Introducción a las bases de datos

Rafael Camps Paré

P06/M2109/02147

UOC
www.uoc.edu

Índice

Introducción a las bases de datos

Introducción

Empezaremos esta unidad didáctica viendo cuáles son los **objetivos de los sistemas de gestión de las bases de datos (SGBD)** y, a continuación, daremos una visión general de la **arquitectura**, el **funcionamiento** y el **entorno** de estos sistemas.

Objetivos

En los materiales didácticos de esta unidad encontraréis las herramientas para adquirir una visión global del mundo de las BD y de los SGBD, así como para alcanzar los siguientes objetivos:

1. Conocer a grandes rasgos la evolución de los SGBD desde los años sesenta hasta la actualidad.

2. Distinguir los principales objetivos de los SGBD actuales y contrastarlos con los sistemas de ficheros tradicionales.

3. Saber explicar mediante ejemplos los problemas que intenta resolver el concepto de *transacción*.

4. Relacionar la idea de flexibilidad en los cambios con la independencia lógica y física de los datos, así como con la arquitectura de tres niveles.

5. Distinguir los principales modelos de BD.

6. Conocer a grandes rasgos el funcionamiento de un SGBD.

7. Saber relacionar los diferentes tipos de lenguajes con los diferentes tipos de usuarios.

1. Concepto y origen de las BD y de los SGBD

Las aplicaciones informáticas de los años sesenta acostumbraban a darse totalmente por lotes (*batch*) y estaban pensadas para una tarea muy específica relacionada con muy pocas entidades tipo.

Aplicaciones informáticas de los años sesenta

La emisión de facturas, el control de pedidos pendientes de servir, el mantenimiento del fichero de productos o la nómina del personal eran algunas de las aplicaciones informáticas habituales en los años sesenta.

Cada aplicación (una o varias cadenas de programas) utilizaba ficheros de movimientos para actualizar (creando una copia nueva) y/o para consultar uno o dos ficheros maestros o, excepcionalmente, más de dos. Cada programa trataba como máximo un fichero maestro, que solía estar sobre cinta magnética y, en consecuencia, se trabajaba con acceso secuencial. Cada vez que se le quería añadir una aplicación que requería el uso de algunos de los datos que ya existían y de otros nuevos, se diseñaba un fichero nuevo con todos los datos necesarios (algo que provocaba redundancia) para evitar que los programas tuviesen que leer muchos ficheros.

A medida que se fueron introduciendo las líneas de comunicación, los terminales y los discos, se fueron escribiendo programas que permitían a varios usuarios consultar los mismos ficheros *on-line* y de forma simultánea. Más adelante fue surgiendo la necesidad de hacer las actualizaciones también *on-line*.

Integración de aplicaciones

Por ejemplo, se integra la aplicación de facturas, la de pedidos pendientes y la gestión del fichero de productos.

A medida que se integraban las aplicaciones, se tuvieron que interrelacionar sus ficheros y fue necesario eliminar la redundancia. El nuevo conjunto de ficheros se debía diseñar de modo que estuviesen interrelacionados; al mismo tiempo, las informaciones redundantes (como por ejemplo, el nombre y la dirección de los clientes o el nombre y el precio de los productos), que figuraban en los ficheros de más de una de las aplicaciones, debían estar ahora en un solo lugar.

El acceso *on-line* y la utilización eficiente de las interrelaciones exigían estructuras físicas que diesen un acceso rápido, como por ejemplo los índices, las multilistas, las técnicas de *hashing*, etc.

Estos **conjuntos de ficheros interrelacionados**, con estructuras complejas y compartidos por varios procesos de forma simultánea (unos *on-line* y otros por lotes), recibieron al principio el nombre de *Data Banks*, y después, a inicios de los años setenta, el de *Data Bases*. Aquí los denominamos **bases de datos (BD)**.

El ***software* de gestión de ficheros** era demasiado elemental para dar satisfacción a todas estas necesidades. Por ejemplo, el tratamiento de las interrelaciones no estaba previsto, no era posible que varios usuarios actualizaran datos simultáneamente, etc. La utilización de estos conjuntos de ficheros por parte de los programas de aplicación era excesivamente compleja, de modo que, especialmente durante la segunda mitad de los años setenta, fue saliendo al mercado

software más sofisticado: los *Data Base Management Systems*, que aquí denominamos **sistemas de gestión de BD (SGBD)**.

Con todo lo que hemos dicho hasta ahora, podríamos definir el término *BD*; una **base de datos de un SI** es la representación integrada de los conjuntos de entidades instancia correspondientes a las diferentes entidades tipo del SI y de sus interrelaciones. Esta representación informática (o conjunto estructurado de datos) debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos.

En otras palabras, una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir utilizaciones varias y simultáneas.

Los ficheros tradicionales y las BD

Aunque de forma muy simplificada, podríamos enumerar las principales diferencias entre los ficheros tradicionales y las BD tal y como se indica a continuación:

1) Entidades tipos:

- Ficheros: tienen registros de una sola entidad tipo.
- BD: tienen datos de varias entidades tipo.

2) Interrelaciones:

- Ficheros: el sistema no interrelaciona ficheros.
- BD: el sistema tiene previstas herramientas para interrelacionar entidades.

3) Redundancia:

- Ficheros: se crean ficheros a la medida de cada aplicación, con todos los datos necesarios aunque algunos sean redundantes respecto de otros ficheros.
- BD: todas las aplicaciones trabajan con la misma BD y la integración de los datos es básica, de modo que se evita la redundancia.

4) Usuarios

- Ficheros: sirven para un solo usuario o una sola aplicación. Dan una sola visión del mundo real.
- BD: es compartida por muchos usuarios de distintos tipos. Ofrece varias visiones del mundo real.

2. Evolución de los SGBD

Para entender mejor qué son los SGBD, haremos un repaso de su evolución desde los años sesenta hasta nuestros días.

2.1. Los años sesenta y setenta: sistemas centralizados

Los SGBD de los años sesenta y setenta (IMS de IBM, IDS de Bull, DMS de Univac, etc.) eran **sistemas totalmente centralizados**, como corresponde a los sistemas operativos de aquellos años, y al *hardware* para el que estaban hechos: un gran ordenador para toda la empresa y una red de terminales sin inteligencia ni memoria.

Los **primeros SGBD** –en los años sesenta todavía no se les denominaba así– estaban orientados a facilitar la utilización de grandes conjuntos de datos en los que las interrelaciones eran complejas. El arquetipo de aplicación era el *Bill of materials* o *Parts explosion*, típica en las industrias del automóvil, en la construcción de naves espaciales y en campos similares. Estos sistemas trabajaban exclusivamente por lotes (*batch*).

Al aparecer los terminales de teclado, conectados al ordenador central mediante una línea telefónica, se empiezan a construir grandes **aplicaciones *on-line* transaccionales (OLTP)**. Los SGBD estaban íntimamente ligados al *software* de comunicaciones y de gestión de transacciones.

El *Data Base / Data Comunications*

IBM denominaba *Data Base/ Data Comunications* (DB/DC) el *software* de comunicaciones y de gestión de transacciones y de datos. Las aplicaciones típicas eran la reserva/compra de billetes a las compañías aéreas y de ferrocarriles y, un poco más tarde, las cuentas de clientes en el mundo bancario.

