Capítulo 2

José Ramón Paramá Gabía

Índice general

Capítulo 2

Introducción a los sistemas de bases de datos

Una base de datos (BD) es una colección de datos relacionados. Por datos queremos decir hechos conocidos que pueden registrarse y que tienen un significado implícito.

Según la definición anterior, el catálogo manual (basado en fichas) de una biblioteca es una base de datos, sin embargo, aquí nos referimos a un tipo más específico de bases de datos. Una definición más precisa de lo que es una base de datos podría ser: una colección compartida de datos lógicamente relacionados, y una descripción de estos datos, diseñados para cumplir con las necesidades de información de una organización.

Estas definiciones, como se puede observar, no concuerdan con el significado que coloquialmente se le asigna al término *base de datos*, que en muchos casos se utiliza para referirse en realidad a *sistemas de gestión de bases de datos* (SGBD). Un SGBD es una colección de programas que permiten a los usuarios crear y mantener una base de datos. El SGBD es por tanto un *sistema software de propósito general* que facilita los procesos de *definición, construcción y manipulación* de bases de datos para distintas aplicaciones. La *definición* de una base de datos consiste en especificar los tipos de datos, las estructuras y restricciones de los datos que se van almacenar. La *construcción* de la base de datos es el proceso de almacenar los datos concretos sobre algún medio de almacenamiento controlado por el SGBD. La *manipulación* de la base de datos incluye funciones tales como consultar la bases de datos para recuperar unos datos específicos, actualizar la base de datos para reflejar los cambios ocurridos y generar informes a partir de los datos. Además, los sistemas de bases de datos deben proporcionar la fiabilidad de los datos almacenados, a pesar de las caídas del sistema o los intentos de acceso sin autorización. Si los datos van a ser compartidos entre diversos usuarios, el sistema debe evitar posibles resultados anómalos.

Denominaremos *sistema de bases de datos* al conjunto formado por la base de datos más el SGBD.

2.1. Los usuarios se sistemas de bases de datos

En una pequeña base de datos personal, lo normal es que una sola persona defina, construya y manipule la base de datos. En cambio, en el caso de una base de datos grande, muchas personas participan en su diseño, utilización y mantenimiento. En esta sección identificaremos a las personas cuyo trabajo requiere el empleo cotidiano de una base de datos

grande.

2.1.1. Administradores de bases de datos

En cualquier organización en la que muchas personas utilizan los mismos recursos se requiere un administrador jefe que supervise y gestione dichos recursos. En un entorno de bases de datos, el recurso primario es la propia base de datos, y el secundario es el SGBD y el software relacionado con él. La administración de estos recursos es responsabilidad del administrador de la base de datos (ABD). El ABD se encarga de autorizar el acceso a la base de datos, de coordinar y vigilar su utilización y de adquirir los recursos de software y hardware que sean necesarios. El ABD es la persona responsable cuando surgen problemas como violaciones de la seguridad o una respuesta lenta del sistema. En las organizaciones grandes, el ABD cuenta con la ayuda de personal para poder desempeñar estas funciones.

2.1.2. Diseñadores de bases de datos

Los diseñadores de bases de datos se encargan de identificar los datos que se almacenarán en la base de datos y de elegir las estructuras apropiadas para representar y almacenar dichos datos. Por lo general, estas tareas se realizan antes de que se implemente la base de datos y se carguen los datos. Los diseñadores tienen la responsabilidad de comunicarse con todos los futuros usuarios de la base de datos con el fin de comprender sus necesidades, y de presentar un diseño que satisfaga esos requerimientos.

2.1.3. Usuarios finales

Los usuarios finales son las personas cuyo trabajo requiere acceder a la base de datos para consultarla, actualizarla y generar informes; la base de datos existe principalmente para que ellos la utilicen. Hay varias categorías de usuarios finales:

- Los usuarios finales *ocasionales* acceden de vez en cuando a la base de datos, pero es posible que requieran información diferente en cada ocasión. Utilizan un lenguaje de consulta de base de datos avanzado para especificar sus solicitudes. Interactúan directamente con el SGBD con una aplicación del propio sistema que permite introducir sentencias del lenguaje de consulta y ver su resultado.

- Los usuarios finales *simples o paramétricos* constituyen una porción apreciable de la totalidad de los usuarios finales. La función principal de su trabajo gira en torno a consultas y actualizaciones constantes de la base de datos, utilizando tipos estándar de consultas y actualizaciones, llamadas *transacciones programadas*, que se han programado y probado con mucho cuidado.

  Generalmente las transacciones programadas están incluidas en algún programa de aplicación escrito previamente, que suele tener una interfaz basada en formularios donde los usuarios introducen valores en los campos apropiados.

- Los usuarios finales *avanzados* son aquellos suficientemente familiarizados con los recursos del SGBD como para implementar sus propias aplicaciones.

2.1.4. Analistas de sistemas y programadores de aplicaciones

Los analistas de sistemas determinan los requisitos de los usuarios finales, sobre todo los de los simples o paramétricos y desarrollan especificaciones para transacciones programadas que satisfagan dichos requisitos. Los programadores de aplicaciones implementan esas especificaciones en forma de programas, y luego prueban, depuran, documentan y mantienen estas transacciones programadas. Para realizar dichas tareas, los responsables de las mismas deben conocer a la perfección toda la gama de capacidades del SGBD.

2.2. SGBD frente a sistemas de ficheros

En los capítulos anteriores se ha realizado un repaso de la tecnología disponible en sistemas de ficheros. Se ha mostrado que las posibilidades de almacenamiento y acceso a los datos almacenados por medio de sistemas de ficheros son amplias y variadas. Entonces surge la pregunta, ¿por qué surgen y para que sirven los SGBD?

Para ilustrar lo comentado anteriormente, considérese el ejemplo de un banco que mantiene información acerca de todos los clientes y cuentas de ahorro. Una manera de almacenar esta información es utilizar archivos tal y como vimos en los capítulos anteriores. Para permitir a los usuarios manipular la información, el sistema tiene un número de programas de aplicación que manipula los archivos, incluyendo:

- Un programa para efectuar cargos o abonos en una cuenta.
- Un programa para añadir una cuenta nueva.
- Un programa para calcular el saldo de una cuenta.
- Un programa para generar las operaciones mensuales.

