interfaces de usuario

Antes de implementar los formularios y los informes, hay que diseñar su aspecto. Es conveniente tener en cuenta las siguientes recomendaciones:

- Utilizar títulos que sean significativos, que identifiquen sin ambigüedad el propósito del informe o formulario.
- Dar instrucciones breves y fáciles de comprender.
- Agrupar y secuenciar los campos de forma lógica.
- Hacer que el aspecto del informe o formulario sea atractivo a la vista.
- Utilizar nombres familiares para etiquetar los campos.
- Utilizar terminología y abreviaturas consistentes.
- Hacer un uso razonable y consistente de los colores.
- Dejar un espacio visible para los datos de entrada y delimitarlos.
- Permitir un uso sencillo y adecuado del cursor.
- Permitir la corrección carácter a carácter y de campos completos.
- Dar mensajes de error para los valores "ilegales".
- Marcar los campos que sean opcionales.
- Dar mensajes a nivel de campo para explicar su significado.
- Dar una señal que indique cuándo el informe o formulario está completo.

5.6 Herramientas CASE

Cuando se hace la planificación de la base de datos, la primera etapa del ciclo de vida de las aplicaciones de bases de datos, también se puede escoger una herramienta CASE (Computer–Aided Software Engineering) que permita llevar a cabo el resto de tareas del modo más eficiente y efectivo posible. Una herramienta CASE suele incluir:

- Un diccionario de datos para almacenar información sobre los datos de la aplicación de bases de datos.
- Herramientas de diseño para dar apoyo al análisis de datos.
- Herramientas que permitan desarrollar el modelo de datos corporativo, así como los esquemas conceptual y lógico.
- Herramientas para desarrollar los prototipos de las aplicaciones.

El uso de las herramientas CASE puede mejorar la productividad en el desarrollo de una aplicación de bases de datos. Y por productividad se entiende tanto la eficiencia en el desarrollo, como la efectividad del sistema desarrollado. La eficiencia se refiere al coste, tanto en tiempo como en dinero, de desarrollar la aplicación. La efectividad se refiere al grado en que el sistema satisface las necesidades de los usuarios. Para obtener una buena productividad, subir el nivel de efectividad puede ser más importante que aumentar la eficiencia.

5.7 Administración de datos y de la base de datos

El administrador de datos y el administrador de la base de datos son las personas o grupos de personas encargadas de gestionar y controlar todas las actividades que tienen que ver con los datos de la empresa y con la base de datos, respectivamente.

El administrador de datos es quien entiende los datos y las necesidades de la empresa con respecto a dichos datos. Su trabajo es decidir qué datos deben almacenarse en la base de datos y establecer políticas para mantener y gestionar los datos una vez hayan sido almacenados. Un ejemplo de tal política sería una que estableciera quién puede realizar qué operaciones sobre qué datos y en qué circunstancias.

La persona (o personas) que se encarga de implementar las decisiones del administrador de datos es el administrador de la base de datos. Su trabajo es crear la base de datos e implementar los controles necesarios para que se respeten las políticas establecidas por el administrador de datos. El administrador de la base de datos es el responsable de garantizar que el sistema obtenga las prestaciones deseadas, además de prestar otros servicios técnicos.

El administrador de datos juega un papel más importante que el administrador de la base de datos en las siguientes etapas del ciclo de vida: planificación de la base de datos, definición del sistema, recolección y análisis de los requisitos, diseño conceptual y diseño lógico de la base de datos. En el resto de las etapas es donde el administrador de la base de datos tiene el papel más importante: selección del SGBD, diseño de las aplicaciones, diseño físico, prototipado, implementación, conversión y carga de datos, prueba y mantenimiento.

5.8 Resumen

Un sistema de información es el conjunto de recursos que se utilizan para recoger, gestionar, controlar y divulgar la información dentro de una empresa u organización. Desde los años setenta los sistemas de bases de datos han ido reemplazando a los sistemas de ficheros en los sistemas de información de las empresas, de modo que éstos constan de los siguientes componentes: la base de datos, el SGBD, los programas de aplicación, los equipos informáticos y el personal que utiliza y que desarrolla el sistema.

La base de datos es uno de los componentes principales de un sistema de información, por lo que el ciclo de vida de un sistema de información está inherentemente ligado al ciclo de vida de la base de datos sobre la que se apoya. Las etapas de este ciclo de vida son: planificación de la base de datos, definición del sistema, recolección y análisis de los requisitos, diseño de la base de datos, selección del SGBD, diseño de aplicaciones, elaboración de prototipos, implementación, conversión y carga de datos, prueba y mantenimiento.

En el diseño de una base de datos se debe realizar un modelo de datos que ayude a entender el significado de los datos y que facilite la comunicación en cuanto a los requisitos de información. La primera etapa es el diseño conceptual, en donde se construye un esquema de la información que maneja la empresa, independientemente de todas las consideraciones físicas. Después viene el diseño lógico, en el que el esquema anterior se transforma según el modelo de base de datos que se vaya a utilizar para implementar el sistema. Por último, en la etapa del diseño físico, se produce una descripción de la implementación de la base de datos en memoria secundaria.

El diseño de las aplicaciones, una fase que se debe llevar a cabo en paralelo con el diseño

de la base de datos, está compuesta por dos actividades: el diseño de las transacciones y el diseño de las interfaces de usuario de informes y formularios.

Las herramientas CASE permiten que el desarrollo de los sistemas de información se realice de modo eficiente y efectivo.

La administración de datos consiste en la gestión de los datos como recurso, mientras que la administración de la base de datos es la gestión de la base de datos física.

Bibliografía

Siendo este un tema introductorio, se encuentra reflejado en la mayoría de los textos sobre bases de datos. Este capítulo se ha elaborado a partir de los textos de Date (1993), Connolly, Begg y Strachan (1996) y Hansen y Hansen (1997).

Capítulo 6

Diseño conceptual de bases de datos. Modelo entidad–relación

En este capítulo se presenta una metodología para el diseño conceptual de bases de datos que se basa en el modelo de datos más popular en la actualidad, el modelo entidad–relación.

6.1 Introducción

Para la introducción a este capítulo se toman algunos párrafos del texto de Batini, Ceri y Navathe (1994).

"El diseño de bases de datos es el proceso por el que se determina la organización de una base de datos, incluidos su estructura, contenido y las aplicaciones que se han de desarrollar. Durante mucho tiempo, el diseño de bases de datos fue considerado una tarea para expertos: más un arte que una ciencia. Sin embargo, se ha progresado mucho en el diseño de bases de datos y éste se considera ahora una disciplina estable, con métodos y técnicas propios. Debido a la creciente aceptación de las bases de datos por parte de la industria y el gobierno en el plano comercial, y a una variedad de aplicaciones científicas y técnicas, el diseño de bases de datos desempeña un papel central en el empleo de los recursos de información en la mayoría de las organizaciones. El diseño de bases de datos ha pasado a constituir parte

de la formación general de los informáticos, en el mismo nivel que la capacidad de construir algoritmos usando un lenguaje de programación convencional."

"Las últimas dos décadas se han caracterizado por un fuerte crecimiento en el número e importancia de las aplicaciones de bases de datos. Las bases de datos son componentes esenciales de los sistemas de información, usadas rutinariamente en todos los computadores [...]. El diseño de bases de datos se ha convertido en una actividad popular, desarrollada no sólo por profesionales sino también por no especialistas.

A finales de la década de 1960, cuando las bases de datos entraron por primera vez en el mercado del software, los diseñadores de bases de datos actuaban como artesanos, con herramientas muy primitivas: diagramas de bloques y estructuras de registros eran los formatos comunes para las especificaciones, y el diseño de bases de datos se confundía frecuentemente con la implantación de las bases de datos. Esta situación ahora ha cambiado: los métodos y modelos de diseño de bases de datos han evolucionado paralelamente con el progreso de la tecnología en los sistemas de bases de datos. Se ha entrado en la era de los sistemas relacionales de bases de datos, que ofrecen poderosos lenguajes de consulta, herramientas para el desarrollo de aplicaciones e interfaces amables con los usuarios. La tecnología de bases de datos cuenta ya con un marco teórico, que incluye la teoría relacional de datos, procesamiento y optimización de consultas, control de concurrencia, gestión de transacciones y recuperación, etc.

Según ha avanzado la tecnología de bases de datos, así se han desarrollado las metodologías y técnicas de diseño. Se ha alcanzado un consenso, por ejemplo, sobre la descomposición del proceso de diseño en fases, sobre los principales objetivos de cada fase y sobre las técnicas para conseguir estos objetivos."

"Desafortunadamente, las metodologías de diseño de bases de datos no son muy populares; la mayoría de las organizaciones y de los diseñadores individuales confía muy poco en las metodologías para llevar a cabo el diseño y esto se considera, con frecuencia, una de las principales causas de fracaso en el desarrollo de los sistemas de información. Debido a la falta de enfoques estructurados para el diseño de bases de datos, a menudo se subestiman el tiempo o los recursos necesarios para un proyecto de bases de datos, las bases de datos son inadecuadas o ineficientes en relación a las demandas de la aplicación, la documentación es

limitada y el mantenimiento es difícil.

Muchos de estos problemas se deben a la falta de una claridad que permita entender la naturaleza exacta de los datos, a un nivel conceptual y abstracto. En muchos casos, los datos se describen desde el comienzo del proyecto en términos de las estructuras finales de almacenamiento; no se da peso a un entendimiento de las propiedades estructurales de los datos que sea independiente de los detalles de la realización."

6.2 Metodología de diseño de bases de datos

El diseño de una base de datos es un proceso complejo que abarca decisiones a muy distintos niveles. La complejidad se controla mejor si se descompone el problema en subproblemas y se resuelve cada uno de estos subproblemas independientemente, utilizando técnicas específicas. Así, el diseño de una base de datos se descompone en diseño conceptual, diseño lógico y diseño físico.

El diseño conceptual parte de las especificaciones de requisitos de usuario y su resultado es el esquema conceptual de la base de datos. Un *esquema conceptual* es una descripción de alto nivel de la estructura de la base de datos, independientemente del SGBD que se vaya a utilizar para manipularla. Un *modelo conceptual* es un lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es describir el contenido de información de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar esta información.

El diseño lógico parte del esquema conceptual y da como resultado un esquema lógico. Un *esquema lógico* es una descripción de la estructura de la base de datos en términos de las estructuras de datos que puede procesar un tipo de SGBD. Un *modelo lógico* es un lenguaje usado para especificar esquemas lógicos (modelo relacional, modelo de red, etc.). El diseño lógico depende del tipo de SGBD que se vaya a utilizar, no depende del producto concreto.