Aunque para escribir los **programas de aplicación** se utilizaban lenguajes de alto nivel como Cobol o PL/I, se disponía también de instrucciones y de subrutinas especializadas para tratar las BD que requerían que el programador conociese muchos detalles del diseño físico, y que hacían que la programación fuese muy compleja.

Puesto que los programas estaban relacionados con el nivel físico, se debían modificar continuamente cuando se hacían cambios en el diseño y la organización de la BD. La preocupación básica era maximizar el rendimiento: el tiempo de respuesta y las transacciones por segundo.

2.2. Los años ochenta: SGBD relacionales

Los **ordenadores *minis***, en primer lugar, y después los **ordenadores *micros***, extendieron la informática a prácticamente todas las empresas e instituciones.

Esto exigía que el desarrollo de aplicaciones fuese más sencillo. Los SGBD de los años setenta eran demasiado complejos e inflexibles, y sólo los podía utilizar un personal muy cualificado.

La aparición de los **SGBD relacionales*** supone un avance importante para facilitar la programación de aplicaciones con BD y para conseguir que los programas sean independientes de los aspectos físicos de la BD.

*** Oracle aparece en el año 1980.**

Todos estos factores hacen que se extienda el uso de los SGBD. La estandarización, en el año 1986, del **lenguaje SQL** produjo una auténtica explosión de los SGBD relacionales.

Los ordenadores personales

Durante los años ochenta aparecen y se extienden muy rápidamente los ordenadores personales. También surge *software* para estos equipos monousuario (por ejemplo, dBase y sus derivados, Access), con los cuales es muy fácil crear y utilizar conjuntos de datos, y que se denominan *personal data bases*. Notad que el hecho de denominar SGBD estos primeros sistemas para PC es un poco forzado, ya que no aceptaban estructuras complejas ni interrelaciones, ni podían ser utilizados en una red que sirviese simultáneamente a muchos usuarios de diferentes tipos. Sin embargo, algunos, con el tiempo, se han ido convirtiendo en auténticos SGBD.

2.3. Los años noventa: distribución, C/S y 4GL

Al acabar la década de los ochenta, los SGBD relacionales ya se utilizaban prácticamente en todas las empresas. A pesar de todo, hasta la mitad de los noventa, cuando se ha necesitado un rendimiento elevado se han seguido utilizando los SGBD prerrelacionales.

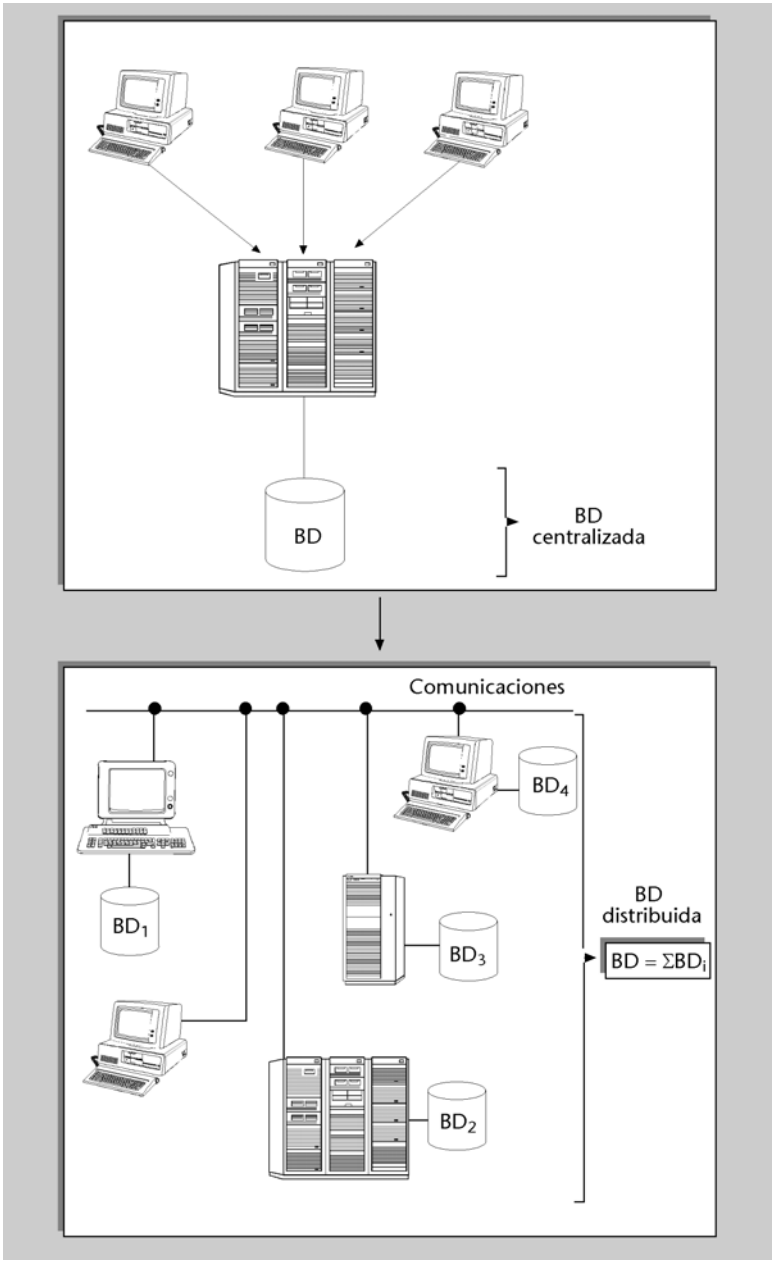
A finales de los ochenta y principios de los noventa, las empresas se han encontrado con el hecho de que sus departamentos han ido comprando ordenadores departamentales y personales, y han ido haciendo aplicaciones con BD. El resultado ha sido que en el seno de la empresa hay numerosas BD y varios SGBD de diferentes tipos o proveedores. Este fenómeno de **multiplicación de las BD y de los SGBD** se ha visto incrementado por la fiebre de las fusiones de empresas.

La necesidad de tener una visión global de la empresa y de interrelacionar diferentes aplicaciones que utilizan BD diferentes, junto con la facilidad que dan las redes para la intercomunicación entre ordenadores, ha conducido a los SGBD actuales, que permiten que un programa pueda trabajar con diferentes BD como si se tratase de una sola. Es lo que se conoce como **base de datos distribuida**.

Esta distribución ideal se consigue cuando las diferentes BD son soportadas por una misma marca de SGBD, es decir, cuando hay homogeneidad. Sin em-

bargo, esto no es tan sencillo si los SGBD son heterogéneos. En la actualidad, gracias principalmente a la estandarización del lenguaje SQL, los SGBD de marcas diferentes pueden darse servicio unos a otros y colaborar para dar servicio a un programa de aplicación. No obstante, en general, en los casos de heterogeneidad no se llega a poder dar en el programa que los utiliza la apariencia de que se trata de una única BD.

Figura 1



Además de esta distribución "impuesta", al querer tratar de forma integrada distintas BD preexistentes, también se puede hacer una distribución "deseada", diseñando una BD distribuida físicamente, y con ciertas partes replicadas en diferentes sistemas. Las razones básicas por las que interesa esta distribución son las siguientes:

1) Disponibilidad. La disponibilidad de un sistema con una BD distribuida puede ser más alta, porque si queda fuera de servicio uno de los sistemas, los de-

más seguirán funcionando. Si los datos residentes en el sistema no disponible están replicados en otro sistema, continuarán estando disponibles. En caso contrario, sólo estarán disponibles los datos de los demás sistemas.