Estos programas de aplicación se han escrito por programadores de sistemas en respuesta a las necesidades de la entidad bancaria. Si las necesidades se incrementan, se añaden nuevos programas de aplicación al sistema. Por ejemplo, supóngase que las regulaciones de un nuevo gobierno permiten crear cuentas de ahorro vivienda con ventajas fiscales especiales. Como resultado se crean nuevos archivos permanentes que contengan la información acerca de todas las cuentas de ahorro vivienda mantenidas por el banco, y puede ser necesario escribir nuevos programas de aplicación para tratar situaciones que no existían con las cuentas de ahorro normales, tales como realizar retenciones especiales, sobre los intereses generados, por parte de hacienda. Así, sobre la marcha, se añaden más archivos y programas de aplicación al sistema.

Este *sistema de procesamiento de archivos* típico que se acaba de describir se mantiene mediante un sistema operativo convencional. Los registros permanentes son almacenados en varios archivos y se escriben diferentes programas de aplicación para extraer registros y para añadir registros a los archivos adecuados. Antes de la llegada de los SGBD, las organizaciones normalmente han almacenado la información usando tales sistemas.

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

- **Redundancia e inconsistencia de los datos**. Debido a que los archivos y programas de aplicación son creados por diferentes programadores en un largo período de tiempo,

los diversos archivos tienen probablemente diferentes formatos y los programas pueden estar escritos en diferentes lenguajes. Más aún, la misma información puede estar duplicada en diferentes lugares (archivos). Por ejemplo, la dirección y número de teléfono de un cliente particular puede aparecer en un archivo que contenga registros de cuentas de ahorro normales y en un archivo que contenga registros de cuentas de ahorro vivienda. Esta redundancia conduce a un almacenamiento y coste de acceso más altos. Además, puede conducir a la *inconsistencia de datos*, es decir, las diversas copias de los mismos datos pueden no coincidir. Por ejemplo, un cambio en la dirección del cliente puede reflejarse en los registros de la cuentas de ahorro normal, pero no en las de ahorro vivienda.

- **Dificultad en el acceso a los datos**. Supóngase que uno de los empleados del banco necesita averiguar los nombres de todos los clientes que viven en el código postal 15500. El empleado le pide al departamento de proceso de datos que genere dicha lista. Debido a que esa petición no fue prevista cuando el sistema original fue diseñado, no hay ningún programa que la realice. Hay, sin embargo, un programa de aplicación que genera la lista de *todos* los clientes. El empleado tiene ahora dos opciones: o bien, a partir de la lista de todos los clientes generada por la aplicación antes comentada, obtener manualmente la información, o bien pedir al departamento de proceso de datos que haga un programa de aplicación para tal fin. Ambas alternativas son obviamente insatisfactorias. Supóngase que se escribe tal programa y que, varios días más tarde, el mismo empleado necesita arreglar esa lista para incluir sólo aquellos clientes que tienen una cuenta con saldo superior a 10.000 €. Como se puede esperar, un programa para generar tal lista no existe. De nuevo, el empleado tiene que elegir entre dos opciones, ninguna de las cuales es satisfactoria.

  La cuestión aquí es que el entorno de procesamiento de archivos convencional no permite que los datos necesarios sean obtenidos de una forma práctica y eficiente. Se deben desarrollar sistemas de recuperación de datos más interesantes para un uso general.

- **Aislamiento de datos**. Debido a que los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos apropiados.

- **Problemas de seguridad**. No todos los usuarios de un sistema de bases de datos deberían poder acceder a todos los datos. Por ejemplo, en un sistema bancario, el personal de nóminas necesita ver sólo la parte de la base de datos concerniente a ese campo. Por ejemplo, no necesitan tener acceso a las cuentas de los clientes. Como los programas de aplicación se añaden al sistema de una forma ad hoc, es difícil garantizar tales restricciones de seguridad.

- **Problemas de integridad**. Los valores de los datos almacenados en la base de datos deben satisfacer ciertos tipos de *restricciones de consistencia*. Por ejemplo, el saldo de una cuenta bancaria no puede nunca ser más bajo de una cantidad predeterminada (por ejemplo 25 €). Los desarrolladores hacen cumplir esas restricciones en el sistema añadiendo el código apropiado en los diversos programas de aplicación. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan. El problema es complicado cuando las restricciones implican diferentes elementos de datos de diferentes archivos.

- **Problemas de atomicidad**. Un sistema de un computador, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallos. En muchas aplicaciones es crucial asegurar que, una vez que un fallo ha ocurrido y se ha detectado, los datos se restauran al estado de consistencia que existía antes del fallo. Consideremos un programa para transferir 100 € de la cuenta A a la cuenta B. Si ocurre un fallo del sistema durante la ejecución del programa, es posible que los 100 € fueran eliminados de la cuenta A, pero no abonados en la cuenta B, resultando un estado de la base de datos inconsistente. Claramente, es esencial para consistencia de la base de datos que ambos, el abono y el cargo, tengan lugar o que ninguno tenga lugar. Es decir, la transferencia de fondos debe ser *atómica*: ésta debe ocurrir por completo o no ocurrir en absoluto. Es difícil asegurar esta propiedad en un sistema de procesamiento de archivos convencional.

- **Anomalías en el acceso concurrente**. En muchos tipos de aplicaciones es necesario permitir que varios usuarios accedan y modifiquen datos al mismo tiempo para que los tiempos de respuesta se mantengan en valores razonables. En tales sistemas, si no se dispone de los medios adecuados se pueden llegar a producir inconsistencia. Considérese una cuenta bancaria A, que contiene 300 €. Si dos clientes retiran fondos (por ejemplo 30 € y 100 € respectivamente de la cuenta A en aproximadamente el mismo instante de tiempo, el resultado de las ejecuciones de concurrentes de los programas encargados de reflejar esas disposiciones puede dejar la cuenta en un estado incorrecto (o inconsistente). Supongamos que los programas se ejecutan para cada reintegro y escriben el resultado en el fichero al final de su ejecución. Si los dos programas funcionan concurrentemente, pueden leer ambos el valor del saldo de la cuenta (300 €), y operar con esta información es su espacio de direcciones particular en memoria principal. Antes de finalizar, cada uno de las dos ejecuciones actualiza los datos del fichero con la información de las variables en memoria principal, así cada uno de ellos escribe en el saldo de la (única) cuenta (considerada) 270 € y 200 € respectivamente. Dependiendo de cuál de las dos ejecuciones escriba de última, la cuenta puede tener o bien 270 € o 200 € de saldo, en lugar del valor correcto, 170 €. Para evitar estos problemas, el sistema debe mantener algún mecanismo que controle estas situaciones al mismo tiempo que mantenga un nivel razonable de concurrencia para que los tiempos de respuesta no se alarguen demasiado. Sin embargo, ya que se pueden acceder a los datos desde diversas aplicaciones del sistema que no han sido previamente coordinadas, el mantenimiento de la consistencia puede ser complicado.