El diseño físico parte del esquema lógico y da como resultado un esquema físico. Un *esquema físico* es una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Por ello, el diseño físico depende del SGBD concreto y el esquema físico

se expresa mediante su lenguaje de definición de datos.

6.3 Modelos de datos

Un *modelo de datos* es una serie de conceptos que puede utilizarse para describir un conjunto de datos y las operaciones para manipularlos. Hay dos tipos de modelos de datos: los modelos conceptuales y los modelos lógicos. Los *modelos conceptuales* se utilizan para representar la realidad a un alto nivel de abstracción. Mediante los modelos conceptuales se puede construir una descripción de la realidad fácil de entender. En los *modelos lógicos*, las descripciones de los datos tienen una correspondencia sencilla con la estructura física de la base de datos.

En el diseño de bases de datos se usan primero los modelos conceptuales para lograr una descripción de alto nivel de la realidad, y luego se transforma el esquema conceptual en un esquema lógico. El motivo de realizar estas dos etapas es la dificultad de abstraer la estructura de una base de datos que presente cierta complejidad. Un *esquema* es un conjunto de representaciones lingüísticas o gráficas que describen la estructura de los datos de interés.

Los modelos conceptuales deben ser buenas herramientas para representar la realidad, por lo que deben poseer las siguientes cualidades:

- *Expresividad*: deben tener suficientes conceptos para expresar perfectamente la realidad.
- *Simplicidad*: deben ser simples para que los esquemas sean fáciles de entender.
- *Minimalidad*: cada concepto debe tener un significado distinto.
- *Formalidad*: todos los conceptos deben tener una interpretación única, precisa y bien definida.

En general, un modelo no es capaz de expresar todas las propiedades de una realidad determinada, por lo que hay que añadir aserciones que complementen el esquema.

6.4 El modelo entidad–relación

El modelo entidad–relación es el modelo conceptual más utilizado para el diseño conceptual de bases de datos. Fue introducido por Peter Chen en 1976. El modelo entidad–relación está formado por un conjunto de conceptos que permiten describir la realidad mediante un conjunto de representaciones gráficas y lingüísticas.

Originalmente, el modelo entidad–relación sólo incluía los conceptos de entidad, relación y atributo. Más tarde, se añadieron otros conceptos, como los atributos compuestos y las jerarquías de generalización, en lo que se ha denominado *modelo entidad–relación extendido.*

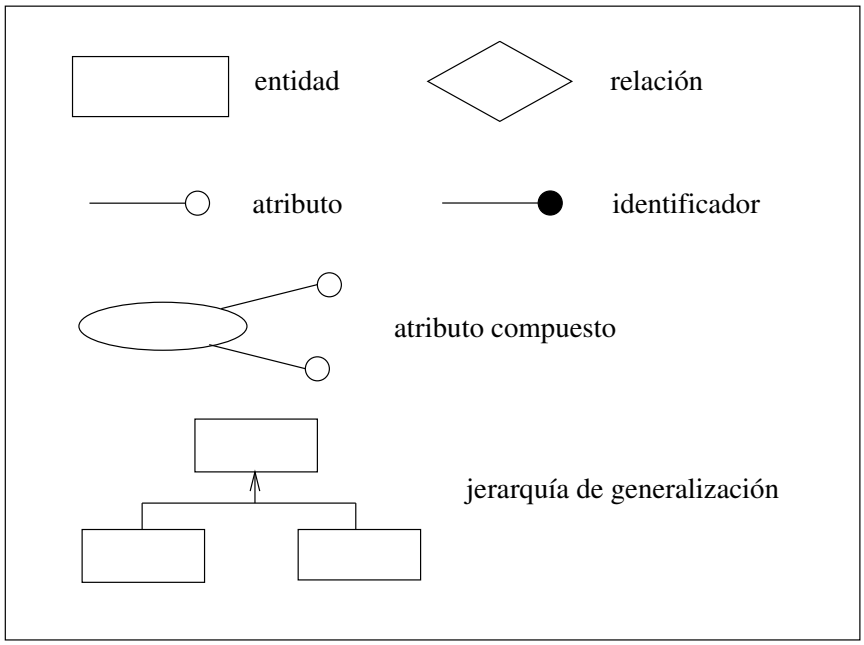

Figura 6.1: *Conceptos del modelo entidad–relación extendido.*

Entidad

Cualquier tipo de objeto o concepto sobre el que se recoge información: cosa, persona, concepto abstracto o suceso. Por ejemplo: coches, casas, empleados, clientes, empresas, oficios, diseños de productos, conciertos, excursiones, etc. Las entidades se representan gráficamente mediante rectángulos y su nombre aparece en el interior. Un nombre de entidad sólo puede aparecer una vez en el esquema conceptual.

Hay dos tipos de entidades: fuertes y débiles. Una *entidad débil* es una entidad cuya existencia depende de la existencia de otra entidad. Una *entidad fuerte* es una entidad que no es débil.

Relación (interrelación)

Es una correspondencia o asociación entre dos o más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior.

Las entidades que están involucradas en una determinada relación se denominan *entidades participantes*. El número de participantes en una relación es lo que se denomina *grado* de la relación. Por lo tanto, una relación en la que participan dos entidades es una relación *binaria*; si son tres las entidades participantes, la relación es *ternaria*; etc.

Una *relación recursiva* es una relación donde la misma entidad participa más de una vez en la relación con distintos papeles. El nombre de estos papeles es importante para determinar la función de cada participación.

La *cardinalidad* con la que una entidad participa en una relación especifica el número mínimo y el número máximo de correspondencias en las que puede tomar parte cada ocurrencia de dicha entidad. La participación de una entidad en una relación es *obligatoria (total)* si la existencia de cada una de sus ocurrencias requiere la existencia de, al menos, una ocurrencia de la otra entidad participante. Si no, la participación es *opcional (parcial)*. Las reglas que definen la cardinalidad de las relaciones son las *reglas de negocio*.

A veces, surgen problemas cuando se está diseñado un esquema conceptual. Estos problemas, denominados *trampas*, suelen producirse a causa de una mala interpretación en el significado de alguna relación, por lo que es importante comprobar que el esquema conceptual carece de dichas trampas. En general, para encontrar las trampas, hay que asegurarse de que se entiende completamente el significado de cada relación. Si no se entienden las relaciones, se puede crear un esquema que no represente fielmente la realidad.

Una de las trampas que pueden encontrarse ocurre cuando el esquema representa una relación entre entidades, pero el camino entre algunas de sus ocurrencias es ambiguo. El

modo de resolverla es reestructurando el esquema para representar la asociación entre las entidades correctamente.

Otra de las trampas sucede cuando un esquema sugiere la existencia de una relación entre entidades, pero el camino entre una y otra no existe para algunas de sus ocurrencias. En este caso, se produce una pérdida de información que se puede subsanar introduciendo la relación que sugería el esquema y que no estaba representada.

Atributo

Es una característica de interés o un hecho sobre una entidad o sobre una relación. Los atributos representan las propiedades básicas de las entidades y de las relaciones. Toda la información extensiva es portada por los atributos. Gráficamente, se representan mediante bolitas que cuelgan de las entidades o relaciones a las que pertenecen.

Cada atributo tiene un conjunto de valores asociados denominado *dominio*. El dominio define todos los valores posibles que puede tomar un atributo. Puede haber varios atributos definidos sobre un mismo dominio.

Los atributos pueden ser simples o compuestos. Un *atributo simple* es un atributo que tiene un solo componente, que no se puede dividir en partes más pequeñas que tengan un significado propio. Un *atributo compuesto* es un atributo con varios componentes, cada uno con un significado por sí mismo. Un grupo de atributos se representa mediante un atributo compuesto cuando tienen afinidad en cuanto a su significado, o en cuanto a su uso. Un atributo compuesto se representa gráficamente mediante un óvalo.

Los atributos también pueden clasificarse en monovalentes o polivalentes. Un *atributo monovalente* es aquel que tiene un solo valor para cada ocurrencia de la entidad o relación a la que pertenece. Un *atributo polivalente* es aquel que tiene varios valores para cada ocurrencia de la entidad o relación a la que pertenece. A estos atributos también se les denomina *multivaluados*, y pueden tener un número máximo y un número mínimo de valores. La *cardinalidad* de un atributo indica el número mínimo y el número máximo de valores que puede tomar para cada ocurrencia de la entidad o relación a la que pertenece. El valor por omisión es $(1, 1)$.

Por último, los atributos pueden ser derivados. Un *atributo derivado* es aquel que representa un valor que se puede obtener a partir del valor de uno o varios atributos, que no necesariamente deben pertenecer a la misma entidad o relación.

Identificador

Un identificador de una entidad es un atributo o conjunto de atributos que determina de modo único cada ocurrencia de esa entidad. Un identificador de una entidad debe cumplir dos condiciones:

1. No pueden existir dos ocurrencias de la entidad con el mismo valor del identificador.
2. Si se omite cualquier atributo del identificador, la condición anterior deja de cumplirse.

Toda entidad tiene al menos un identificador y puede tener varios identificadores alternativos. Las relaciones no tienen identificadores.

Jerarquía de generalización

Una entidad E es una generalización de un grupo de entidades E_1, E_2, ... E_n, si cada ocurrencia de cada una de esas entidades es también una ocurrencia de E. Todas las propiedades de la entidad genérica E son heredadas por las subentidades.

Cada jerarquía es total o parcial, y exclusiva o superpuesta. Una jerarquía es *total* si cada ocurrencia de la entidad genérica corresponde al menos con una ocurrencia de alguna subentidad. Es *parcial* si existe alguna ocurrencia de la entidad genérica que no corresponde con ninguna ocurrencia de ninguna subentidad. Una jerarquía es *exclusiva* si cada ocurrencia de la entidad genérica corresponde, como mucho, con una ocurrencia de una sola de las subentidades. Es *superpuesta* si existe alguna ocurrencia de la entidad genérica que corresponde a ocurrencias de dos o más subentidades diferentes.

Un *subconjunto* es un caso particular de generalización con una sola entidad como subentidad. Un subconjunto siempre es una jerarquía parcial y exclusiva.

6.5 Metodología de diseño conceptual

El primer paso en el diseño de una base de datos es la producción del esquema conceptual. Normalmente, se construyen varios esquemas conceptuales, cada uno para representar las distintas visiones que los usuarios tienen de la información. Cada una de estas visiones suelen corresponder a las diferentes áreas funcionales de la empresa como, por ejemplo, producción, ventas, recursos humanos, etc.

Estas visiones de la información, denominadas *vistas*, se pueden identificar de varias formas. Una opción consiste en examinar los diagramas de flujo de datos, que se pueden haber producido previamente, para identificar cada una de las áreas funcionales. La otra opción consiste en entrevistar a los usuarios, examinar los procedimientos, los informes y los formularios, y también observar el funcionamiento de la empresa.