2) Coste. Una BD distribuida puede reducir el coste. En el caso de un sistema centralizado, todos los equipos usuarios, que pueden estar distribuidos por distintas y lejanas áreas geográficas, están conectados al sistema central por medio de líneas de comunicación. El coste total de las comunicaciones se puede reducir haciendo que un usuario tenga más cerca los datos que utiliza con mayor frecuencia; por ejemplo, en un ordenador de su propia oficina o, incluso, en su ordenador personal.

La tecnología que se utiliza habitualmente para distribuir datos es la que se conoce como **entorno** (o arquitectura) **cliente/servidor (C/S)**. Todos los SGBD relacionales del mercado han sido adaptados a este entorno.

La idea del C/S es sencilla. Dos procesos diferentes, que se ejecutan en un mismo sistema o en sistemas separados, actúan de forma que uno tiene el papel de **cliente** o peticionario de un servicio, y el otro el de **servidor** o proveedor del servicio.

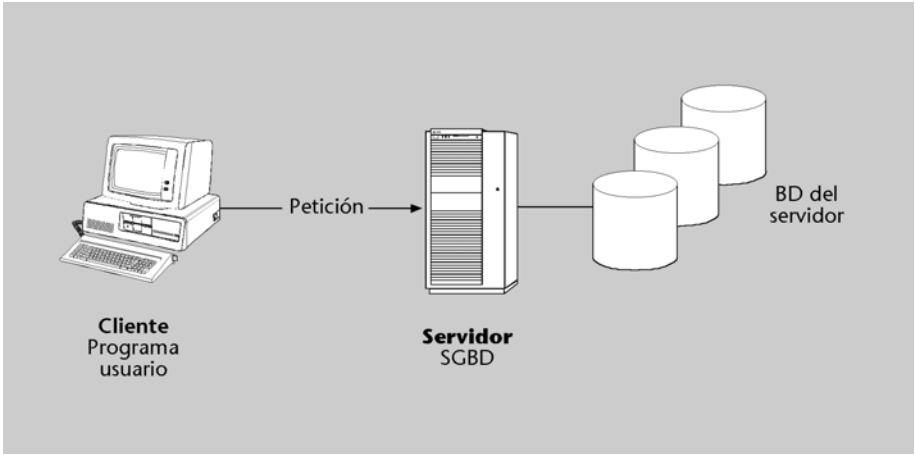
Por ejemplo, un programa de aplicación que un usuario ejecuta en su PC (que está conectado a una red) pide ciertos datos de una BD que reside en un equipo UNIX donde, a su vez, se ejecuta el SGBD relacional que la gestiona. El programa de aplicación es el cliente y el SGBD es el servidor.

Otros servicios

Notad que el servicio que da un servidor de un sistema C/S no tiene por qué estar relacionado con las BD; puede ser un servicio de impresión, de envío de un fax, etc., pero aquí nos interesan los servidores que son SGBD.

Un proceso cliente puede pedir servicios a varios servidores. Un servidor puede recibir peticiones de muchos clientes. En general, un proceso *A* que hace de cliente, pidiendo un servicio a otro proceso *B* puede hacer también de servidor de un servicio que le pida otro proceso *C* (o incluso el *B*, que en esta petición sería el cliente). Incluso el cliente y el servidor pueden residir en un mismo sistema.

Figura 2



La facilidad para disponer de distribución de datos no es la única razón, ni siquiera la básica, del gran éxito de los entornos C/S en los años noventa. Tal vez el motivo fundamental ha sido la flexibilidad para construir y hacer crecer la configuración informática global de la empresa, así como de hacer modificaciones en ella, mediante *hardware* y *software* muy estándar y barato.

El éxito de las BD, incluso en sistemas personales, ha llevado a la aparición de los ***Fourth Generation Languages* (4GL)**, lenguajes muy fáciles y potentes, especializados en el desarrollo de aplicaciones fundamentadas en BD. Proporcionan muchas facilidades en el momento de definir, generalmente de forma visual, diálogos para introducir, modificar y consultar datos en entornos C/S.

C/S, SQL y 4GL...

... son siglas de moda desde el principio de los años noventa en el mundo de los sistemas de información.

2.4. Tendencias actuales

Hoy día, los SGBD relacionales están en plena transformación para adaptarse a tres tecnologías de éxito reciente, fuertemente relacionadas: la multimedia, la de orientación a objetos (OO) e Internet y la web.

Los tipos de datos que se pueden definir en los SGBD relacionales de los años ochenta y noventa son muy limitados. La **incorporación de tecnologías multimedia –**imagen y sonido**–** en los SI hace necesario que los SGBD relacionales acepten atributos de estos tipos.

Nos puede interesar,...

... por ejemplo, tener en la entidad *alumno* un atributo *foto* tal que su valor sea una tira de bits muy larga, resultado de la digitalización de la fotografía del alumno.

Sin embargo, algunas aplicaciones no tienen suficiente con la incorporación de tipos especializados en multimedia. Necesitan tipos complejos que el desarrollador pueda definir a medida de la aplicación. En definitiva, se necesitan **tipos abstractos de datos: TAD**. Los SGBD más recientes ya incorporaban esta posibilidad, y abren un amplio mercado de TAD predefinidos o librerías de clases.

Esto nos lleva a la **orientación a objetos (OO)**. El éxito de la OO al final de los años ochenta, en el desarrollo de *software* básico, en las aplicaciones de ingeniería industrial y en la construcción de interfaces gráficas con los usuarios, ha hecho que durante la década de los noventa se extendiese en prácticamente todos los campos de la informática.

En los SI se inicia también la adopción, tímida de momento, de la OO. La utilización de lenguajes como C++ o Java requiere que los SGBD relacionales se adapten a ellos con interfaces adecuadas.

La rápida adopción de la **web** a los SI hace que los SGBD incorporen recursos para ser servidores de páginas web, como por ejemplo la inclusión de SQL en guiones HTML, SQL incorporado en Java, etc. Notad que en el mundo de la web son habituales los datos multimedia y la OO.

Durante estos últimos años se ha empezado a extender un tipo de aplicación de las BD denominado *Data Warehouse*, o **almacén de datos**, que también produce algunos cambios en los SGBD relacionales del mercado.

A lo largo de los años que han trabajado con BD de distintas aplicaciones, las empresas han ido acumulando gran cantidad de datos de todo tipo. Si estos datos se analizan convenientemente pueden dar información valiosa*.

*** Por ejemplo, la evolución del mercado en relación con la política de precios.**

Por lo tanto, se trata de mantener una gran BD con información proveniente de toda clase de aplicaciones de la empresa (e, incluso, de fuera). Los datos de este gran almacén, el *Data Warehouse*, se obtienen por una replicación más o menos elaborada de las que hay en las BD que se utilizan en el trabajo cotidiano de la empresa. Estos almacenes de datos se utilizan exclusivamente para hacer consultas, de forma especial para que lleven a cabo estudios* los analistas financieros, los analistas de mercado, etc.