- **Dependencia de datos**. En los sistemas de procesamiento de archivos la estructura física y la organización de los ficheros está definida en el código de las aplicaciones. Esto implica que es difícil realizar cambios en los archivos existentes. Por ejemplo, cambiar el tamaño del campo *dirección* de un cliente de 40 a 41 caracteres parece un cambio simple, pero requiere la creación de un programa (que sólo se ejecutará una vez) que convierte el fichero de cuentas al nuevo formato. Este programa tiene que:

  - abrir el fichero de cuentas original para leerlo,
  - abrir un fichero temporal con la nueva estructura,
  - leer un registro del fichero original, convertir los datos a la nueva estructura, y escribir en el fichero temporal. Repetir este paso para todos los registros del fichero original,

- borrar el fichero original,
- renombrar el fichero temporal como el fichero de cuentas.

Además, todos los programas que accedían al fichero de cuentas deben ser modificados para adecuarlos a la nueva estructura del fichero. Debe tenerse en cuenta que puede haber muchos programas diferentes que accedan a ese fichero. Por lo tanto, el programador debe localizar todos esos programas, modificarlos, y comprobar que funcionan correctamente. Obsérvese que un programa que utiliza el fichero de cuentas, aunque no utilice el campo dirección, se ve afectado por la modificación. Claramente esto es una tarea que consume mucho tiempo y lo que es peor, puede dar lugar a la introducción de errores en los programas (que antes tal vez no tenían). Esta característica de los sistemas basados en ficheros es conocida como *dependencia de los datos*.

Estas dificultades, entre otras, han motivado el desarrollo de los sistemas de bases de datos. Como indicamos antes un SGBD es un conjunto de programas, pero dichos programas por sí solos, no solventarían los problemas mencionados. Este software debe ir unido a la figura del administrador de la base de datos (ABD).

De un modo más concreto para solventar las carencias de los sistemas basados en ficheros los SGBD proporcionan:

- **Control de redundancia**. La información almacenada en una base de datos suele tener muchos usuarios, cada uno de los cuales puede requerir una perspectiva o *vista* diferente de la base de datos. Una vista puede ser un subconjunto de la base de datos o puede contener *datos virtuales* derivados de los que están almacenados físicamente en la base de datos.

  Con el enfoque de base de datos, las vistas de los diferentes grupos de usuarios se integran durante el diseño de la base de datos. Para conservar la consistencia, debe crearse un diseño que almacene cada dato lógico (como el nombre o la fecha de nacimiento de un cliente) en un *solo lugar* de la base de datos. Ello evita la inconsistencia y ahorra espacio de almacenamiento. Sin embargo en algunos casos puede convenir la *redundancia controlada* para mejorar el rendimiento de las consultas. Este tipo de decisiones junto con otras tareas son responsabilidad del ABD.

- **Lenguajes no procedimentales o declarativos**. Para paliar los problemas debidos a accesos no planificados, los SGBD proporcionan un Lenguaje de Manipulación de Datos (LMD) que permite a los usuarios insertar, actualizar, borrar y recuperar datos de la base de datos mediante un lenguaje no procedimental, es decir, donde se especifica lo *que se quiere* y *NO cómo hacerlo*. Este tipo de lenguajes facilitan los accesos no planificados puesto que son lenguajes sencillos que pueden ser usados incluso por personal no especialista.

  También disponen de un Lenguaje de Definición de Datos (LDD) que permite a los usuarios especificar los tipos de datos, estructuras y restricciones sobre los datos que se van almacenar en la base de datos. Este lenguaje también es declarativo.

  En la práctica estos dos lenguajes no están separados en la mayoría de los SGBD, ambos forman parte de un único lenguaje llamado SQL.

- **Administración centralizada de los datos**. Para evitar los problemas de aislamiento de los datos, integridad de los datos, seguridad y acceso concurrente, los SGBD proporcionan los mecanismos adecuados, que además irán acompañados de la figura del ABD que se encargará de:

  - Imponer normas: el ABD puede definir e imponer normas a los usuarios de la base de datos. Esto facilita la comunicación y cooperación entre diversos departamentos, proyectos y usuarios de esa organización. Es posible definir normas para los nombres y formatos de los elementos de datos, para las estructuras de acceso, etc. Es más fácil que el ABD imponga normas en un entorno centralizado de bases de datos que un entorno en el que cada grupo de usuarios tenga el control de sus propios ficheros y programas.
  - Definir las estructuras de almacenamiento y su ubicación. Como por ejemplo, cuántos discos y partes de los mismos se utilizarán para almacenar la base de datos y los programas del SGBD.
  - Definir las estructuras de datos, o en su caso dar permiso a los usuarios para que creen sus propias estructuras de datos. En el caso de que los usuarios puedan crear sus estructuras de datos, siempre las crearán en los lugares preestablecidos por el ABD.
  - Definir políticas de seguridad. A cada usuario se le asigna un nombre de usuario que debe validar con una contraseña. A cada usuario o grupo de usuarios el ABD le asigna una serie de privilegios (acceso, modificación, inserción, borrado, etc.) sobre los objetos del SGBD.
  - Seguimiento y evaluación del sistema. Cuando el sistema tiene cualquier problema, bien sea de rendimiento o derivado de una situación excepcional como un fallo, el ABD se encarga de subsanarlo en los posible utilizando las herramientas que el SGBD le proporciona.

  Para todas estas tareas y otras, el SGBD debe proporcionar el soporte adecuado.

- **Garantizar el cumplimiento de las restricciones de integridad**. Los SGBD ofrecen recursos para definir restricciones de integridad y garantizar que se cumplan.