A los esquemas conceptuales correspondientes a cada vista de usuario se les denomina *esquemas conceptuales locales*. Cada uno de estos esquemas se compone de entidades, relaciones, atributos, dominios de atributos e identificadores. El esquema conceptual también tendrá una documentación, que se irá produciendo durante su desarrollo. Las tareas a realizar en el diseño conceptual son las siguientes:

1. Identificar las entidades.
2. Identificar las relaciones.
3. Identificar los atributos y asociarlos a entidades y relaciones.
4. Determinar los dominios de los atributos.
5. Determinar los identificadores.
6. Determinar las jerarquías de generalización (si las hay).
7. Dibujar el diagrama entidad–relación.
8. Revisar el esquema conceptual local con el usuario.

1. Identificar las entidades

En primer lugar hay que definir los principales objetos que interesan al usuario. Estos objetos serán las entidades. Una forma de identificar las entidades es examinar las especificaciones de requisitos de usuario. En estas especificaciones se buscan los nombres o los sintagmas nominales que se mencionan (por ejemplo: número de empleado, nombre de empleado, número de inmueble, dirección del inmueble, alquiler, número de habitaciones). También se buscan objetos importantes como personas, lugares o conceptos de interés, excluyendo aquellos nombres que sólo son propiedades de otros objetos. Por ejemplo, se pueden agrupar el número de empleado y el nombre de empleado en una entidad denominada *empleado*, y agrupar número de inmueble, dirección del inmueble, alquiler y número de habitaciones en otra entidad denominada *inmueble*.

Otra forma de identificar las entidades es buscar aquellos objetos que existen por sí mismos. Por ejemplo, *empleado* es una entidad porque los empleados existen, sepamos o no sus nombres, direcciones y teléfonos. Siempre que sea posible, el usuario debe colaborar en la identificación de las entidades.

A veces, es difícil identificar las entidades por la forma en que aparecen en las especificaciones de requisitos. Los usuarios, a veces, hablan utilizando ejemplos o analogías. En lugar de hablar de empleados en general, hablan de personas concretas, o bien, hablan de los puestos que ocupan esas personas.

Para liarlo aún más, los usuarios usan, muchas veces, sinónimos y homónimos. Dos palabras son sinónimos cuando tienen el mismo significado. Los homónimos ocurren cuando la misma palabra puede tener distintos significados dependiendo del contexto.

No siempre es obvio saber si un objeto es una entidad, una relación o un atributo. Por ejemplo ¿cómo se podría clasificar *matrimonio*? Pues de cualquiera de las tres formas. El análisis es subjetivo, por lo que distintos diseñadores pueden hacer distintas interpretaciones, aunque todas igualmente válidas. Todo depende de la opinión y la experiencia de cada uno. Los diseñadores de bases de datos deben tener una visión selectiva y clasificar las cosas que observan dentro del contexto de la empresa u organización. A partir de unas especificaciones de usuario es posible que no se pueda deducir un conjunto único de entidades, pero después de varias iteraciones del proceso de análisis, se llegará a obtener un conjunto de entidades

que sean adecuadas para el sistema que se ha de construir.

Conforme se van identificando las entidades, se les dan nombres que tengan un significado y que sean obvias para el usuario. Los nombres de las entidades y sus descripciones se anotan en el diccionario de datos. Cuando sea posible, se debe anotar también el número aproximado de ocurrencias de cada entidad. Si una entidad se conoce por varios nombres, éstos se deben anotar en el diccionario de datos como alias o sinónimos.

2. Identificar las relaciones

Una vez definidas las entidades, se deben definir las relaciones existentes entre ellas. Del mismo modo que para identificar las entidades se buscaban nombres en las especificaciones de requisitos, para identificar las relaciones se suelen buscar las expresiones verbales (por ejemplo: oficina tiene empleados, empleado gestiona inmueble, cliente visita inmueble). Si las especificaciones de requisitos reflejan estas relaciones es porque son importantes para la empresa y, por lo tanto, se deben reflejar en el esquema conceptual.

Pero sólo interesan las relaciones que son necesarias. En el ejemplo anterior, se han identificado las relaciones *empleado gestiona inmueble* y *cliente visita inmueble*. Se podría pensar en incluir una relación entre empleado y cliente: *empleado atiende a cliente*, pero observando las especificaciones de requisitos no parece que haya interés en modelar tal relación.

La mayoría de las relaciones son binarias (entre dos entidades), pero no hay que olvidar que también puede haber relaciones en las que participen más de dos entidades, así como relaciones recursivas.

Es muy importante repasar las especificaciones para comprobar que todas las relaciones, explícitas o implícitas, se han encontrado. Si se tienen pocas entidades, se puede comprobar por parejas si hay alguna relación entre ellas. De todos modos, las relaciones que no se identifican ahora se suelen encontrar cuando se valida el esquema con las transacciones que debe soportar.

Una vez identificadas todas las relaciones, hay que determinar la cardinalidad mínima y máxima con la que participa cada entidad en cada una de ellas. De este modo, el esquema representa de un modo más explícito la semántica de las relaciones. La cardinalidad es un

tipo de restricción que se utiliza para comprobar y mantener la calidad de los datos. Estas restricciones son aserciones sobre las entidades que se pueden aplicar cuando se actualiza la base de datos para determinar si las actualizaciones violan o no las reglas establecidas sobre la semántica de los datos.

Conforme se van identificando las relaciones, se les van asignando nombres que tengan significado para el usuario. En el diccionario de datos se anotan los nombres de las relaciones, su descripción y las cardinalidades con las que participan las entidades en ellas.

3. Identificar los atributos y asociarlos a entidades y relaciones

Al igual que con las entidades, se buscan nombres en las especificaciones de requisitos. Son atributos los nombres que identifican propiedades, cualidades, identificadores o características de entidades o relaciones.

Lo más sencillo es preguntarse, para cada entidad y cada relación, ¿qué información se quiere saber de ...? La respuesta a esta pregunta se debe encontrar en las especificaciones de requisitos. Pero, en ocasiones, será necesario preguntar a los usuarios para que aclaren los requisitos. Desgraciadamente, los usuarios pueden dar respuestas a esta pregunta que también contengan otros conceptos, por lo que hay que considerar sus respuestas con mucho cuidado.

Al identificar los atributos, hay que tener en cuenta si son simples o compuestos. Por ejemplo, el atributo *dirección* puede ser simple, teniendo la dirección completa como un solo valor: 'San Rafael 45, Almazora'; o puede ser un atributo compuesto, formado por la *calle* ('San Rafael'), el *número* ('45') y la *población* ('Almazora'). El escoger entre atributo simple o compuesto depende de los requisitos del usuario. Si el usuario no necesita acceder a cada uno de los componentes de la dirección por separado, se puede representar como un atributo simple. Pero si el usuario quiere acceder a los componentes de forma individual, entonces se debe representar como un atributo compuesto.

También se deben identificar los atributos derivados o calculados, que son aquellos cuyo valor se puede calcular a partir de los valores de otros atributos. Por ejemplo, el número de empleados de cada oficina, la edad de los empleados o el número de inmuebles que gestiona cada empleado. Algunos diseñadores no representan los atributos derivados en los esquemas

conceptuales. Si se hace, se debe indicar claramente que el atributo es derivado y a partir de qué atributos se obtiene su valor. Donde hay que considerar los atributos derivados es en el diseño físico.

Cuando se están identificando los atributos, se puede descubrir alguna entidad que no se ha identificado previamente, por lo que hay que volver al principio introduciendo esta entidad y viendo si se relaciona con otras entidades.

Es muy útil elaborar una lista de atributos e ir eliminándolos de la lista conforme se vayan asociando a una entidad o relación. De este modo, uno se puede asegurar de que cada atributo se asocia a una sola entidad o relación, y que cuando la lista se ha acabado, se han asociado todos los atributos.

Hay que tener mucho cuidado cuando parece que un mismo atributo se debe asociar a varias entidades. Esto puede ser por una de las siguientes causas:

- Se han identificado varias entidades, como *director*, *supervisor* y *administrativo*, cuando, de hecho, pueden representarse como una sola entidad denominada *empleado*. En este caso, se puede escoger entre introducir una jerarquía de generalización, o dejar las entidades que representan cada uno de los puestos de empleado.
- Se ha identificado una relación entre entidades. En este caso, se debe asociar el atributo a una sola de las entidades y hay que asegurarse de que la relación ya se había identificado previamente. Si no es así, se debe actualizar la documentación para recoger la nueva relación.

Conforme se van identificando los atributos, se les asignan nombres que tengan significado para el usuario. De cada atributo se debe anotar la siguiente información:

- Nombre y descripción del atributo.
- Alias o sinónimos por los que se conoce al atributo.
- Tipo de dato y longitud.
- Valores por defecto del atributo (si se especifican).

- Si el atributo siempre va a tener un valor (si admite o no nulos).
- Si el atributo es compuesto y, en su caso, qué atributos simples lo forman.
- Si el atributo es derivado y, en su caso, cómo se calcula su valor.
- Si el atributo es multievaluado.

4. Determinar los dominios de los atributos

El dominio de un atributo es el conjunto de valores que puede tomar el atributo. Por ejemplo el dominio de los números de oficina son las tiras de hasta tres caracteres en donde el primero es una letra y el siguiente o los dos siguientes son dígitos en el rango de 1 a 99; el dominio de los números de teléfono y los números de fax son las tiras de 9 dígitos.

Un esquema conceptual está completo si incluye los dominios de cada atributo: los valores permitidos para cada atributo, su tamaño y su formato. También se puede incluir información adicional sobre los dominios como, por ejemplo, las operaciones que se pueden realizar sobre cada atributo, qué atributos pueden compararse entre sí o qué atributos pueden combinarse con otros. Aunque sería muy interesante que el sistema final respetara todas estas indicaciones sobre los dominios, esto es todavía una línea abierta de investigación.

Toda la información sobre los dominios se debe anotar también en el diccionario de datos.

5. Determinar los identificadores

Cada entidad tiene al menos un identificador. En este paso, se trata de encontrar todos los identificadores de cada una de las entidades. Los identificadores pueden ser simples o compuestos. De cada entidad se escogerá uno de los identificadores como clave primaria en la fase del diseño lógico.

Cuando se determinan los identificadores es fácil darse cuenta de si una entidad es fuerte o débil. Si una entidad tiene al menos un identificador, es *fuerte* (otras denominaciones son *padre*, *propietaria* o *dominante*). Si una entidad no tiene atributos que le sirvan de identificador, es *débil* (otras denominaciones son *hijo*, *dependiente* o *subordinada*).