*** Con frecuencia se trata de estadísticas multidimensionales.**

Actualmente, los SGBD se adaptan a este tipo de aplicación, incorporando, por ejemplo, herramientas como las siguientes:

a) La creación y el mantenimiento de réplicas, con una cierta elaboración de los datos.

b) La consolidación de datos de orígenes diferentes.

c) La creación de estructuras físicas que soporten eficientemente el análisis multidimensional.

3. Objetivos y servicios de los SGBD

Los SGBD que actualmente están en el mercado pretenden satisfacer un conjunto de objetivos directamente deducibles de lo que hemos explicado hasta ahora. A continuación los comentaremos, pero sin entrar en detalles que ya se verán más adelante en otras asignaturas.

3.1. Consultas no predefinidas y complejas

El objetivo fundamental de los SGBD es permitir que se hagan **consultas no predefinidas** (*ad hoc*) y **complejas**.

Consultas que afectan a más de una entidad tipo

- Se quiere conocer el número de alumnos de más de veinticinco años y con nota media superior a siete que están matriculados actualmente en la asignatura *Bases de datos I*.
- De cada alumno matriculado en menos de tres asignaturas, se quiere obtener el nombre, el número de matrícula, el nombre de las asignaturas y el nombre de profesores de estas asignaturas.

Los usuarios podrán hacer consultas de cualquier tipo y complejidad directamente al SGBD. El SGBD tendrá que responder inmediatamente sin que estas consultas estén preestablecidas; es decir, sin que se tenga que escribir, compilar y ejecutar un programa específico para cada consulta.

Ficheros tradicionales

En los ficheros tradicionales, cada vez que se quería hacer una consulta se tenía que escribir un programa a medida.

El usuario debe formular la consulta con un **lenguaje sencillo** (que se quede, obviamente, en el nivel lógico), que el sistema debe interpretar directamente. Sin embargo, esto no significa que no se puedan escribir programas con consultas incorporadas (por ejemplo, para procesos repetitivos).

La solución estándar para alcanzar este doble objetivo (consultas no predefinidas y complejas) es el **lenguaje SQL**, que explicaremos en otro módulo didáctico.

3.2. Flexibilidad e independencia

La complejidad de las BD y la necesidad de irlas adaptando a la evolución del SI hacen que un objetivo básico de los SGBD sea dar **flexibilidad a los cambios**.

Interesa obtener la **máxima independencia** posible entre los datos y los procesos usuarios para que se pueda llevar a cabo todo tipo de cambios tecnológicos

y variaciones en la descripción de la BD, sin que se deban modificar los programas de aplicación ya escritos ni cambiar la forma de escribir las consultas (o actualizaciones) directas.

Para conseguir esta independencia, tanto los usuarios que hacen consultas (o actualizaciones) directas como los profesionales informáticos que escriben programas que las llevan incorporadas, deben poder desconocer las características físicas de la BD con que trabajan. No necesitan saber nada sobre el soporte físico, ni estar al corriente de qué SO se utiliza, qué índices hay, la compresión o no compresión de datos, etc.

De este modo, se pueden hacer cambios de tecnología y cambios físicos para mejorar el rendimiento sin afectar a nadie. Este tipo de independencia recibe el nombre de **independencia física de los datos**.

En el mundo de los ficheros ya había independencia física en un cierto grado, pero en el mundo de las BD acostumbra a ser mucho mayor.

Sin embargo, con la independencia física no tenemos suficiente. También queremos que los usuarios (los programadores de aplicaciones o los usuarios directos) no tengan que hacer cambios cuando se modifica la descripción lógica o el esquema de la BD (por ejemplo, cuando se añaden/suprimen entidades o interrelaciones, atributos, etc.

Y todavía más: queremos que diferentes procesos usuarios puedan tener diferentes visiones lógicas de una misma BD, y que estas visiones se puedan mantener lo más independientes posibles de la BD, y entre ellas mismas. Este tipo de independencia se denomina **independencia lógica de los datos**, y da flexibilidad y elasticidad a los cambios lógicos.

Independencia lógica de los datos

Por ejemplo, el hecho de suprimir el atributo *fecha de nacimiento* de la entidad *alumno* y añadir otra entidad *aula* no debería afectar a ninguno de los programas existentes que no utilicen el atributo *fecha de nacimiento*.

Las diferentes visiones lógicas de una BD se verán en el apartado 4 de esta unidad didáctica.

Ejemplos de independencia lógica

1) El personal administrativo de secretaría podría tener una visión de la entidad *alumno* sin que fuese necesario que existiese el atributo *nota*. Sin embargo, los usuarios profesores (o los programas dirigidos a ellos) podrían tener una visión en la que existiese el atributo *nota* pero no el atributo *fecha de pago*.

2) Decidimos ampliar la longitud del atributo *nombre* y lo aumentamos de treinta a cincuenta caracteres, pero no sería necesario modificar los programas que ya tenemos escritos si no nos importa que los valores obtenidos tengan sólo los primeros treinta caracteres del nombre.

Independencia lógica

Los sistemas de gestión de ficheros tradicionales no dan ninguna independencia lógica. Los SGBD sí que la dan; uno de sus objetivos es conseguir la máxima posible, pero dan menos de lo que sería deseable.

3.3. Problemas de la redundancia

En el mundo de los ficheros tradicionales, cada aplicación utilizaba su fichero. Sin embargo, puesto que se daba mucha coincidencia de datos entre aplicacio-

nes, se producía también mucha redundancia entre los ficheros. Ya hemos dicho que uno de los objetivos de los SGBD es **facilitar la eliminación de la redundancia**.

Consultad el apartado 1 de esta unidad didáctica.

Seguramente pensáis que el problema de la redundancia es el espacio perdido. Antiguamente, cuando el precio del *byte* de disco era muy elevado, esto era un problema grave, pero actualmente prácticamente nunca lo es. ¿Qué problema hay, entonces? Simplemente, lo que todos hemos sufrido más de una vez; si tenemos algo apuntado en dos lugares diferentes no pasará demasiado tiempo hasta que las dos anotaciones dejen de ser coherentes, porque habremos modificado la anotación en uno de los lugares y nos habremos olvidado de hacerlo en el otro.

¿Por qué queremos evitar la redundancia? ¿Qué problema nos comporta?

Así pues, el verdadero problema es el grave riesgo de inconsistencia o incoherencia de los datos; es decir, la **pérdida de integridad** que las actualizaciones pueden provocar cuando existe redundancia.

Por lo tanto, convendría evitar la redundancia. En principio, nos conviene hacer que un dato sólo figure una vez en la BD. Sin embargo, esto no siempre será cierto. Por ejemplo, para representar una interrelación entre dos entidades, se suele repetir un mismo atributo en las dos, para que una haga referencia a la otra.

Otro ejemplo podría ser el disponer de réplicas de los datos por razones de fiabilidad, disponibilidad o costes de comunicaciones.

Para recordar lo que se ha dicho sobre datos replicados, consultad el subapartado 2.3 de esta unidad didáctica.

El SGBD debe permitir que el diseñador defina datos redundantes, pero entonces tendría que ser el mismo SGBD el que hiciese automáticamente la **actualización de los datos** en todos los lugares donde estuviesen repetidos.