- **Representación de vínculos complejos entre los datos**. El SGBD puede representar diversas relaciones complejas entre los datos y también obtener y actualizar con rapidez y eficiencia datos que estén mutuamente relacionados. Por ejemplo, en nuestro ejemplo bancario, podemos tener un fichero con los datos personales de los clientes de modo que no se repitan para cada una de sus cuentas, de este modo, los registros de cuentas de cada cliente deben estar relacionados con el registro correspondiente con sus datos personales.

- **Suministro de copias de seguridad y recuperación**. La mayoría de los SGBD cuenta con recursos para recuperarse de fallos de hardware o de software.

- **Atomicidad**. Los SGBD proporcionan mecanismos para asegurar que ciertas transacciones se realicen de forma atómica, incluso gracias a las capacidades de recuperación, en presencia de fallos software o hardware.

- **Compartimiento de datos**. Los SGBD multiusuario permiten que varios usuarios accedan de forma simultánea a las bases de datos. Los SGBD incluyen software de control de concurrencia para tal efecto.

- **Separación entre los programas y los datos, y abstracción de datos**. Como mencionamos antes, en los sistemas basados en ficheros, la estructura de los ficheros de datos viene integrada en los programas de acceso, así que cualquier modificación de la estructura de un fichero requiere la *modificación de todos los programas* que acceden a dicho fichero. Por el contrario, los programas de acceso del SGBD no requieren dichas modificaciones en la mayoría de los casos. La estructura de los ficheros de datos se almacena en el *catálogo del SGBD* separadamente de los programas de acceso.

  El catálogo del SGBD contiene la descripción completa de la estructura de la base de datos y sus restricciones. Esta descripción incluye, entre otras cosas, la estructura de cada fichero (ya que al final, los datos siempre se almacenan en ficheros), el tipo y formato de almacenamiento de cada elemento y varias restricciones sobre los datos. La información almacenada en el catálogo se denomina *meta-datos*.

  Gracias a almacenar la descripción de los ficheros separada de los programas de acceso, los SGBD son capaces de proporcionar lo que se denomina *independencia entre programas y datos*, esto es, los programas se aíslan de los cambios en el modo en el que los datos son estructurados y almacenados. Por ejemplo, si tenemos un programa que accede al saldo de una cuenta en nuestro ejemplo bancario, si lo hacemos en un sistema de procesamiento de archivos, dicho programa contiene la definición del fichero, si en un momento dado se añade un nuevo campo al fichero de cuentas, como por ejemplo *tipo de cuenta de ahorro*, el programa deberá ser modificado (además del fichero) para manejar el nuevo fichero. Sin embargo, si la información sobre cuentas se almacenase en una BD, el antiguo programa no necesitaría ser modificado para continuar funcionando, únicamente sería necesario modificar la definición del fichero de cuentas en el catálogo del sistema.

  En las bases de datos orientadas a objetos y objeto-relacionales, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos. Una *operación* se especifica en dos partes. La *interfaz* de una operación incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La *implementación* (o *método*) de la operación se especifica separadamente y puede modificarse sin modificar la interfaz. Los programas de aplicación de los usuarios pueden operar sobre los datos invocando a dichas operaciones a través de sus nombres y argumentos, sea cual sea la forma en la que se han implementado. Esto podría denominarse *independencia entre programas y operaciones*.

  La característica que permite la independencia de programas y datos se llama *abstracción de datos*. Un SGBD ofrece a los usuarios una *representación conceptual* de los datos que no incluye muchos detalles sobre el almacenamiento de los mismos ni sobre cómo se implementan las operaciones. En términos informales, *un modelo de datos*[1] es un tipo de abstracción de datos que se utiliza para proporcionar esta representación conceptual. El modelo de datos utiliza conceptos lógicos, tales como objetos, sus propiedades y sus relaciones, que pueden ser más fáciles de comprender

[1] A veces se utiliza la palabra modelo para denotar una descripción o esquema de una base de datos.

por los usuarios que los conceptos de almacenamiento (como por ejemplo los vistos en los capítulos anteriores). Por tanto, el modelo de datos oculta los detalles de almacenamiento e implementación que no interesan a la mayoría de los usuarios de la base de datos.

2.3. Modelos de datos, esquemas e instancias

Un modelo de datos es una colección de herramientas conceptuales que sirven para describir la estructura de una base de datos. Con el concepto de *estructura de una base de datos* nos referimos a los tipos de datos, los vínculos y las restricciones que deben cumplirse para esos datos. La mayoría de los modelos de datos contienen además un conjunto de *operaciones básicas* para especificar lecturas y actualizaciones de la base de datos.

Además en los últimos modelos de datos (orientado a objetos y objeto-relacional) también incluyen conceptos para especificar el *aspecto dinámico* o *comportamiento* de una aplicación de bases de datos. Esto permite al diseñador especificar un conjunto de *operaciones definidas por el usuario* válidas que están permitidas sobre los objetos de la base de datos. Un ejemplo de operación definida podría ser *calcular saldo* sobre un objeto *cuenta*.

2.3.1. Categorías de los modelos de datos

Se han propuesto muchos modelos de datos que 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 *conceptuales* disponen de conceptos muy cercanos al modo como la mayoría de los usuarios percibe los datos, mientras que los *modelos de datos de bajo nivel* o *físicos* proporcionan conceptos que describen los detalles de cómo se almacenan los datos en el ordenador. Entre estos dos extremos hay una clase de *modelos de datos de representación* (o de *implementación*) cuyos conceptos pueden ser entendidos por los usuarios finales aunque no están demasiado alejados de la forma en que los datos se organizan dentro del computador. Los modelos de datos de representación ocultan algunos detalles sobre cómo se almacenan los datos, pero pueden ser implementados de manera directa en un sistema de computador.

Los modelos de datos conceptuales utilizan conceptos como entidades, atributos y relaciones. Una *entidad* representa un objeto o objeto del mundo real, como un empleado o un proyecto, que se describe en la base de datos. Un *atributo* representa alguna propiedad de interés que da una descripción más amplia de una entidad, como el nombre o el salario del empleado. Una *relación* entre dos o más entidades describe una interacción entre las entidades, por ejemplo, la relación "pertenece" entre cliente y cuenta. Un ejemplo de modelo conceptual es el *modelo Entidad-Relación* que veremos en el Capítulo **??**.