Todos los identificadores de las entidades se deben anotar en el diccionario de datos.

6. Determinar las jerarquías de generalización

En este paso hay que observar las entidades que se han identificado hasta el momento. Hay que ver si es necesario reflejar las diferencias entre distintas ocurrencias de una entidad, con lo que surgirán nuevas subentidades de esta entidad genérica; o bien, si hay entidades que tienen características en común y que realmente son subentidades de una nueva entidad genérica.

En cada jerarquía hay que determinar si es total o parcial y exclusiva o superpuesta.

7. Dibujar el diagrama entidad–relación

Una vez identificados todos los conceptos, se puede dibujar el diagrama entidad–relación correspondiente a una de las vistas de los usuarios. Se obtiene así un esquema conceptual local.

8. Revisar el esquema conceptual local con el usuario

Antes de dar por finalizada la fase del diseño conceptual, se debe revisar el esquema conceptual local con el usuario. Este esquema está formado por el diagrama entidad–relación y toda la documentación que describe el esquema. Si se encuentra alguna anomalía, hay que corregirla haciendo los cambios oportunos, por lo que posiblemente haya que repetir alguno de los pasos anteriores. Este proceso debe repetirse hasta que se esté seguro de que el esquema conceptual es una fiel representación de la parte de la empresa que se está tratando de modelar.

6.6 Resumen

El diseño de bases de datos se descompone en tres etapas: diseño conceptual, diseño lógico y diseño físico. El diseño conceptual es el proceso por el cual se construye un modelo

de la información que se utiliza en una empresa u organización, independientemente del SGBD que se vaya a utilizar para implementar el sistema y de los equipos informáticos o cualquier otra consideración física.

Un modelo conceptual es un conjunto de conceptos que permiten describir la realidad mediante representaciones lingüísticas y gráficas. Los modelos conceptuales deben poseer una serie de propiedades: expresividad, simplicidad, minimalidad y formalidad.

El modelo conceptual más utilizado es el modelo entidad–relación, que posee los siguientes conceptos: entidades, relaciones, atributos, dominios de atributos, identificadores y jerarquías de generalización.

En la metodología del diseño conceptual se construye un esquema conceptual local para cada vista de cada usuario o grupo de usuarios. En el diseño lógico se obtiene un esquema lógico local para cada esquema conceptual local. Estos esquemas lógicos se integran después para formar un esquema lógico global que represente todas las vistas de los distintos usuarios de la empresa. Por último, en el diseño físico, se construye la implementación de la base de datos sobre un SGBD determinado. Ya que este diseño debe adaptarse al SGBD, es posible que haya que introducir cambios en el esquema lógico para mejorar las prestaciones a nivel físico.

Cada vista de usuario comprende los datos que un usuario maneja para llevar a cabo una determinada tarea. Normalmente, estas vistas corresponden a las distintas áreas funcionales de la empresa, y se pueden identificar examinando los diagramas de flujo de datos o entrevistando a los usuarios, examinando los procedimientos, informes y formularios, y observando el funcionamiento de la empresa.

Cada esquema conceptual local está formado por entidades, relaciones, atributos, dominios de atributos, identificadores y puede haber también jerarquías de generalización. Además, estos esquemas se completan documentándolos en el diccionario de datos.

Bibliografía

Los aspectos sobre el modelo entidad–relación en donde se incorporan características del modelo entidad–relación extendido, como las jerarquías de generalización, se tratan muy bien en los textos de Batini, Ceri y Navathe (1994) y Connolly, Begg y Strachan (1996). Estos últimos son los que mejor presentan la metodología del diseño conceptual, proporcionando todo tipo de "trucos" para identificar cada uno de los componentes de un esquema conceptual.

Capítulo 7

Diseño lógico de bases de datos

En este capítulo se describen los pasos para llevar a cabo el diseño lógico. Ya que aquí se trata el diseño de bases de datos relacionales, en esta etapa se obtiene un conjunto de relaciones (tablas) que representen los datos de interés. Este conjunto de relaciones se valida mediante la normalización, técnica que se estudia al final del capítulo.

7.1 Introducción

El objetivo del diseño lógico es convertir los esquemas conceptuales locales en un esquema lógico global que se ajuste al modelo de SGBD sobre el que se vaya a implementar el sistema. Mientras que el objetivo fundamental del diseño conceptual es la compleción y expresividad de los esquemas conceptuales locales, el objetivo del diseño lógico es obtener una representación que use, del modo más eficiente posible, los recursos que el modelo de SGBD posee para estructurar los datos y para modelar las restricciones

Los modelos de bases de datos más extendidos son el modelo relacional, el modelo de red y el modelo jerárquico. El modelo orientado a objetos es también muy popular, pero no existe un modelo estándar orientado a objetos.

El modelo relacional (y los modelos previos) carecen de ciertos rasgos de abstracción que se usan en los modelos conceptuales. Por lo tanto, un primer paso en la fase del diseño lógico

consistirá en la conversión de esos mecanismos de representación de alto nivel en términos de las estructuras de bajo nivel disponibles en el modelo relacional.

7.2 Metodología de diseño lógico en el modelo relacional

La metodología que se va a seguir para el diseño lógico en el modelo relacional consta de dos fases, cada una de ellas compuesta por varios pasos que se detallan a continuación.

- Construir y validar los esquemas lógicos locales para cada vista de usuario.
  1. Convertir los esquemas conceptuales locales en esquemas lógicos locales.
  2. Derivar un conjunto de relaciones (tablas) para cada esquema lógico local.
  3. Validar cada esquema mediante la normalización.
  4. Validar cada esquema frente a las transacciones del usuario.
  5. Dibujar el diagrama entidad–relación.
  6. Definir las restricciones de integridad.
  7. Revisar cada esquema lógico local con el usuario correspondiente.
- Construir y validar el esquema lógico global.
  8. Mezclar los esquemas lógicos locales en un esquema lógico global.
  9. Validar el esquema lógico global.
  10. Estudiar el crecimiento futuro.
  11. Dibujar el diagrama entidad–relación final.
  12. Revisar el esquema lógico global con los usuarios.

En la primera fase, se construyen los esquemas lógicos locales para cada vista de usuario y se validan. En esta fase se refinan los esquemas conceptuales creados durante el diseño conceptual, eliminando las estructuras de datos que no se pueden implementar de manera directa sobre el modelo que soporta el SGBD, en el caso que nos ocupa, el modelo relacional. Una vez hecho esto, se obtiene un primer esquema lógico que se valida mediante la normalización y frente a las transacciones que el sistema debe llevar a cabo, tal y como se refleja en

las especificaciones de requisitos de usuario. El esquema lógico ya validado se puede utilizar como base para el desarrollo de prototipos. Una vez finalizada esta fase, se dispone de un esquema lógico para cada vista de usuario que es correcto, comprensible y sin ambigüedad.

1. Convertir los esquemas conceptuales locales en esquemas lógicos locales

En este paso, se eliminan de cada esquema conceptual las estructuras de datos que los sistemas relacionales no modelan directamente:

(a) *Eliminar las relaciones de muchos a muchos*, sustituyendo cada una de ellas por una nueva entidad intermedia y dos relaciones de uno a muchos de esta nueva entidad con las entidades originales. La nueva entidad será débil, ya que sus ocurrencias dependen de la existencia de ocurrencias en las entidades originales.

(b) *Eliminar las relaciones entre tres o más entidades*, sustituyendo cada una de ellas por una nueva entidad (débil) intermedia que se relaciona con cada una de las entidades originales. La cardinalidad de estas nuevas relaciones binarias dependerá de su significado.

(c) *Eliminar las relaciones recursivas*, sustituyendo cada una de ellas por una nueva entidad (débil) y dos relaciones binarias de esta nueva entidad con la entidad original. La cardinalidad de estas relaciones dependerá de su significado.

(d) *Eliminar las relaciones con atributos*, sustituyendo cada una de ellas por una nueva entidad (débil) y las relaciones binarias correspondientes de esta nueva entidad con las entidades originales. La cardinalidad de estas relaciones dependerá del tipo de la relación original y de su significado.

(e) *Eliminar los atributos multievaluados*, sustituyendo cada uno de ellos por una nueva entidad (débil) y una relación binaria de uno a muchos con la entidad original.

(f) *Revisar las relaciones de uno a uno*, ya que es posible que se hayan identificado dos entidades que representen el mismo objeto (sinónimos). Si así fuera, ambas entidades deben integrarse en una sola.

(g) *Eliminar las relaciones redundantes.* Una relación es redundante cuando se puede obtener la misma información que ella aporta mediante otras relaciones. El hecho de que haya dos caminos diferentes entre dos entidades no implica que uno de los caminos corresponda a una relación redundante, eso dependerá del significado de cada relación.

Una vez finalizado este paso, es más correcto referirse a los esquemas conceptuales locales refinados como esquemas lógicos locales, ya que se adaptan al modelo de base de datos que soporta el SGBD escogido.

2. Derivar un conjunto de relaciones (tablas) para cada esquema lógico local

En este paso, se obtiene un conjunto de relaciones (tablas) para cada uno de los esquemas lógicos locales en donde se representen las entidades y relaciones entre entidades, que se describen en cada una de las vistas que los usuarios tienen de la empresa. Cada relación de la base de datos tendrá un nombre, y el nombre de sus atributos aparecerá, a continuación, entre paréntesis. El atributo o atributos que forman la clave primaria se subrayan. Las claves ajenas, mecanismo que se utiliza para representar las relaciones entre entidades en el modelo relacional, se especifican aparte indicando la relación (tabla) a la que hacen referencia.

A continuación, se describe cómo las relaciones (tablas) del modelo relacional representan las entidades y relaciones que pueden aparecer en los esquemas lógicos.

(a) *Entidades fuertes.* Crear una relación para cada entidad fuerte que incluya todos sus atributos simples. De los atributos compuestos incluir sólo sus componentes.

Cada uno de los identificadores de la entidad será una clave candidata. De entre las claves candidatas hay que escoger la clave primaria; el resto serán claves alternativas. Para escoger la clave primaria entre las claves candidatas se pueden seguir estas indicaciones:

- Escoger la clave candidata que tenga menos atributos.
- Escoger la clave candidata cuyos valores no tengan probabilidad de cambiar en el futuro.
- Escoger la clave candidata cuyos valores no tengan probabilidad de perder la unicidad en el futuro.

- Escoger la clave candidata con el mínimo número de caracteres (si es de tipo texto).
- Escoger la clave candidata más fácil de utilizar desde el punto de vista de los usuarios.

(b) *Entidades débiles.* Crear una relación para cada entidad débil incluyendo todos sus atributos simples. De los atributos compuestos incluir sólo sus componentes. Añadir una clave ajena a la entidad de la que depende. Para ello, se incluye la clave primaria de la relación que representa a la entidad padre en la nueva relación creada para la entidad débil. A continuación, determinar la clave primaria de la nueva relación.