La duplicación de datos es el tipo de redundancia más habitual, pero también tenemos redundancia cuando guardamos en la BD datos derivados (o calculados) a partir de otros datos de la misma BD. De este modo podemos responder rápidamente a consultas globales, ya que nos ahorramos la lectura de gran cantidad de registros.

Datos derivados

Es frecuente tener datos numéricos acumulados o agregados: el importe total de todas las matrículas hechas hasta hoy, el número medio de alumnos por asignatura, el saldo de la caja de la oficina, etc.

En los casos de datos derivados, para que el resultado del cálculo se mantenga consistente con los datos elementales, es necesario rehacer el cálculo cada vez que éstos se modifican. El usuario (ya sea programador o no) puede olvidarse de hacer el nuevo cálculo; por ello convendrá que el mismo SGBD lo haga automáticamente.

3.4. Integridad de los datos

Nos interesará que los SGBD aseguren el **mantenimiento de la calidad de los datos** en cualquier circunstancia. Acabamos de ver que la redundancia puede provocar pérdida de integridad de los datos, pero no es la única causa posible. Se podría perder la corrección o la consistencia de los datos por muchas otras razones: errores de programas, errores de operación humana, avería de disco, transacciones incompletas por corte de alimentación eléctrica, etc.

En el subapartado anterior hemos visto que podremos decir al SGBD que nos lleve el control de las actualizaciones en el caso de las redundancias, para garantizar la integridad. Del mismo modo, podremos darle otras **reglas de integridad** –o restricciones– para que asegure que los programas las cumplen cuando efectúan las actualizaciones.

Reglas de integridad

Por ejemplo, podremos declarar que el atributo *DNI* debe ser clave o que la *fecha de nacimiento* debe ser una fecha correcta y, además, se debe cumplir que el alumno no pueda tener menos de dieciocho años ni más de noventa y nueve, o que el número de alumnos matriculados de una asignatura no sea superior a veintisiete, etc.

Cuando el SGBD detecte que un programa quiere hacer una operación que va contra las reglas establecidas al definir la BD, no se lo deberá permitir, y le tendrá que devolver un estado de error.

Al diseñar una BD para un SI concreto y escribir su esquema, no sólo definiremos los datos, sino también las reglas de integridad que queremos que el SGBD haga cumplir.

Aparte de las reglas de integridad que el diseñador de la BD puede definir y que el SGBD entenderá y hará cumplir, el mismo SGBD tiene reglas de integridad inherentes al modelo de datos que utiliza y que siempre se cumplirán. Son las denominadas **reglas de integridad del modelo**. Las reglas definibles por parte del usuario son las **reglas de integridad del usuario**. El concepto de *integridad de los datos* va más allá de prevenir que los programas usuarios almacenen datos incorrectos. En casos de errores o desastres, también podríamos perder la integridad de los datos. El SGBD nos debe dar las herramientas para reconstruir o restaurar los datos estropeados.

Reglas de integridad del modelo

Por ejemplo, un SGBD relacional nunca aceptará que una tabla tenga filas duplicadas, un SGBD jerárquico nunca aceptará que una entidad tipo esté definida como hija de dos entidades tipo diferentes, etc.

Los **procesos de restauración** (*restore* o *recovery*) de los que todo SGBD dispone pueden reconstruir la BD y darle el estado consistente y correcto anterior al incidente. Esto se acostumbra a hacer gracias a la obtención de copias periódicas de los datos (se denominan *copias de seguridad* o *back-up*) y mediante el mantenimiento continuo de un diario (*log*) donde el SGBD va anotando todas las escrituras que se hacen en la BD.

3.5. Concurrencia de usuarios

Usuarios concurrentes

Actualmente ya no son raros los SI que tienen, en un instante determinado, miles de sesiones de usuario abiertas simultáneamente. No hace falta pensar en los típicos sistemas de los consorcios de compañías aéreas o casos similares, sino en los servidores de páginas web.

Un objetivo fundamental de los SGBD es permitir que varios usuarios puedan acceder concurrentemente a la misma BD.

Cuando los accesos concurrentes son todos de lectura (es decir, cuando la BD sólo se consulta), el problema que se produce es simplemente de rendimiento, causado por las limitaciones de los soportes de que se dispone: pocos mecanismos de acceso independientes, movimiento del brazo y del giro del disco demasiado lentos, *buffers* locales demasiado pequeños, etc.

Cuando un usuario o más de uno están actualizando los datos, se pueden producir **problemas de interferencia** que tengan como consecuencia la obtención de datos erróneos y la pérdida de integridad de la BD. 

Para tratar los accesos concurrentes, los SGBD utilizan el concepto de transacción de BD, concepto de especial utilidad para todo aquello que hace referencia a la integridad de los datos, como veremos a continuación.

Denominamos **transacción de BD** o, simplemente, **transacción** un conjunto de operaciones simples que se ejecutan como una unidad. Los SGBD deben conseguir que el conjunto de operaciones de una transacción nunca se ejecute parcialmente. O se ejecutan todas, o no se ejecuta ninguna.

Ejemplos de transacciones

1) Imaginemos un programa pensado para llevar a cabo la operación de transferencia de dinero de una cuenta X a otra Y. Supongamos que la transferencia efectúa dos operaciones: en primer lugar, el cargo a X y después, el abono a Y. Este programa se debe ejecutar de forma que se hagan las dos operaciones o ninguna, ya que si por cualquier razón (por ejemplo, por interrupción del flujo eléctrico) el programa ejecutase sólo el cargo de dinero a X sin abonarlos a Y, la BD quedaría en un estado incorrecto. Queremos que la ejecución de este programa sea tratada por el SGBD como una transacción de BD.

2) Otro ejemplo de programa que querríamos que tuviera un comportamiento de transacción podría ser el que aumentara el 30% de la nota de todos los alumnos. Si sólo aumentara la nota a unos cuantos alumnos, la BD quedaría incorrecta.

Para indicar al SGBD que damos por acabada la ejecución de la transacción, el programa utilizará la operación de `COMMIT`. Si el programa no puede acabar normalmente (es decir, si el conjunto de operaciones se ha hecho sólo de forma parcial), el SGBD tendrá que deshacer todo lo que la transacción ya haya hecho. Esta operación se denomina `ROLLBACK`.

Acabamos de observar la utilidad del concepto de *transacción* para el mantenimiento de la integridad de los datos en caso de interrupción de un conjunto de operaciones lógicamente unitario. Sin embargo, entre transacciones que se ejecutan concurrentemente se pueden producir problemas de interferencia que hagan obtener resultados erróneos o que comporten la pérdida de la integridad de los datos.