Los modelos de datos de representación o de implementación son los más utilizados en los SGBD tradicionales, y entre ellos se encuentra ampliamente utilizado el *modelo de datos relacional*, así como los *modelos de red* y *jerárquico*, que se utilizaron mucho en el pasado. El modelo relacional se presentará en el Capítulo **??** Los modelos de datos de representación representan los datos mediante estructuras de registro y por tanto, a veces se les denomina *modelos de datos basados en registros*.

Podemos considerar a los *modelos de datos orientados a objetos* como una nueva familia de modelos de implementación de alto nivel más próxima a los modelos conceptuales.

Los modelos de datos físicos describen cómo se almacenan los datos en el ordenador mediante la representación de información como, por ejemplo, formatos de registro,

ordenaciones de registros, índices, etc., es decir, al nivel presentado en los capítulos anteriores.

2.3.2. Esquemas, instancias y estado de la base de datos

En cualquier modelo de datos es importante distinguir entre la descripción de la base de datos y de la base de datos misma. La descripción se conoce como *esquema de la base de datos*, se especifica durante el diseño de la base de datos y no es de esperar que se modifique muy a menudo. Un posible esquema para nuestro ejemplo bancario podría ser:

Ejemplo 2.3.1 Ejemplo bancario:

```
cuenta(nºcuenta, tipo, saldo)
cliente(DNI, nombre, dirección, tlf)
cuenta-cliente(nºcuenta, DNI)
```

□

Como se puede apreciar, en esta representación del esquema de la base de datos no se presentan muchos detalles, como los tipos de datos de los atributos o campos de cada fichero.

Los datos reales de la base de datos pueden cambiar con mucha frecuencia; por ejemplo en nuestro ejemplo bancario, la base de datos cambia cada vez que se modifica el saldo de una cuenta. Los datos que contiene la base de datos en un momento determinado se denominan *estado de la base de datos* o *instantánea*. También se le denomina conjunto actual de ocurrencias o instancias de la base de datos. Es posible construir muchos estados de una base de datos que correspondan a un esquema particular. Cada vez que insertamos o eliminamos un registro, o que modifiquemos el valor de un elemento de datos, pasamos de un estado de la base de datos a otro.

La distinción entre el esquema y el estado de la base de datos es muy importante. Cuando *definimos* una nueva base de datos, especificamos su esquema al SGBD. En ese momento, el estado correspondiente a la base de datos está *vacío*, es decir sin datos. Cuando *poblamos* o *cargamos* los datos por primera vez, la base de datos cambia de estado. De ahí en adelante, siempre que se aplique una operación de actualización a la base de datos, tendremos otro estado. En cualquier punto del tiempo la base de datos tiene un *estado actual*. El SGBD se encarga en parte de asegurar que todos los estados de la base de datos sean *estados válidos*; es decir, que satisfagan la estructura y las restricciones especificadas en el esquema. Por tanto es muy importante especificar un esquema correcto, es decir, hay que tener mucho cuidado al realizar el diseño del esquema. El SGBD almacena el esquema en el catálogo, para consultarlo cuando sea necesario. En ocasiones, al esquema se le llama *intensión* y a un estado de la base de datos *extensión* del esquema.

2.4. Arquitectura de un SGBD e independencia de datos

El comité de planificación de estándares y requisitos (SPARC) del Instituto Nacional Americano de Estándares (ANSI), ANSI/X3/SPARC, produjo en 1975 un documento donde se especificaba una terminología estándar y una propuesta de arquitectura estándar de SGBD. ANSI-SPARC reconoció la necesidad de una aproximación de tres niveles con un catálogo del sistema. Esta propuesta reflejaba los requisitos publicados por IBM años antes, que se concentraba en la necesidad de una capa que fuese independiente de la implementación de

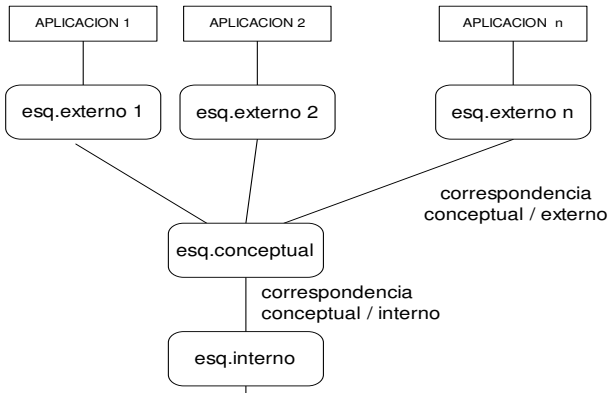

Figura 2.1: Arquitectura ANSI

manera que aislara los programas de las representaciones internas. Aunque el modelo de ANSI-SPARC no se convirtió en estándar, todavía proporciona una base para entender algunas de las funcionalidades de los SGBD.

Resumiendo, el objetivo de la arquitectura propuesta por ANSI-SPARC es la separación de las aplicaciones de usuario y la base de datos física. Para nuestro objetivo, la idea central de la arquitectura ANSI-SPARC es la identificación de tres niveles de de abstracción, esto es, tres niveles de descripción de los datos:

1. El *nivel interno* tiene un *esquema interno*, que describe la estructura física de almacenamiento de la base de datos. El esquema interno emplea un modelo de datos físico y describe todos los detalles para su almacenamiento, así como los índices de la base de datos. Dicho de un modo sencillo, describe *cómo* se almacenan realmente los datos.

2. El *nivel conceptual* tiene un *esquema conceptual*, que describe la estructura de la base de datos completa para una comunidad de usuarios. El esquema conceptual oculta los detalles de las estructuras físicas de almacenamiento y se concentra en describir entidades, tipos de datos, vínculos, operaciones de los usuarios y restricciones. En este nivel se puede usar un modelo de datos de alto nivel o uno de implementación. En resumen, el nivel conceptual describe *qué* datos se almacenan en la base de datos y los vínculos (o relaciones) que existen entre ellos.

3. El *nivel externo* o *de vistas* incluye 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 podemos usar un modelo de datos de alto nivel o uno de implementación.

La arquitectura de tres niveles es una herramienta adecuada para que el usuario visualice los niveles del esquema de un SGBD. Sin embargo, la mayoría de los SGBD no separa los tres

niveles completamente, pero en algunos de ellos se soporta, en cierta medida, esta separación. Algunos SGBD incluyen detalles del nivel físico en el esquema conceptual.

Cabe señalar que los tres esquemas no son más que *descripciones* de los datos; los únicos datos que existen *realmente* están en el nivel físico.