(c) *Relaciones binarias de uno a uno.* Para cada relación binaria se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. La entidad hijo es la que participa de forma total (obligatoria) en la relación, mientras que la entidad padre es la que participa de forma parcial (opcional). Si las dos entidades participan de forma total o parcial en la relación, la elección de padre e hijo es arbitraria. Además, en caso de que ambas entidades participen de forma total en la relación, se tiene la opción de integrar las dos entidades en una sola relación (tabla). Esto se suele hacer si una de las entidades no participa en ninguna otra relación.

(d) *Relaciones binarias de uno a muchos.* Como en las relaciones de uno a uno, se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. Pero ahora, la entidad padre es la de "la parte del muchos" (cada padre tiene muchos hijos), mientras que la entidad hijo es la de "la parte del uno" (cada hijo tiene un solo padre).

(e) *Jerarquías de generalización.* En las jerarquías, se denomina entidad padre a la entidad genérica y entidades hijo a las subentidades. Hay tres opciones distintas para representar las jerarquías. La elección de la más adecuada se hará en función de su tipo (total/parcial, exclusiva/superpuesta).

1. Crear una relación por cada entidad. Las relaciones de las entidades hijo heredan como clave primaria la de la entidad padre. Por lo tanto, la clave primaria de

las entidades hijo es también una clave ajena al padre. Esta opción sirve para cualquier tipo de jerarquía, total o parcial y exclusiva o superpuesta.

2. Crear una relación por cada entidad hijo, heredando los atributos de la entidad padre. Esta opción sólo sirve para jerarquías totales y exclusivas.
3. Integrar todas las entidades en una relación, incluyendo en ella los atributos de la entidad padre, los atributos de todos los hijos y un atributo discriminativo para indicar el caso al cual pertenece la entidad en consideración. Esta opción sirve para cualquier tipo de jerarquía. Si la jerarquía es superpuesta, el atributo discriminativo será multievaluado.

Una vez obtenidas las relaciones con sus atributos, claves primarias y claves ajenas, sólo queda actualizar el diccionario de datos con los nuevos atributos que se hayan identificado en este paso.

3. Validar cada esquema mediante la normalización

La normalización se utiliza para mejorar el esquema lógico, de modo que satisfaga ciertas restricciones que eviten la duplicidad de datos. La normalización garantiza que el esquema resultante se encuentra más próximo al modelo de la empresa, que es consistente y que tiene la mínima redundancia y la máxima estabilidad.

La normalización es un proceso que permite decidir a qué entidad pertenece cada atributo. Uno de los conceptos básicos del modelo relacional es que los atributos se agrupan en relaciones (tablas) porque están relacionados a nivel lógico. En la mayoría de las ocasiones, una base de datos normalizada no proporciona la máxima eficiencia, sin embargo, el objetivo ahora es conseguir una base de datos normalizada por las siguientes razones:

- Un esquema normalizado organiza los datos de acuerdo a sus dependencias funcionales, es decir, de acuerdo a sus relaciones lógicas.
- El esquema lógico no tiene porqué ser el esquema final. Debe representar lo que el diseñador entiende sobre la naturaleza y el significado de los datos de la empresa. Si se establecen unos objetivos en cuanto a prestaciones, el diseño físico cambiará el esquema

lógico de modo adecuado. Una posibilidad es que algunas relaciones normalizadas se desnormalicen. Pero la desnormalización no implica que se haya malgastado tiempo normalizando, ya que mediante este proceso el diseñador aprende más sobre el significado de los datos. De hecho, la normalización obliga a entender completamente cada uno de los atributos que se han de representar en la base de datos.

- Un esquema normalizado es robusto y carece de redundancias, por lo que está libre de ciertas anomalías que éstas pueden provocar cuando se actualiza la base de datos.
- Los equipos informáticos de hoy en día son mucho más potentes, por lo que en ocasiones es más razonable implementar bases de datos fáciles de manejar (las normalizadas), a costa de un tiempo adicional de proceso.
- La normalización produce bases de datos con esquemas flexibles que pueden extenderse con facilidad.

El objetivo de este paso es obtener un conjunto de relaciones que se encuentren en la forma normal de Boyce–Codd. Para ello, hay que pasar por la primera, segunda y tercera formas normales. El proceso de normalización se describe en el apartado 7.3.

4. Validar cada esquema frente a las transacciones del usuario

El objetivo de este paso es validar cada esquema lógico local para garantizar que puede soportar las transacciones requeridas por los correspondientes usuarios. Estas transacciones se encontrarán en las especificaciones de requisitos de usuario. Lo que se debe hacer es tratar de realizar las transacciones de forma manual utilizando el diagrama entidad–relación, el diccionario de datos y las conexiones que establecen las claves ajenas de las relaciones (tablas). Si todas las transacciones se pueden realizar, el esquema queda validado. Pero si alguna transacción no se puede realizar, seguramente será porque alguna entidad, relación o atributo no se ha incluido en el esquema.

5. Dibujar el diagrama entidad–relación

En este momento, se puede dibujar el diagrama entidad–relación final para cada vista de usuario que recoja la representación lógica de los datos desde su punto de vista. Este

diagrama habrá sido validado mediante la normalización y frente a las transacciones de los usuarios.

6. Definir las restricciones de integridad

Las restricciones de integridad son reglas que se quieren imponer para proteger la base de datos, de modo que no pueda llegar a un estado inconsistente. Hay cinco tipos de restricciones de integridad.

(a) *Datos requeridos.* Algunos atributos deben contener valores en todo momento, es decir, no admiten nulos.

(b) *Restricciones de dominios.* Todos los atributos tienen un dominio asociado, que es el conjunto los valores que cada atributo puede tomar.

(c) *Integridad de entidades.* El identificador de una entidad no puede ser nulo, por lo tanto, las claves primarias de las relaciones (tablas) no admiten nulos.

(d) *Integridad referencial.* Una clave ajena enlaza cada tupla de la relación hijo con la tupla de la relación padre que tiene el mismo valor en su clave primaria. La integridad referencial dice que si una clave ajena tiene un valor (si es no nula), ese valor debe ser uno de los valores de la clave primaria a la que referencia. Hay varios aspectos a tener en cuenta sobre las claves ajenas para lograr que se cumpla la integridad referencial.

1. ¿Admite nulos la clave ajena? Cada clave ajena expresa una relación. Si la participación de la entidad hijo en la relación es total, entonces la clave ajena no admite nulos; si es parcial, la clave ajena debe aceptar nulos.
2. ¿Qué hacer cuando se quiere borrar una ocurrencia de la entidad padre que tiene algún hijo? O lo que es lo mismo, ¿qué hacer cuando se quiere borrar una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Hay varias respuestas posibles:
   - *Restringir:* no se pueden borrar tuplas que están siendo referenciadas por otras tuplas.
   - *Propagar:* se borra la tupla deseada y se propaga el borrado a todas las tuplas que le hacen referencia.

- *Anular:* se borra la tupla deseada y todas las referencias que tenía se ponen, automáticamente, a nulo (esta respuesta sólo es válida si la clave ajena acepta nulos).
- *Valor por defecto:* se borra la tupla deseada y todas las referencias toman, automáticamente, el valor por defecto (esta respuesta sólo es válida si se ha especificado un valor por defecto para la clave ajena).
- *No comprobar:* se borra la tupla deseada y no se hace nada para garantizar que se sigue cumpliendo la integridad referencial.

3. ¿Qué hacer cuando se quiere modificar la clave primaria de una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Las respuestas posibles son las mismas que en el caso anterior. Cuando se escoge propagar, se actualiza la clave primaria en la tupla deseada y se propaga el cambio a los valores de clave ajena que le hacían referencia.

(e) *Reglas de negocio.* Cualquier operación que se realice sobre los datos debe cumplir las restricciones que impone el funcionamiento de la empresa.

Todas las restricciones de integridad establecidas en este paso se deben reflejar en el diccionario de datos para que puedan ser tenidas en cuenta durante la fase del diseño físico.

7. Revisar cada esquema lógico local con el usuario correspondiente

Para garantizar que cada esquema lógico local es una fiel representación de la vista del usuario lo que se debe hacer es comprobar con él que lo reflejado en el esquema y en la documentación es correcto y está completo.

Relación entre el esquema lógico y los diagramas de flujo de datos

El esquema lógico refleja la estructura de los datos a almacenar que maneja la empresa. Un diagrama de flujo de datos muestra cómo se mueven los datos en la empresa y los almacenes en donde se guardan. Si se han utilizado diagramas de flujo de datos para modelar las especificaciones de requisitos de usuario, se pueden utilizar para comprobar la consistencia y completitud del esquema lógico desarrollado. Para ello:

- Cada almacén de datos debe corresponder con una o varias entidades completas.
- Los atributos en los flujos de datos deben corresponder a alguna entidad.

Los esquemas lógicos locales obtenidos hasta este momento se integrarán en un solo esquema lógico global en la siguiente fase para modelar los datos de toda la empresa.

8. Mezclar los esquemas lógicos locales en un esquema lógico global

En este paso, se deben integrar todos los esquemas locales en un solo esquema global. En un sistema pequeño, con dos o tres vistas de usuario y unas pocas entidades y relaciones, es relativamente sencillo comparar los esquemas locales, mezclarlos y resolver cualquier tipo de diferencia que pueda existir. Pero en los sistemas grandes, se debe seguir un proceso más sistemático para llevar a cabo este paso con éxito:

1. Revisar los nombres de las entidades y sus claves primarias.
2. Revisar los nombres de las relaciones.
3. Mezclar las entidades de las vistas locales.
4. Incluir (sin mezclar) las entidades que pertenecen a una sola vista de usuario.
5. Mezclar las relaciones de las vistas locales.
6. Incluir (sin mezclar) las relaciones que pertenecen a una sola vista de usuario.
7. Comprobar que no se ha omitido ninguna entidad ni relación.
8. Comprobar las claves ajenas.
9. Comprobar las restricciones de integridad.
10. Dibujar el esquema lógico global.
11. Actualizar la documentación.

9. Validar el esquema lógico global

Este proceso de validación se realiza, de nuevo, mediante la normalización y mediante la prueba frente a las transacciones de los usuarios. Pero ahora sólo hay que normalizar las relaciones que hayan cambiado al mezclar los esquemas lógicos locales y sólo hay que probar las transacciones que requieran acceso a áreas que hayan sufrido algún cambio.