Consecuencias de la interferencia entre transacciones

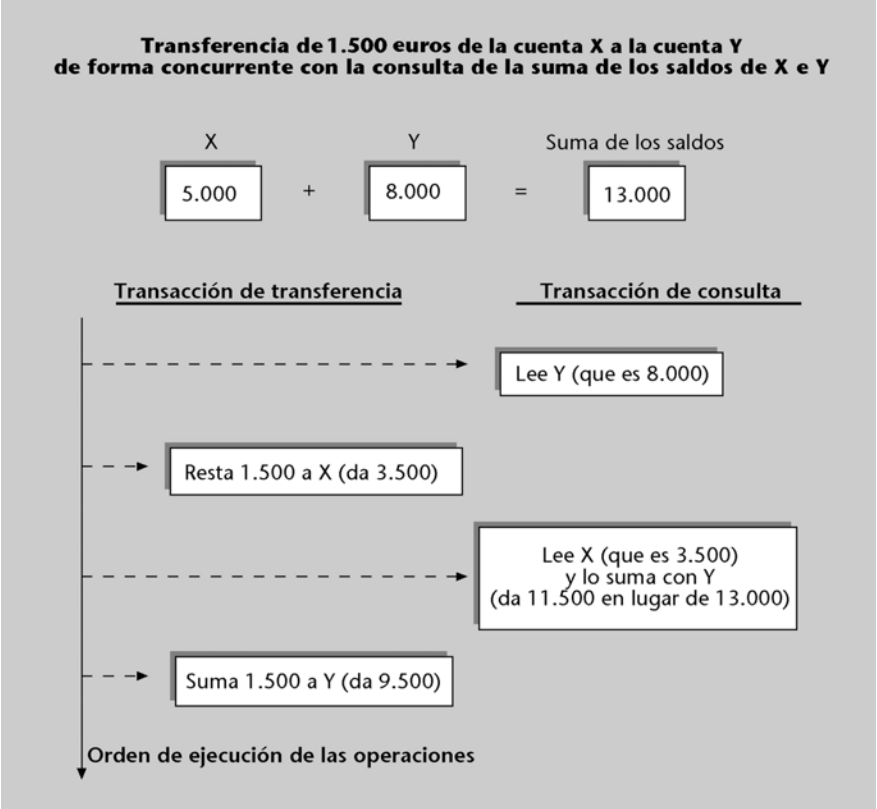
1) Imaginemos que una transacción que transfiere dinero de X a Y se ejecuta concurrentemente con una transacción que observa el saldo de las cuentas Y y X, en este orden, y nos muestra su suma. Si la ejecución de forma concurrente de las dos transacciones casualmente

es tal que la transferencia se ejecuta entre la ejecución de las dos lecturas de la transacción de suma, puede producir resultados incorrectos. Además, si los decide escribir en la BD, ésta quedará inconsistente (consultad la figura 3).

2) Si simultáneamente con el generoso programa que aumenta la nota de los alumnos en un 30%, se ejecuta un programa que determina la nota media de todos los alumnos de una determinada asignatura, se podrá encontrar a alumnos ya gratificados y a otros no gratificados, algo que producirá resultados erróneos.

Estos son sólo dos ejemplos de las diferentes consecuencias negativas que puede tener la interferencia entre transacciones en la integridad de la BD y en la corrección del resultado de las consultas.

Figura 3



Nos interesará que el SGBD ejecute las transacciones de forma que no se interfieran; es decir, que queden aisladas unas de otras. Para conseguir que las transacciones se ejecuten como si estuviesen aisladas, los SGBD utilizan distintas técnicas. La más conocida es el **bloqueo** (*lock*).

El bloqueo de unos datos en beneficio de una transacción consiste en poner limitaciones a los accesos que las demás transacciones podrán hacer a estos datos.

Bloqueos en la transferencia de dinero

Por ejemplo, si la transacción de transferencia de dinero modifica el saldo de la cuenta X, el SGBD bloquea esta cuenta de forma que, cuando la transacción de suma quiera leerla, tenga que esperar a que la transacción de transferencia acabe; esto se debe al hecho de que al acabar una transacción, cuando se efectúan las operaciones `COMMIT` o `ROLLBACK` se liberan los objetos que tenía bloqueados.

Cuando se provocan bloqueos, se producen esperas, retenciones y, en consecuencia, el sistema es más lento. Los SGBD se esfuerzan en minimizar estos efectos negativos.

Recordad que esta asignatura es sólo introductoria y que más adelante, en otras asignaturas, estudiaréis con más detalle los temas que, como el de la concurrencia, aquí sólo introducimos.

3.6. Seguridad

El término *seguridad* se ha utilizado en diferentes sentidos a lo largo de la historia de la informática.

Actualmente, en el campo de los SGBD, el término *seguridad* se suele utilizar para hacer referencia a los temas relativos a la confidencialidad, las autorizaciones, los derechos de acceso, etc.

Estas cuestiones siempre han sido importantes en los SI militares, las agencias de información y en ámbitos similares, pero durante los años noventa han ido adquiriendo importancia en cualquier SI donde se almacenen datos sobre personas. Recordad que en el Estado español tenemos una ley*, que exige la protección de la confidencialidad de estos datos.

*** Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal. (BOE núm. 298, de 12/12/1999, págs. 43088-43099).**

Los SGBD permiten definir **autorizaciones** o derechos de acceso a diferentes niveles: al nivel global de toda la BD, al nivel entidad y al nivel atributo.

Derechos de acceso

Por ejemplo, el usuario SECRE3 podría tener autorización para consultar y modificar todas las entidades de la BD, excepto el valor del atributo *nota de los alumnos*, y no estar autorizado a hacer ningún tipo de supresión o inserción.

Estos mecanismos de seguridad requieren que el usuario se pueda identificar. Se acostumbra a utilizar códigos de usuarios (y grupos de usuarios) acompañados de contraseñas (*passwords*), pero también se utilizan tarjetas magnéticas, identificación por reconocimiento de la voz, etc.

Nos puede interesar almacenar la información con una codificación secreta; es decir, con **técnicas de encriptación** (como mínimo se deberían encriptar las contraseñas). Muchos de los SGBD actuales tienen prevista la encriptación.

Prácticamente todos los SGBD del mercado dan una gran variedad de herramientas para la vigilancia y la administración de la seguridad. Los hay que, incluso, tienen opciones (con precio separado) para los SI con unas exigencias altísimas, como por ejemplo los militares.

3.7. Otros objetivos

Para recordar las tendencias actuales sobre SGBD, podéis revisar el subapartado 2.4. de esta unidad didáctica.

Acabamos de ver los objetivos fundamentales de los SGBD actuales. Sin embargo, a medida que los SGBD evolucionan, se imponen nuevos objetivos adaptados a las nuevas necesidades y las nuevas tecnologías. Como ya hemos visto, en estos momentos podríamos citar como objetivos nuevos o recientes los siguientes:

1) Servir eficientemente los *Data Warehouse*.

2) Adaptarse al desarrollo orientado a objetos.

3) Incorporar el tiempo como un elemento de caracterización de la información.

4) Adaptarse al mundo de Internet.

4. Arquitectura de los SGBD

4.1. Esquemas y niveles

Para trabajar con nuestras BD, los SGBD necesitan conocer su estructura (qué entidades tipo habrá, qué atributos tendrán, etc.).

Los SGBD necesitan que les demos una descripción o definición de la BD. Esta descripción recibe el nombre de **esquema de la BD**, y los SGBD la tendrán continuamente a su alcance.

El esquema de la BD es un elemento fundamental de la arquitectura de un SGBD que permite independizar el SGBD de la BD; de este modo, se puede cambiar el diseño de la BD (su esquema) sin tener que hacer ningún cambio en el SGBD.

Anteriormente, ya hemos hablado de la distinción entre dos niveles de representación informática: el nivel lógico y el físico.

El **nivel lógico** nos oculta los detalles de cómo se almacenan los datos, cómo se mantienen y cómo se accede físicamente a ellos. En este nivel sólo se habla de entidades, atributos y reglas de integridad.