En un SGBD basado en la arquitectura de tres niveles, cada grupo de usuarios hace referencia exclusiva a su propio esquema externo; por tanto, el SGBD debe transformar una solicitud expresada en términos de un esquema externo, en una solicitud expresada en términos del esquema conceptual y luego en una solicitud en el esquema interno que se procesará sobre la base de datos almacenada. Si la solicitud es una 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 solicitudes y resultados de un nivel a otro se denomina *correspondencia* o *transformación*.

2.4.1. Independencia de datos

La arquitectura de tres niveles implementa la independencia de datos en dos niveles:

1. La *independencia lógica de los datos* es la capacidad de modificar el esquema conceptual sin tener que alterar los esquemas externos ni los programas de aplicación. Podemos modificar el esquema conceptual para ampliar la base de datos (añadiendo un nuevo tipo de registro o un campo a un registro existente) o para reducir la base de datos (eliminando un tipo de registro o un campo de un registro). En el segundo caso, la modificación no deberá afectar a los esquemas externos que sólo se refieran a los datos restantes. Además, las restricciones podrán modificarse en el esquema conceptual sin afectar a los esquemas externos ni a los programas de aplicación.

2. La *independencia física de los datos* es la capacidad de modificar el esquema interno sin tener que alterar el esquema conceptual (o los externos). Tal vez sea preciso modificar el esquema interno por la necesidad de reorganizar ciertos ficheros físicos (por ejemplo, al crear índices, o porque se cambia de organización) con el fin de mejorar el rendimiento de las operaciones de recuperación y actualización. Si la base de datos aún contiene los mismos datos, no será necesario modificar el esquema conceptual.

En todo SGBD de múltiples niveles es preciso ampliar el catálogo de modo que incluya información sobre cómo establecer la correspondencia entre las solicitudes y los datos entre los diversos niveles. El SGBD utiliza software adicional para realizar estas correspondencias haciendo referencia a la información de correspondencia que se encuentra en el catálogo. La independencia de datos se logra porque, al modificarse el esquema de algún nivel, el esquema del nivel inmediatamente superior permanece sin cambios; sólo se modifica la *correspondencia* entre los dos niveles.

La arquitectura de tres esquemas puede facilitar la consecución de la verdadera independencia de datos, tanto física como lógica. Sin embargo, los dos niveles de correspondencia implican un gasto extra durante la compilación y la ejecución de una consulta o de un programa, lo cual reduce la eficiencia del SGBD. Por ello, son pocos los SGBD que han implementado la arquitectura de tres niveles completa.

2.4.2. Estructura de un sistema de bases de datos

Un sistema de bases de datos se divide en módulos que se encargan de cada una de las responsabilidades del sistema completo. Los componentes funcionales de un sistema de bases de datos se pueden dividir a grandes rasgos en los *componentes gestor de almacenamiento* y *procesador de consultas*.

El gestor de almacenamiento es importante porque las bases de datos requieren normalmente una gran cantidad de espacio almacenamiento. Las bases de datos corporativas tienen un tamaño de entre cientos de gigabytes y, para las mayores bases de datos, terabytes de datos. Para minimizar las transferencias entre disco y memoria, es fundamental que el SGBD estructure los datos del modo adecuado.

El procesador de consultas es importante porque ayuda al SGBD a simplificar y facilitar el acceso a los datos. Gracias a los esquemas externos, los usuarios no deben preocuparse por los detalles físicos de bajo nivel. Sin embargo, el rápido procesamiento de las consultas y actualizaciones es importante. Como los usuarios realizan las actualizaciones y las consultas haciendo referencia al esquema externo y además están escritas en un lenguaje no procedimental, es trabajo del SGBD traducir todo ello a una secuencia de operaciones (procedimentales) en el nivel interno, haciendo referencia al esquema interno. Dicha traducción no es sencilla, y debe hacerse de modo que los tiempos de respuesta sean razonables.

Gestor de almacenamiento

Un gestor de almacenamiento es un módulo del SGBD que proporciona la interfaz entre los datos de bajo nivel en la base de datos y los programas de aplicación y consultas emitidas al sistema. El gestor de almacenamiento es responsable de la interacción con el gestor de archivos. Los datos en bruto se almacenan en disco utilizando un sistema de archivos, que está habitualmente disponible en un sistema operativo convencional, aunque en algunos casos, el SGBD contiene su propio sistema de archivos. El gestor de almacenamiento traduce las instrucciones en LMD y LDD a órdenes de un sistema de archivos de bajo nivel. Así, el gestor de almacenamiento es responsable del almacenamiento, recuperación y actualización de los datos en la base de datos.

Los componentes del gestor de almacenamiento incluyen:

- **Gestor de autorización e integridad**, que comprueba que se satisfagan las restricciones de integridad y la autorización de los usuarios para acceder a los datos.
- **Gestor de transacciones**, que asegura que la base de datos quede en un estado válido (consistente) a pesar de los fallos del sistema, y que las ejecuciones de transacciones concurrentes ocurran sin conflictos.
- **Gestor de archivos**, que gestiona la reserva de espacio de almacenamiento de disco y las estructuras de datos usadas para representar la información almacenada en disco.
- **Gestor de memoria intermedia**, que es responsable de traer los datos del disco de almacenamiento a memoria principal y decidir qué datos tratar en memoria caché.

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

- **Archivos de datos**, que almacenan las bases de datos en sí.