10. Estudiar el crecimiento futuro

En este paso, se trata de comprobar que el esquema obtenido puede acomodar los futuros cambios en los requisitos con un impacto mínimo. Si el esquema lógico se puede extender fácilmente, cualquiera de los cambios previstos se podrá incorporar al mismo con un efecto mínimo sobre los usuarios existentes.

11. Dibujar el diagrama entidad–relación final

Una vez validado el esquema lógico global, ya se puede dibujar el diagrama entidad–relación que representa el modelo de los datos de la empresa que son de interés. La documentación que describe este modelo (incluyendo el esquema relacional y el diccionario de datos) se debe actualizar y completar.

12. Revisar el esquema lógico global con los usuarios

Una vez más, se debe revisar con los usuarios el esquema global y la documentación obtenida para asegurarse de que son una fiel representación de la empresa.

7.3 Normalización

La normalización es una técnica para diseñar la estructura lógica de los datos de un sistema de información en el modelo relacional, desarrollada por E. F. Codd en 1972. Es una estrategia de diseño de abajo a arriba: se parte de los atributos y éstos se van agrupando en

relaciones (tablas) según su afinidad. Aquí no se utilizará la normalización como una técnica de diseño de bases de datos, sino como una etapa posterior a la correspondencia entre el esquema conceptual y el esquema lógico, que elimine las dependencias entre atributos no deseadas. Las ventajas de la normalización son las siguientes:

- Evita anomalías en inserciones, modificaciones y borrados.
- Mejora la independencia de datos.
- No establece restricciones artificiales en la estructura de los datos.

Uno de los conceptos fundamentales en la normalización es el de *dependencia funcional*. Una dependencia funcional es una relación entre atributos de una misma relación (tabla). Si x e y son atributos de la relación R, se dice que y es funcionalmente dependiente de x (se denota por $x \longrightarrow y$) si cada valor de x tiene asociado un solo valor de y (x e y pueden constar de uno o varios atributos). A x se le denomina *determinante*, ya que x determina el valor de y. Se dice que el atributo y es *completamente dependiente* de x si depende funcionalmente de x y no depende de ningún subconjunto de x.

La dependencia funcional es una noción semántica. Si hay o no dependencias funcionales entre atributos no lo determina una serie abstracta de reglas, sino, más bien, los modelos mentales del usuario y las reglas de negocio de la organización o empresa para la que se desarrolla el sistema de información. Cada dependencia funcional es una clase especial de regla de integridad y representa una relación de uno a muchos.

En el proceso de normalización se debe ir comprobando que cada relación (tabla) cumple una serie de reglas que se basan en la clave primaria y las dependencias funcionales. Cada regla que se cumple aumenta el grado de normalización. Si una regla no se cumple, la relación se debe descomponer en varias relaciones que sí la cumplan.

La normalización se lleva a cabo en una serie pasos. Cada paso corresponde a una forma normal que tiene unas propiedades. Conforme se va avanzando en la normalización, las relaciones tienen un formato más estricto (más fuerte) y, por lo tanto, son menos vulnerables a las anomalías de actualización. El modelo relacional sólo requiere un conjunto de relaciones en primera forma normal. Las restantes formas normales son opcionales. Sin embargo, para

evitar las anomalías de actualización, es recomendable llegar al menos a la tercera forma normal.

Primera forma normal (1FN)

Una relación está en primera forma normal si, y sólo si, todos los dominios de la misma contienen valores atómicos, es decir, no hay grupos repetitivos. Si se ve la relación gráficamente como una tabla, estará en 1FN si tiene un solo valor en la intersección de cada fila con cada columna.

Si una relación no está en 1FN, hay que eliminar de ella los grupos repetitivos. Un grupo repetitivo será el atributo o grupo de atributos que tiene múltiples valores para cada tupla de la relación. Hay dos formas de eliminar los grupos repetitivos. En la primera, se repiten los atributos con un solo valor para cada valor del grupo repetitivo. De este modo, se introducen redundancias ya que se duplican valores, pero estas redundancias se eliminarán después mediante las restantes formas normales. La segunda forma de eliminar los grupos repetitivos consiste en poner cada uno de ellos en una relación aparte, heredando la clave primaria de la relación en la que se encontraban.

Un conjunto de relaciones se encuentra en 1FN si ninguna de ellas tiene grupos repetitivos.

Segunda forma normal (2FN)

Una relación está en segunda forma normal si, y sólo si, está en 1FN y, además, cada atributo que no está en la clave primaria es completamente dependiente de la clave primaria.

La 2FN se aplica a las relaciones que tienen claves primarias compuestas por dos o más atributos. Si una relación está en 1FN y su clave primaria es simple (tiene un solo atributo), entonces también está en 2FN. Las relaciones que no están en 2FN pueden sufrir anomalías cuando se realizan actualizaciones.

Para pasar una relación en 1FN a 2FN hay que eliminar las dependencias parciales de la clave primaria. Para ello, se eliminan los atributos que son funcionalmente dependientes y

se ponen en una nueva relación con una copia de su determinante (los atributos de la clave primaria de los que dependen).

Tercera forma normal (3FN)

Una relación está en tercera forma normal si, y sólo si, está en 2FN y, además, cada atributo que no está en la clave primaria no depende transitivamente de la clave primaria. La dependencia $x \longrightarrow z$ es transitiva si existen las dependencias $x \longrightarrow y$, $y \longrightarrow z$, siendo x, y, z atributos o conjuntos de atributos de una misma relación.

Aunque las relaciones en 2FN tienen menos redundancias que las relaciones en 1FN, todavía pueden sufrir anomalías frente a las actualizaciones. Para pasar una relación de 2FN a 3FN hay que eliminar las dependencias transitivas. Para ello, se eliminan los atributos que dependen transitivamente y se ponen en una nueva relación con una copia de su determinante (el atributo o atributos no clave de los que dependen).

Forma normal de Boyce–Codd (BCFN)

Una relación está en la forma normal de Boyce–Codd si, y sólo si, todo determinante es una clave candidata.

La 2FN y la 3FN eliminan las dependencias parciales y las dependencias transitivas de la clave primaria. Pero este tipo de dependencias todavía pueden existir sobre otras claves candidatas, si éstas existen. La BCFN es más fuerte que la 3FN, por lo tanto, toda relación en BCFN está en 3FN.

La violación de la BCFN es poco frecuente ya que se da bajo ciertas condiciones que raramente se presentan. Se debe comprobar si una relación viola la BCFN si tiene dos o más claves candidatas compuestas que tienen al menos un atributo en común.

7.4 Resumen

El diseño de bases de datos consta de tres etapas: diseño conceptual, lógico y físico. El diseño lógico es el proceso mediante el que se construye un esquema que representa la

información que maneja una empresa, basándose en un modelo lógico determinado, pero independientemente del SGBD concreto que se vaya a utilizar para implementar la base de datos e independientemente de cualquier otra consideración física.

Las dos fases de que consta el diseño lógico son la construcción y validación de los esquemas lógicos locales para cada vista de usuario, y la construcción y validación de un esquema lógico global. Cada una de estas fases consta de una serie de pasos.

Un paso importante es la conversión del esquema conceptual a un esquema lógico adecuado al modelo relacional. Para ello, se deben hacer algunas transformaciones: eliminar las relaciones de muchos a muchos, eliminar las relaciones complejas, eliminar las relaciones recursivas, eliminar las relaciones con atributos, eliminar los atributos multievaluados, reconsiderar las relaciones de uno a uno y eliminar las relaciones redundantes.

Los esquemas lógicos se pueden validar mediante la normalización y frente a las transacciones de los usuarios. La normalización se utiliza para mejorar el esquema, de modo que éste satisface ciertas restricciones que evitan la duplicidad de datos. La normalización garantiza que el esquema resultante está más próximo al modelo de la empresa, es consistente, tiene la mínima redundancia y la máxima estabilidad.

Las restricciones de integridad son las restricciones que se imponen para que la base de datos nunca llegue a un estado inconsistente. Hay cinco tipos de restricciones de integridad: datos requeridos, restricciones de dominio, integridad de entidades, integridad referencial y reglas de negocio.

Para garantizar la integridad referencial se debe especificar el comportamiento de las claves ajenas: si aceptan nulos y qué hacer cuando se borra la tupla a la que se hace referencia, o cuando se modifica el valor de su clave primaria.

Bibliografía

El diseño de bases de datos relacionales es un tema de consenso, coincidiendo la mayoría de autores en las tres etapas de diseño conceptual, diseño lógico y diseño físico. Sin embargo, los pasos de que consta cada una de estas etapas y la terminología utilizada no es muy

uniforme. Este capítulo se ha elaborado siguiendo los pasos que se especifican en el texto de Connolly, Begg y Strachan (1996), combinado con la terminología de Batini, Ceri y Navathe (1994).

Capítulo 8

Diseño físico de bases de datos

En este capítulo se describe la metodología de diseño físico para bases de datos relacionales. En esta etapa, se parte del esquema lógico global obtenido durante el diseño lógico y se obtiene una descripción de la implementación de la base de datos en memoria secundaria. Esta descripción es completamente dependiente del SGBD específico que se vaya a utilizar. En este capítulo se dan una serie de directrices para escoger las estructuras de almacenamiento de las relaciones base, decidir cuándo crear índices y cuándo desnormalizar el esquema lógico e introducir redundancias.

8.1 Introducción

El diseño de una base de datos se descompone en tres etapas: diseño conceptual, lógico y físico. La etapa del diseño lógico es independiente de los detalles de implementación y dependiente del tipo de SGBD que se vaya a utilizar. La salida de esta etapa es el esquema lógico global y la documentación que lo describe. Todo ello es la entrada para la etapa que viene a continuación, el diseño físico.

Mientras que en el diseño lógico se especifica qué se guarda, en el diseño físico se especifica cómo se guarda. Para ello, el diseñador debe conocer muy bien toda la funcionalidad del SGBD concreto que se vaya a utilizar y también el sistema informático sobre el que éste va a trabajar. El diseño físico no es una etapa aislada, ya que algunas decisiones que se

tomen durante su desarrollo, por ejemplo para mejorar las prestaciones, pueden provocar una reestructuración del esquema lógico.

8.2 Metodología de diseño físico para bases de datos relacionales

El objetivo de esta etapa es producir una descripción de la implementación de la base de datos en memoria secundaria. Esta descripción incluye las estructuras de almacenamiento y los métodos de acceso que se utilizarán para conseguir un acceso eficiente a los datos.

El diseño físico se divide de cuatro fases, cada una de ellas compuesta por una serie de pasos:

- Traducir el esquema lógico global para el SGBD específico.
  1. Diseñar las relaciones base para el SGBD específico.
  2. Diseñar las reglas de negocio para el SGBD específico.
- Diseñar la representación física.
  3. Analizar las transacciones.
  4. Escoger las organizaciones de ficheros.
  5. Escoger los índices secundarios.
  6. Considerar la introducción de redundancias controladas.
  7. Estimar la necesidad de espacio en disco.
- Diseñar los mecanismos de seguridad.
  8. Diseñar las vistas de los usuarios.
  9. Diseñar las reglas de acceso.
- Monitorizar y afinar el sistema.