Por cuestiones de rendimiento, nos podrá interesar describir elementos de **nivel físico** como, por ejemplo, qué índices tendremos y qué características presentarán, cómo y dónde (en qué espacio físico) queremos que se agrupen físicamente los registros, de qué tamaño deben ser las páginas, etc.

En el periodo 1975-1982, ANSI intentaba establecer las bases para crear estándares en el campo de las BD. El comité conocido como ANSI/SPARC recomendó que la arquitectura de los SGBD previese tres niveles de descripción de la BD, no sólo dos*.

*** De hecho, en el año 1971, el comité CODASYL ya había propuesto los tres niveles de esquemas.**

De acuerdo con la arquitectura ANSI/SPARC, debía haber tres niveles de esquemas (tres niveles de abstracción). La idea básica de ANSI/SPARC consistía en descomponer el nivel lógico en dos: el **nivel externo** y el **nivel conceptual**. Denominábamos **nivel interno** lo que aquí hemos denominado *nivel físico*.

De este modo, de acuerdo con ANSI/SPARC, habría los tres niveles de esquemas que mencionamos a continuación:

Nota

Es preciso ir con cuidado para no confundir los niveles que se describen aquí con los descritos en el caso de los ficheros, aunque reciban el mismo nombre.

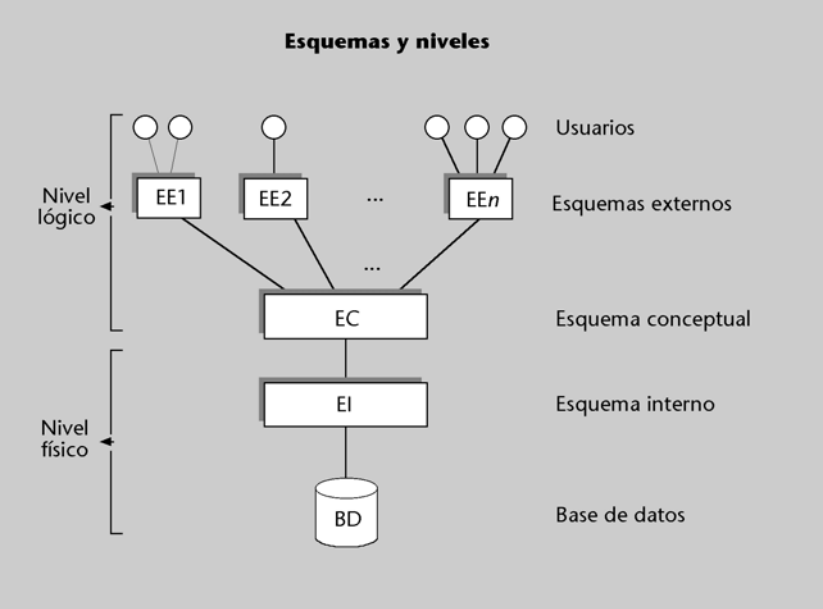
En el caso de las BD, el esquema interno corresponde a la parte física, y el externo a la lógica; en el caso de los ficheros, sucede lo contrario.

a) En el nivel externo se sitúan las diferentes visiones lógicas que los procesos usuarios (programas de aplicación y usuarios directos) tendrán de las partes de la BD que utilizarán. Estas visiones se denominan **esquemas externos**.

b) En el nivel conceptual hay una sola descripción lógica básica, única y global, que denominamos **esquema conceptual**, y que sirve de referencia para el resto de los esquemas.

c) En el nivel físico hay una sola descripción física, que denominamos **esquema interno**.

Figura 4



En el **esquema conceptual** se describirán las entidades tipo, sus atributos, las interrelaciones y las restricciones o reglas de integridad.

El esquema conceptual corresponde a las necesidades del conjunto de la empresa o del SI, por lo que se escribirá de forma centralizada durante el denominado *diseño lógico* de la BD.

Sin embargo, cada aplicación podrá tener su visión particular, y seguramente parcial, del esquema conceptual. Los usuarios (programas o usuarios directos) verán la BD mediante esquemas externos apropiados a sus necesidades. Estos esquemas se pueden considerar redefiniciones del esquema conceptual, con las partes y los términos que convengan para las necesidades de las aplicaciones (o grupos de aplicaciones). Algunos sistemas los denominan *subesquemas*.

Al definir un **esquema externo**, se citarán sólo aquellos atributos y aquellas entidades que interesen; los podremos redenominar, podremos definir datos derivados o redefinir una entidad para que las aplicaciones que utilizan este esquema externo crean que son dos, definir combinaciones de entidades para que parezcan una sola, etc.

Ejemplo de esquema externo

Imaginemos una BD que en el esquema conceptual tiene definida, entre muchas otras, una entidad *alumno* con los siguientes atributos: *numatri, nombre, apellido, numDNI, direccion, fechanac, telefono*. Sin embargo, nos puede interesar que unos determinados programas o usuarios vean la BD formada de acuerdo con un esquema externo que tenga definidas dos entidades, denominadas *estudiante* y *persona*.

a) La entidad *estudiante* podría tener definido el atributo *numero-matricula* (definido como derivable directamente de *numatri*), el atributo *nombre-pila* (de *nombre*), el atributo *apellido* y el atributo *DNI* (de *numDNI*).

b) La entidad *persona* podría tener el atributo *DNI* (obtenido de *numDNI*), el atributo *nombre* (formado por la concatenación de *nombre* y *apellido*), el atributo *direccion* y el atributo *edad* (que deriva dinámicamente de *fechanac*).

El **esquema interno** o físico contendrá la descripción de la organización física de la BD: caminos de acceso (índices, *hashing*, apuntadores, etc.), codificación de los datos, gestión del espacio, tamaño de la página, etc.

El esquema de nivel interno responde a las cuestiones de rendimiento (espacio y tiempo) planteadas al hacer el diseño físico de la BD y al ajustarlo* posteriormente a las necesidades cambiantes.

*** En inglés, el ajuste se conoce con el nombre de *tuning*.**

De acuerdo con la arquitectura ANSI/SPARC, para crear una BD hace falta definir previamente su esquema conceptual, definir como mínimo un esquema externo y, de forma eventual, definir su esquema interno. Si este último esquema no se define, el mismo SGBD tendrá que decidir los detalles de la organización física. El SGBD se encargará de hacer las correspondencias (*mappings*) entre los tres niveles de esquemas.

¿Qué hay que hacer para crear una BD?

Esquemas y niveles en los SGBD relacionales

En los SGBD relacionales (es decir, en el mundo de SQL) se utiliza una terminología ligeramente diferente. No se separan de forma clara tres niveles de descripción. Se habla de un solo esquema ***–schema–***, pero en su interior se incluyen descripciones de los tres niveles. En el *schema* se describen los elementos de aquello que en la arquitectura ANSI/SPARC se denomina *esquema conceptua*l (entidades tipo, atributos y restricciones) y las vistas *–view–*, que corresponden aproximadamente a los esquemas externos.

El modelo relacional en que está inspirado SQL se limita al mundo lógico. Por ello, el estándar ANSI-ISO de SQL no habla en absoluto del mundo físico o interno; lo deja en manos de los SGBD relacionales del mercado. Sin embargo, estos SGBD proporcionan la posibilidad de incluir dentro del *schema* descripciones de estructuras y características físicas (índice, *tablespace*, *cluster*, espacios para excesos, etc.).