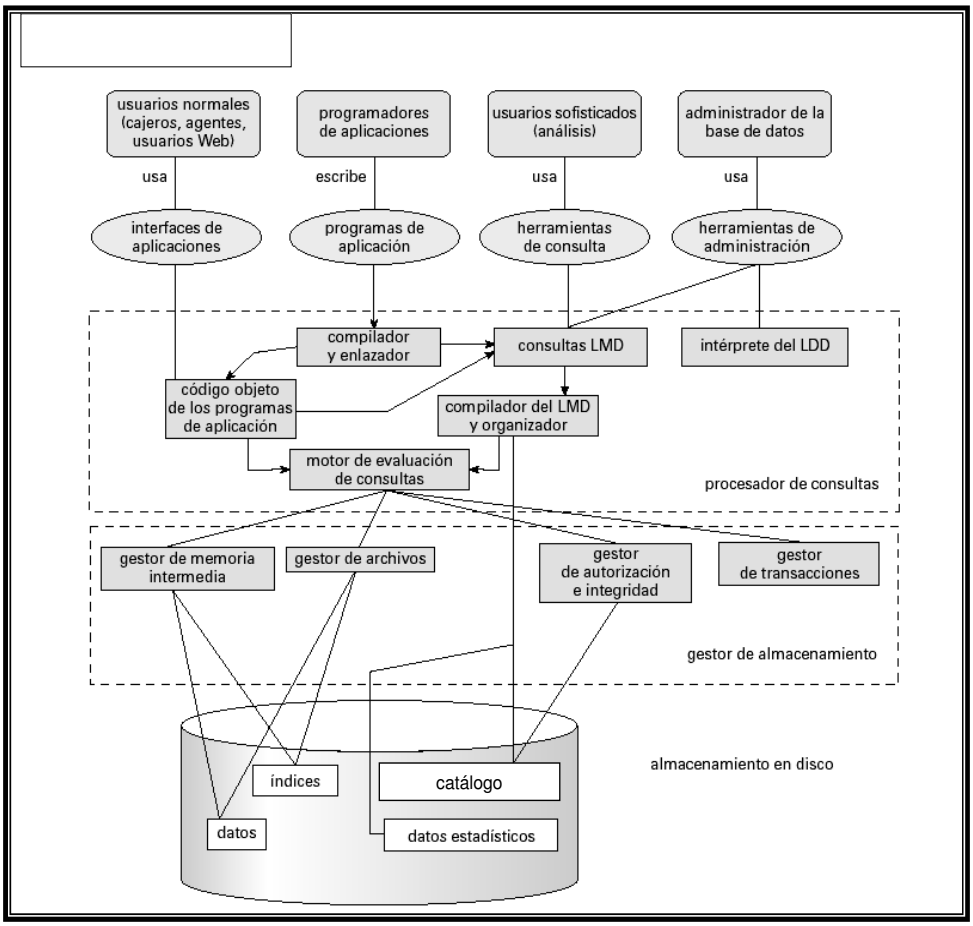

Figura 2.2: Estructura de un SGBD

- **Catálogo** o **diccionario de datos**, que almacena metadatos acerca de la estructura de la base de datos, en particular, el esquema de la base de datos.

- **Índices**, que proporcionan acceso rápido a los registros de los ficheros que almacenan la base de datos.

Procesador de consultas

Los componentes del procesador de consultas incluyen:

- **Intérprete del LDD**, que interpreta las instrucciones del LDD y registra las definiciones en el catálogo.

- **Compilador del LMD**, que traduce las instrucciones del LMD a un plan de evaluación (ejecución) que consiste en instrucciones de bajo nivel que entiende el motor de evaluación de consultas.

  Como el LMD es un lenguaje declarativo, una consulta se puede traducir habitualmente en varios planes de evaluación alternativos que proporcionan el mismo resultado. El compilador de LMD también realiza la *optimización de consultas*, es decir, elige el plan de ejecución de menor coste de entre todas las alternativas.

- **Motor de evaluación de consultas**, que ejecuta las instrucciones de bajo nivel generadas por el compilador del LMD.

- **Compilador y enlazador**. En realidad estos elementos suelen ser externos al SGBD. El SGBD proporciona un precompilador que extrae instrucciones en LMD y LDD de un programa de aplicación escrito en un lenguaje de programación anfitrión. Estas instrucciones se envían al compilador de LMD para convertirlas en planes de evaluación ejecutables por el motor de evaluación de consultas o (según el caso) al intérprete de LDD. El resto del programa se envía al compilador convencional del lenguaje anfitrión. El código objeto generado por el compilador del lenguaje anfitrión y los planes de evaluación generados por el compilador de LMD se enlazan, formando una aplicación (transacción programada) cuyo código ejecutable incluye llamadas al motor de evaluación de consultas durante el tiempo de ejecución.

En la Figura 2.2 se muestran estos componentes y sus conexiones. No se pretende presentar un SGBD específico, sino más bien ilustrar los módulos más representativos de un SGBD.

2.5. Historia de los SGBD

Ya hemos visto que el predecesor de los SGBD fueron los sistemas basados en ficheros. Sin embargo, no existe un punto en el tiempo donde los SGBD aparecieron y los sistemas basados en ficheros desaparecieron. De hecho, los ficheros se siguen utilizando en ciertas situaciones. Algunos han apuntado que los SGBDs tienen sus raíces en el proyecto de viaje a la luna Apolo, que fue iniciado en respuesta al reto del presidente Kennedy de llevar un hombre a la luna al final de la década de los 60. En aquellos momentos no existía un sistema que fuera capaz de manejar las cantidades enormes de información que iba generar aquel proyecto.

Para solucionar el problema, North Ameriacan Aviation (NAA), el principal contratista del proyecto, desarrolló un software que se llamó GUAM (*Generalized Update Access Method*). GUAM se basaba en el concepto de que pequeños componentes unidos forman componentes más grandes, y así sucesivamente hasta que el producto final es construido. Esta estructura, que forma un árbol al revés, es conocida como *estructura jerárquica*. A mediados de los 60, IBM se unió a NAA para evolucionar GUAM a lo que se conoce actualmente como *sistema de manejo de información* (IMS). La razón por la cual IBM restringió IMS al manejo de jerarquías de registros fue para permitir el uso de dispositivos de almacenamiento secuencial, mayormente cintas, lo que era una imposición del mercado en aquel momento. Esta restricción fue posteriormente eliminada. Aún así, el IMS siguió siendo el principal SGBDs jerárquico utilizado durante mucho tiempo en las instalaciones de mainframes.