8.2.1 Traducir el esquema lógico global

La primera fase del diseño lógico consiste en traducir el esquema lógico global en un esquema que se pueda implementar en el SGBD escogido. Para ello, es necesario conocer toda la funcionalidad que éste ofrece. Por ejemplo, el diseñador deberá saber:

- Si el sistema soporta la definición de claves primarias, claves ajenas y claves alternativas.
- Si el sistema soporta la definición de datos requeridos (es decir, si se pueden definir atributos como no nulos).
- Si el sistema soporta la definición de dominios.
- Si el sistema soporta la definición de reglas de negocio.
- Cómo se crean las relaciones base.

1. Diseñar las relaciones base para el SGBD específico

Las relaciones base se definen mediante el lenguaje de definición de datos del SGBD. Para ello, se utiliza la información producida durante el diseño lógico: el esquema lógico global y el diccionario de datos. El esquema lógico consta de un conjunto de relaciones y, para cada una de ellas, se tiene:

- El nombre de la relación.
- La lista de atributos entre paréntesis.
- La clave primaria y las claves ajenas, si las tiene.
- Las reglas de integridad de las claves ajenas.

En el diccionario de datos se describen los atributos y, para cada uno de ellos, se tiene:

- Su dominio: tipo de datos, longitud y restricciones de dominio.

- El valor por defecto, que es opcional.
- Si admite nulos.
- Si es derivado y, en caso de serlo, cómo se calcula su valor.

A continuación, se muestra un ejemplo de la definición de la relación `INMUEBLE` con el estándar SQL2.

```
CREATE DOMAIN pnum      AS VARCHAR(5);
CREATE DOMAIN enum      AS VARCHAR(5);
CREATE DOMAIN onum      AS VARCHAR(3);
CREATE DOMAIN inum      AS VARCHAR(5);
CREATE DOMAIN calle     AS VARCHAR(25);
CREATE DOMAIN area      AS VARCHAR(15);
CREATE DOMAIN poblacion AS VARCHAR(15);
CREATE DOMAIN tipo      AS VARCHAR(1)
       CHECK(VALUE IN ('A','C','D','P','V'));
CREATE DOMAIN hab       AS SMALLINT
       CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN alquiler  AS DECIMAL(6,2)
       CHECK(VALUE BETWEEN 0 AND 9999);

CREATE TABLE inmueble (
   inum       INUM         NOT NULL,
   calle      CALLE        NOT NULL,
   area       AREA,
   poblacion  POBLACION    NOT NULL,
   tipo       TIPO         NOT NULL DEFAULT 'P',
   hab        HAB          NOT NULL DEFAULT 4,
   alquiler   ALQUILER     NOT NULL DEFAULT 350,
   pnum       PNUM         NOT NULL,
   enum       ENUM,
   onum       ONUM         NOT NULL,
```

```
PRIMARY KEY (inum),
FOREIGN KEY (pnum) REFERENCES propietario
          ON DELETE no action ON UPDATE cascade,
FOREIGN KEY (enum) REFERENCES plantilla
          ON DELETE set null  ON UPDATE cascade,
FOREIGN KEY (onum) REFERENCES oficina
          ON DELETE no action ON UPDATE cascade
);
```

2. Diseñar las reglas de negocio para el SGBD específico

Las actualizaciones que se realizan sobre las relaciones de la base de datos deben observar ciertas restricciones que imponen las reglas de negocio de la empresa. Algunos SGBD proporcionan mecanismos que permiten definir estas restricciones y vigilan que no se violen.

Por ejemplo, si no se quiere que un empleado tenga más de diez inmuebles asignados, se puede definir una restricción en la sentencia `CREATE TABLE` de la relación `INMUEBLE`:

```
CONSTRAINT inmuebles_por_empleado
    CHECK (NOT EXISTS (SELECT enum
                       FROM   inmueble
                       GROUP BY enum
                       HAVING COUNT(*)>10))
```

Otro modo de definir esta restricción es mediante un *disparador* (*trigger*):

```
CREATE TRIGGER inmuebles_por_empleado
ON inmueble
FOR INSERT,UPDATE
AS IF ((SELECT COUNT(*)
        FROM inmueble i
        WHERE i.inum=INSERTED.inum)>10)
      BEGIN
```

```
        PRINT "Este empleado ya tiene 10 inmuebles asignados"
        ROLLBACK TRANSACTION
    END
```

Hay algunas restricciones que no las pueden manejar los SGBD, como por ejemplo 'a las 20:30 del último día laborable de cada año archivar los inmuebles vendidos y borrarlos'. Para estas restricciones habrá que escribir programas de aplicación específicos. Por otro lado, hay SGBD que no permiten la definición de restricciones, por lo que éstas deberán incluirse en los programas de aplicación.

Todas las restricciones que se definan deben estar documentadas. Si hay varias opciones posibles para implementarlas, hay que explicar porqué se ha escogido la opción implementada.

8.2.2 Diseñar la representación física

Uno de los objetivos principales del diseño físico es almacenar los datos de modo eficiente. Para medir la eficiencia hay varios factores que se deben tener en cuenta:

- *Productividad de transacciones.* Es el número de transacciones que se quiere procesar en un intervalo de tiempo.
- *Tiempo de respuesta.* Es el tiempo que tarda en ejecutarse una transacción. Desde el punto de vista del usuario, este tiempo debería ser el mínimo posible.
- *Espacio en disco.* Es la cantidad de espacio en disco que hace falta para los ficheros de la base de datos. Normalmente, el diseñador querrá minimizar este espacio.

Lo que suele suceder, es que todos estos factores no se pueden satisfacer a la vez. Por ejemplo, para conseguir un tiempo de respuesta mínimo, puede ser necesario aumentar la cantidad de datos almacenados, ocupando más espacio en disco. Por lo tanto, el diseñador deberá ir ajustando estos factores para conseguir un equilibrio razonable. El diseño físico inicial no será el definitivo, sino que habrá que ir monitorizándolo para observar sus prestaciones e ir ajustándolo como sea oportuno. Muchos SGBD proporcionan herramientas para monitorizar y afinar el sistema.

Hay algunas estructuras de almacenamiento que son muy eficientes para cargar grandes cantidades de datos en la base de datos, pero no son eficientes para el resto de operaciones, por lo que se puede escoger dicha estructura de almacenamiento para inicializar la base de datos y cambiarla, a continuación, para su posterior operación. Los tipos de organizaciones de ficheros disponibles varían en cada SGBD. Algunos sistemas proporcionan más estructuras de almacenamiento que otros. Es muy importante que el diseñador del esquema físico sepa qué estructuras de almacenamiento le proporciona el SGBD y cómo las utiliza.

Para mejorar las prestaciones, el diseñador del esquema físico debe saber cómo interactúan los dispositivos involucrados y cómo esto afecta a las prestaciones:

- *Memoria principal.* Los accesos a memoria principal son mucho más rápidos que los accesos a memoria secundaria (decenas o centenas de miles de veces más rápidos). Generalmente, cuanta más memoria principal se tenga, más rápidas serán las aplicaciones. Sin embargo, es aconsejable tener al menos un 5% de la memoria disponible, pero no más de un 10%. Si no hay bastante memoria disponible para todos los procesos, el sistema operativo debe transferir páginas a disco para liberar memoria (*paging*). Cuando estas páginas se vuelven a necesitar, hay que volver a traerlas desde el disco (*faltas de página*). A veces, es necesario llevar procesos enteros a disco (*swapping*) para liberar memoria. El hacer estas transferencias con demasiada frecuencia empeora las prestaciones.

- *CPU.* La CPU controla los recursos del sistema y ejecuta los procesos de usuario. El principal objetivo con este dispositivo es lograr que no haya bloqueos de procesos para conseguirla. Si el sistema operativo, o los procesos de los usuarios, hacen muchas demandas de CPU, ésta se convierte en un cuello de botella. Esto suele ocurrir cuando hay muchas faltas de página o se realiza mucho swapping.

- *Entrada/salida a disco.* Los discos tienen una velocidad de entrada/salida. Cuando se requieren datos a una velocidad mayor que ésta, el disco se convierte en un cuello de botella. Dependiendo de cómo se organicen los datos en el disco, se conseguirá reducir la probabilidad de empeorar las prestaciones. Los principios básicos que se deberían seguir para repartir los datos en los discos son los siguientes:

  - Los ficheros del sistema operativo deben estar separados de los ficheros de la base

de datos.

  - Los ficheros de datos deben estar separados de los ficheros de índices
  - Los ficheros con los diarios de operaciones deben estar separados del resto de los ficheros de la base de datos.

- *Red.* La red se convierte en un cuello de botella cuando tiene mucho tráfico y cuando hay muchas colisiones.

Cada uno de estos recursos afecta a los demás, de modo que una mejora en alguno de ellos puede provocar mejoras en otros.

3. Analizar las transacciones

Para realizar un buen diseño físico es necesario conocer las consultas y las transacciones que se van a ejecutar sobre la base de datos. Esto incluye tanto información cualitativa, como cuantitativa. Para cada transacción, hay que especificar:

- La frecuencia con que se va a ejecutar.
- Las relaciones y los atributos a los que accede la transacción, y el tipo de acceso: consulta, inserción, modificación o eliminación. Los atributos que se modifican no son buenos candidatos para construir estructuras de acceso.
- Los atributos que se utilizan en los predicados del `WHERE` de las sentencias SQL. Estos atributos pueden ser candidatos para construir estructuras de acceso dependiendo del tipo de predicado que se utilice.
- Si es una consulta, los atributos involucrados en el join de dos o más relaciones. Estos atributos pueden ser candidatos para construir estructuras de acceso.
- Las restricciones temporales impuestas sobre la transacción. Los atributos utilizados en los predicados de la transacción pueden ser candidatos para construir estructuras de acceso.

4. Escoger las organizaciones de ficheros

El objetivo de este paso es escoger la organización de ficheros óptima para cada relación. Por ejemplo, un fichero desordenado es una buena estructura cuando se va a cargar gran cantidad de datos en una relación al inicializarla, cuando la relación tiene pocas tuplas, también cuando en cada acceso se deben obtener todas las tuplas de la relación, o cuando la relación tiene una estructura de acceso adicional, como puede ser un índice. Por otra parte, los ficheros dispersos (hashing) son apropiados cuando se accede a las tuplas a través de los valores exactos de alguno de sus campos (condición de igualdad en el `WHERE`). Si la condición de búsqueda es distinta de la igualdad (búsqueda por rango, por patrón, etc.), la dispersión no es una buena opción. Hay otras organizaciones, como la ISAM o los árboles B+.