A mediados de los 60, apareció otro desarrollo importante, el IDS (*Integrated Data Store*) de General Electric. Este trabajo fue liderado por uno de los pioneros de los sistemas de bases

de datos, Charles Bachmann. Este trabajo dio lugar a un nuevo tipo de SGBDs conocido como SGBD en *red*, que tuvo un gran impacto en los sistemas de información de aquellos tiempos. El modelo de base de datos en red fue desarrollado en parte para afrontar la necesidad de representar relaciones entre datos más complejas que las que se podían incluir en el modelo jerárquico, y en parte para imponer un estándar de bases de datos. Para ayudar a establecer tal estándar, la conferencia en lenguajes de sistemas de datos (CODASYL), que incluía representantes del gobierno americano y del mundo de los negocios, formaron la *list processing task force* en 1965, que en 1967 pasó a llamarse *data base task group* (DBTG). El objetivo del DBTG era establecer definiciones de estándares para un entorno que permitiese la creación de bases de datos y de la manipulación de los datos en ellas almacenadas. En 1969 se presentó un borrador que se convirtió en informe definitivo en 1971. La propuesta del DBTG identificaba tres componentes:

- El *esquema en red*, la organización lógica de la base de datos al completo, tal y como la ve el ABD (incluye la definición del nombre de la base de datos, el tipo de cada registro, y los campos de cada registro).
- El *subesquema*, la parte de la base de datos tal y como la ve el usuario o un programa de aplicación.
- Un lenguaje de manejo de datos para; definir las características de los datos y la estructura de los datos y, para manipular los datos.

El DBTG definió tres lenguajes:

- Un LDD de esquemas, que permite al ABD definir el esquema.
- Un LDD de subesquema, que permite a los programas de aplicación definir las partes de la base de datos que necesitan.
- Un LMD para manipular los datos.

Aunque el informe no fue adoptado por ANSI, varios sistemas fueron desarrollados posteriormente siguiendo la propuesta del DBTG. Esos sistemas se conocen ahora como sistemas CODASYL or DBTG. Los sistemas CODASYL y jerárquicos representan la *primera generación* de SGBDs. Estos sistemas tienen algunas desventajas importantes:

- Es necesario escribir programas bastante complejos para realizar consultas incluso simples debido a que el acceso a los registros era navegacional.
- Había muy poco independencia de datos.
- No existen unos fundamentos teóricos ampliamente aceptados que les den soporte.

En 1970 E. F. Codd del laboratorio de investigación de IBM escribió un artículo fundamental en la historia de los SGBD, el modelo relacional. Este artículo fue muy oportuno y encaró los problemas de los sistemas anteriores. A partir de aquel momento se desarrollaron muchos SGBD relacionales experimentales, pero los primeros comerciales no aparecieron hasta finales de los 70 o principios de los 80. Un proyecto que influyó mucho en la evolución de los SGBD relacionales fue el System R desarrollado en el laboratorio de investigación de IBM

San José en California a finales de los 70. Este proyecto fue diseñado para probar la viabilidad del modelo relacional, creando una implementación de sus estructuras de datos y operaciones. Este proyecto dio lugar a dos grandes desarrollos:

- El desarrollo de un lenguaje estructurado de consulta llamado SQL, que desde entonces es el lenguaje estándar del los SGBDs relacionales.
- La producción de varios SGBD relacionales comerciales durante los años 80, por ejemplo DB2 y SQL/DS de IBM y Oracle.

En la actualidad el los SGBDs relacionales son con mucho los sistemas más populares, hay cientos de ellos, desde versiones para PCs hasta versiones para mainframes, aunque algunos de ellos extienden la definición del modelo relacional al incorporar nuevas capacidades. Otros ejemplos de SGBDs relacionales y multiusuario son INGRES II de Computer Associates y SQL Server de Microsoft. Ejemplos de SGBDs relacionales para PC son Access de Microsoft, Paradox e Interbase de Borland. Los SGBDs relacionales son conocidos como la *segunda generación* de SGBDs. El modelo de datos relacional se presentan en el Capítulo **??**.

El modelo relacional tiene sus limitaciones. Se ha realizado mucha investigación en el modelo relacional para intentar mejorarlo. En 1976, Chen presentó el modelo Entidad-Relación, que hoy en día es una técnica ampliamente aceptada para el diseño de bases de datos. En 1979, Codd intentó solucionar las limitaciones de su modelo relacional extendiéndolo, el resultado lo llamó modelo RM/T (1979) y posteriormente RM/V2 (1990). Los intentos por proporcionar un modelo de datos que represente el mundo real más precisamente que el modelo relacional se han denominado como *modelado de datos semántico*.

En respuesta a la creciente complejidad de las nuevas aplicaciones de bases de datos, han aparecido dos nuevos modelos de bases de datos: los SGBDs *orientados a objetos* y los SGBDs *objeto-relacionales*. Sin embargo, a diferencia de los modelos anteriores, la definición concreta de estos modelos no está clara. Esta evolución representa la *tercera generación* de los SGBDs.

2.6. Desventajas de los SGBDs

A pesar de los numerosos beneficios que reporta el uso de SGBDs, también hay que tener en cuenta una serie de desventajas:

- **Complejidad**, debido a todas las funciones que cumple un SGBD, estos sistemas se han convertido en conjunto software extremadamente complejo. Todos los involucrados en su funcionamiento deben comprenderlo bien y estar bien entrenados, puesto que un mal uso del sistema puede llevar a serias consecuencias para la organización.
- **Tamaño**, también debido a sus numerosas funcionalidades, el paquete software de un SGBD es muy grande, ocupando mucho espacio en disco y necesitando cantidades sustanciales de memoria principal para ejecutarse eficientemente.
- **Coste del SGBD**, el coste varía mucho dependiendo del entorno y funcionalidad proporcionada. Desde SGBD monousuario incluidos en paquetes de oficina con precios de 100 €, hasta SGBD multiusuario para mainframes que pueden llegar a cantidades astronómicas. Además, en los grandes sistemas hay un coste de mantenimiento anual.

- **Coste del hardware**, debido a los requisitos de disco, memoria e incluso procesador del SGBD, hace falta realizar inversiones en este capítulo para conseguir que el SGBD funcione eficientemente.
- **Coste de conversión**, en algunas ocasiones los costes antes mencionados son insignificantes con el coste de migrar las aplicaciones existentes para que funcionen sobre un SGBD. Este coste es una de las razones por la que algunas organizaciones mantienen sus sistemas antiguos, lo que se llama sistemas *legacy*.
- **Rendimiento**, los sistemas de ficheros se diseñan para el problema específico que se está tratando. Por lo tanto, su rendimiento es muy bueno. Sin embargo, los SGBDs se diseñan para dar respuesta a todo tipo de aplicaciones y por lo tanto su enfoque es más general.
- **Alto impacto de fallos**, la centralización de recursos incrementa la vulnerabilidad del sistema. Dado que todos los usuarios y las aplicaciones se basan en la disponibilidad del SGBD, el fallo de cualquier componente puede hacer que las operaciones se detengan.