Las organizaciones de ficheros elegidas deben documentarse, justificando en cada caso la opción escogida.

5. Escoger los índices secundarios

Los índices secundarios permiten especificar caminos de acceso adicionales para las relaciones base. Por ejemplo, la relación `INMUEBLE` se puede haber almacenado en un fichero disperso a través del atributo `inum`. Si se accede a menudo a esta relación a través del atributo `alquiler`, se puede plantear la creación de un índice sobre dicho atributo para favorecer estos accesos. Pero hay que tener en cuenta que estos índices conllevan un coste de mantenimiento que hay que sopesar frente a la ganancia en prestaciones. A la hora de seleccionar los índices, se pueden seguir las siguientes indicaciones:

- Construir un índice sobre la clave primaria de cada relación base.
- No crear índices sobre relaciones pequeñas.
- Añadir un índice sobre los atributos que se utilizan para acceder con mucha frecuencia.
- Añadir un índice sobre las claves ajenas que se utilicen con frecuencia para hacer joins.
- Evitar los índices sobre atributos que se modifican a menudo.

- Evitar los índices sobre atributos poco selectivos (aquellos en los que la consulta selecciona una porción significativa de la relación).
- Evitar los índices sobre atributos formados por tiras de caracteres largas.

Los índices creados se deben documentar, explicando las razones de su elección.

6. Considerar la introducción de redundancias controladas

En ocasiones puede ser conveniente relajar las reglas de normalización introduciendo redundancias de forma controlada, con objeto de mejorar las prestaciones del sistema. En la etapa del diseño lógico se recomienda llegar, al menos, hasta la tercera forma normal para obtener un esquema con una estructura consistente y sin redundancias. Pero, a menudo, sucede que las bases de datos así normalizadas no proporcionan la máxima eficiencia, con lo que es necesario volver atrás y desnormalizar algunas relaciones, sacrificando los beneficios de la normalización para mejorar las prestaciones. Es importante hacer notar que la desnormalización sólo debe realizarse cuando se estime que el sistema no puede alcanzar las prestaciones deseadas. Y, desde luego, la necesidad de desnormalizar en ocasiones no implica eliminar la normalización del diseño lógico: la normalización obliga al diseñador a entender completamente cada uno de los atributos que se han de representar en la base de datos. Por lo tanto, hay que tener en cuenta los siguientes factores:

- La desnormalización hace que la implementación sea más compleja.
- La desnormalización hace que se sacrifique la flexibilidad.
- La desnormalización puede hacer que los accesos a datos sean más rápidos, pero ralentiza las actualizaciones.

Por regla general, la desnormalización de una relación puede ser una opción viable cuando las prestaciones que se obtienen no son las deseadas y la relación se actualiza con poca frecuencia, pero se consulta muy a menudo. Las redundancias que se pueden incluir al desnormalizar son de varios tipos: se pueden introducir datos derivados (calculados a partir de otros datos), se pueden duplicar atributos o se pueden hacer joins de relaciones.

El incluir un atributo derivado dependerá del coste adicional de almacenarlo y mantenerlo consistente con los datos de los que se deriva, frente al coste de calcularlo cada vez que se necesita.

No se pueden establecer una serie de reglas que determinen cuándo desnormalizar relaciones, pero hay algunas situaciones muy comunes en donde puede considerarse esta posibilidad:

- *Combinar relaciones de uno a uno.* Cuando hay relaciones (tablas) involucradas en relaciones de uno a uno, se accede a ellas de manera conjunta con frecuencia y casi no se les accede separadamente, se pueden combinar en una sola relación (tabla).

- *Duplicar atributos no clave en relaciones de uno a muchos para reducir los joins.* Para evitar operaciones de join, se pueden incluir atributos de la relación (tabla) padre en la relación (tabla) hijo de las relaciones de uno a muchos.

- *Tablas de referencia.* Las tablas de referencia (*lookup*) son listas de valores, cada uno de los cuales tiene un código. Por ejemplo puede haber una tabla de referencia para los tipos de inmueble, con las descripciones de estos tipos y un código asociado. Este tipo de tablas son un caso de relación de uno a muchos. En la relación `INMUEBLE` habrá una clave ajena a esta tabla para indicar el tipo de inmueble. De este modo, es muy fácil validar los datos, además de que se ahorra espacio escribiendo sólo el código y no la descripción para cada inmueble, además de ahorrar tiempo cuando se actualizan las descripciones. Si las tablas de referencia se utilizan a menudo en consultas críticas, se puede considerar la introducción de la descripción junto con el código en la relación (tabla) hijo, manteniendo la tabla de referencia para validación de datos.

- *Duplicar claves ajenas en relaciones de uno a muchos para reducir los joins.* Para evitar operaciones de join, se pueden incluir claves ajenas de una relación (tabla) en otra relación (tabla) con la que se relaciona (habrá que tener en cuenta ciertas restricciones).

- *Duplicar atributos en relaciones de muchos a muchos para reducir los joins.* Durante el diseño lógico se eliminan las relaciones de muchos a muchos introduciendo dos relaciones de uno a muchos. Esto hace que aparezca una nueva relación (tabla) intermedia, de modo que si se quiere obtener la información de la relación de muchos a muchos, se

tiene que realizar el join de tres relaciones (tablas). Para evitar algunos de estos joins se pueden incluir algunos de los atributos de las relaciones (tablas) originales en la relación (tabla) intermedia.

- *Introducir grupos repetitivos.* Los grupos repetitivos se eliminan en el primer paso de la normalización para conseguir la primera forma normal. Estos grupos se eliminan introduciendo una nueva relación (tabla), generando una relación de uno a muchos. A veces, puede ser conveniente reintroducir los grupos repetitivos para mejorar las prestaciones.

Todas las redundancias que se introduzcan en este paso se deben documentar y razonar. El esquema lógico se debe actualizar para reflejar los cambios introducidos.

7. Estimar la necesidad de espacio en disco

En caso de que se tenga que adquirir nuevo equipamiento informático, el diseñador debe estimar el espacio necesario en disco para la base de datos. Esta estimación depende del SGBD que se vaya a utilizar y del hardware. En general, se debe estimar el número de tuplas de cada relación y su tamaño. También se debe estimar el factor de crecimiento de cada relación.

8.2.3 Diseñar los mecanismos de seguridad

Los datos constituyen un recurso esencial para la empresa, por lo tanto su seguridad es de vital importancia. Durante el diseño lógico se habrán especificado los requerimientos en cuanto a seguridad que en esta fase se deben implementar. Para llevar a cabo esta implementación, el diseñador debe conocer las posibilidades que ofrece el SGBD que se vaya a utilizar.

8. Diseñar las vistas de los usuarios

El objetivo de este paso es diseñar las vistas de los usuarios correspondientes a los esquemas lógicos locales. Las vistas, además de preservar la seguridad, mejoran la independencia

de datos, reducen la complejidad y permiten que los usuarios vean los datos en el formato deseado.

9. Diseñar las reglas de acceso

El administrador de la base de datos asigna a cada usuario un identificador que tendrá una palabra secreta asociada por motivos de seguridad. Para cada usuario o grupo de usuarios se otorgarán permisos para realizar determinadas acciones sobre determinados objetos de la base de datos. Por ejemplo, los usuarios de un determinado grupo pueden tener permiso para consultar los datos de una relación base concreta y no tener permiso para actualizarlos.

8.2.4 Monitorizar y afinar el sistema

Una vez implementado el esquema físico de la base de datos, se debe poner en marcha para observar sus prestaciones. Si éstas no son las deseadas, el esquema deberá cambiar para intentar satisfacerlas. Una vez afinado el esquema, no permanecerá estático, ya que tendrá que ir cambiando conforme lo requieran los nuevos requisitos de los usuarios. Los SGBD proporcionan herramientas para monitorizar el sistema mientras está en funcionamiento.

8.3 Resumen

El diseño físico es el proceso de producir una descripción de la implementación de la base de datos en memoria secundaria. Describe las relaciones base y las estructuras de almacenamiento y métodos de acceso que se utilizarán para acceder a los datos de modo eficiente. El diseño de las relaciones base sólo se puede realizar cuando el diseñador conoce perfectamente toda la funcionalidad que presenta el SGBD que se vaya a utilizar.

El primer paso consiste en traducir el esquema lógico global de modo que pueda ser fácilmente implementado por el SGBD específico. A continuación, se escogen las organizaciones de ficheros más apropiadas para almacenar las relaciones base, y los métodos de acceso, basándose en el análisis de las transacciones que se van a ejecutar sobre la base de

datos. Se puede considerar la introducción de redundancias controladas para mejorar las prestaciones. Otra tarea a realizar en este paso es estimar el espacio en disco.

La seguridad de la base de datos es fundamental, por lo que el siguiente paso consiste en diseñar las medidas de seguridad necesarias mediante la creación de vistas y el establecimiento de permisos para los usuarios.

El último paso del diseño físico consiste en monitorizar y afinar el sistema para obtener las mejores prestaciones y satisfacer los cambios que se puedan producir en los requisitos.

Bibliografía

El diseño físico es un tema que no se suele tratar en los textos básicos sobre bases de datos. Este capítulo se ha elaborado a partir del texto de Connolly, Begg y Strachan (1996) en el que además, se presenta un ejemplo ilustrativo.

Bibliografía

[1] C. Batini, S. Ceri, S.B. Navathe (1994)
Diseño Conceptual de Bases de Datos. Un enfoque de entidades–interrelaciones
Addison–Wesley / Díaz de Santos

[2] T. Connolly, C. Begg, A. Strachan (1996)
Database Systems. A Practical Approach to Design, Implementation and Management
Addison–Wesley
Segunda Edición en 1998.

[3] C.J. Date (1993)
Introducción a los Sistemas de Bases de Datos
Volumen I, Quinta Edición
Addison–Wesley Iberoamericana
Sexta Edición en 1995 (en inglés, por Addison–Wesley)

[4] R. Elmasri, S.B. Navathe (1997)
Sistemas de Bases de Datos. Conceptos fundamentales
Segunda Edición
Addison–Wesley Iberoamericana
Tercera Edición en 1999 (en inglés, por Addison–Wesley)

[5] M.J. Folk, B. Zoellick (1992)
File Structures
Segunda Edición
Addison–Wesley

[6] G.W. Hansen, J.V. Hansen (1997)
Diseño y Administración de Bases de Datos
Segunda Edición
Prentice Hall

[7] M.J. Hernández (1997)
Database Design for Mere Mortals
Addison–Wesley Developers Press