4.2. Independencia de los datos

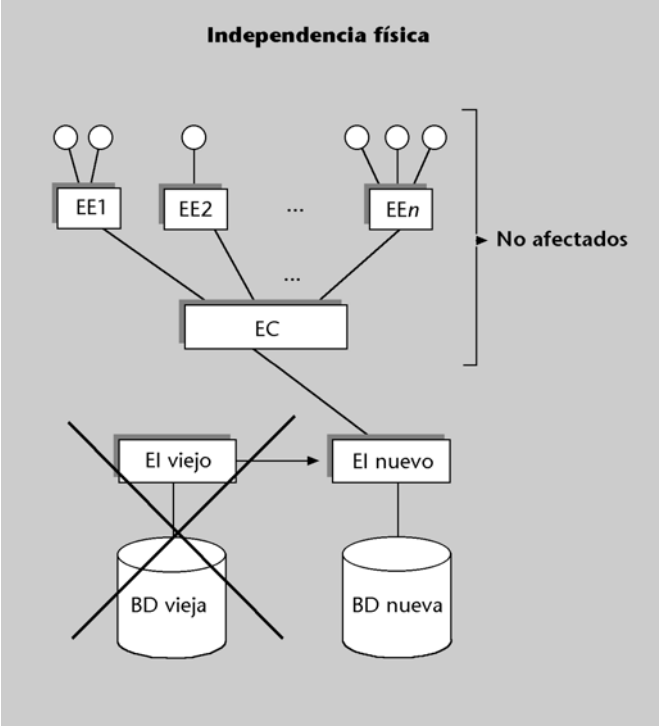
En este subapartado veremos cómo la arquitectura de tres niveles que acabamos de presentar nos proporciona los dos tipos de independencia de los datos: la física y la lógica.

Los dos tipos de independencia de los datos se han explicado en el subapartado 3.2 de esta unidad didáctica.

Hay **independencia física** cuando los cambios en la organización física de la BD no afectan al mundo exterior (es decir, los programas usuarios o los usuarios directos).

De acuerdo con la arquitectura ANSI/SPARC, habrá independencia física cuando los cambios en el esquema interno no afecten al esquema conceptual ni a los esquemas externos.

Figura 5



Es obvio que cuando cambiemos unos datos de un soporte a otro, o los cambiemos de lugar dentro de un soporte, no se verán afectados ni los programas de aplicación ni los usuarios directos, ya que no se modificará el esquema conceptual ni el externo. Sin embargo, tampoco tendrían que verse afectados si cambiásemos, por ejemplo, el método de acceso a unos registros determinados*, el formato o la codificación, etc. Ninguno de estos casos debería afectar al mundo exterior, sino sólo a la BD física, el esquema interno, etc.

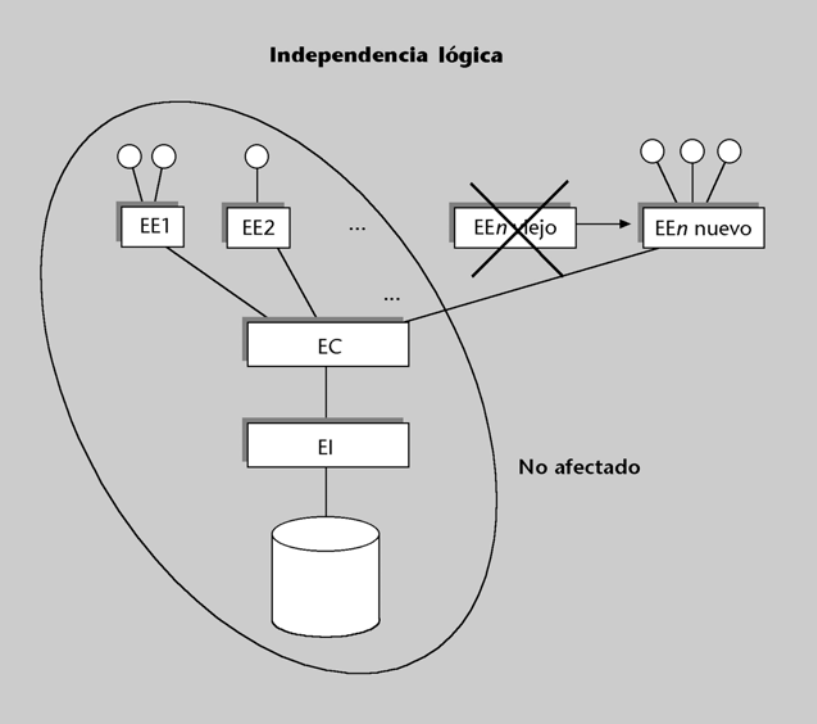
*** Por ejemplo, eliminando un índice en árbol-B o sustituyéndolo por un *hashing*.**

Si hay independencia física de los datos, lo único que variará al cambiar el esquema interno son las correspondencias entre el esquema conceptual y el interno. Obviamente, la mayoría de los cambios del esquema interno obligarán a rehacer la BD real (la física).

Hay **independencia lógica** cuando los usuarios* no se ven afectados por los cambios en el nivel lógico.

*** Programas de aplicación o usuarios directos.**

Figura 6



Dados los dos niveles lógicos de la arquitectura ANSI/SPARC, diferenciaremos las dos situaciones siguientes:

1) Cambios en el esquema conceptual. Un cambio de este tipo no afectará a los esquemas externos que no hagan referencia a las entidades o a los atributos modificados.

Si eliminamos...

... el atributo *apellido*, por ejemplo, no se verán afectados los esquemas externos (ni los usuarios) que no hagan referencia a este atributo. Si se alarga el atributo *dirección* al esquema conceptual, no será necesario modificar el esquema externo donde se ha definido la *dirección* y obviamente, tampoco resultarán afectados los programas y los usuarios que la utilicen.

2) Cambios en los esquemas externos. Efectuar cambios en un esquema externo afectará a los usuarios que utilicen los elementos modificados. Sin embargo, no debería afectar a los demás usuarios ni al esquema conceptual, y tampoco, en consecuencia, al esquema interno y a la BD física.

Usuarios no afectados por los cambios

Notad que no todos los cambios de elementos de un esquema externo afectarán a sus usuarios. Veamos un ejemplo de ello: antes hemos visto que cuando eliminábamos el atributo *apellido* del esquema conceptual, debíamos modificar el esquema externo donde definíamos *nombre*, porque allí estaba definido como concatenación de *nombre* y *apellido*. Pues bien, un programa que utilizase el atributo *nombre* no se vería afectado si modificásemos el esquema externo de modo que *nombre* fuese la concatenación de *nombre* y una cadena constante (por ejemplo, toda en blanco). Como resultado, habría desaparecido el apellido de *nombre*, sin que hubiera sido necesario modificar el programa.

Los SGBD actuales proporcionan bastante independencia lógica, pero menos de la que haría falta, ya que las exigencias de cambios constantes en el SI piden grados muy elevados de flexibilidad. Los sistemas de ficheros tradicionales, en cambio, no ofrecen ninguna independencia lógica.

4.3. Flujo de datos y de control

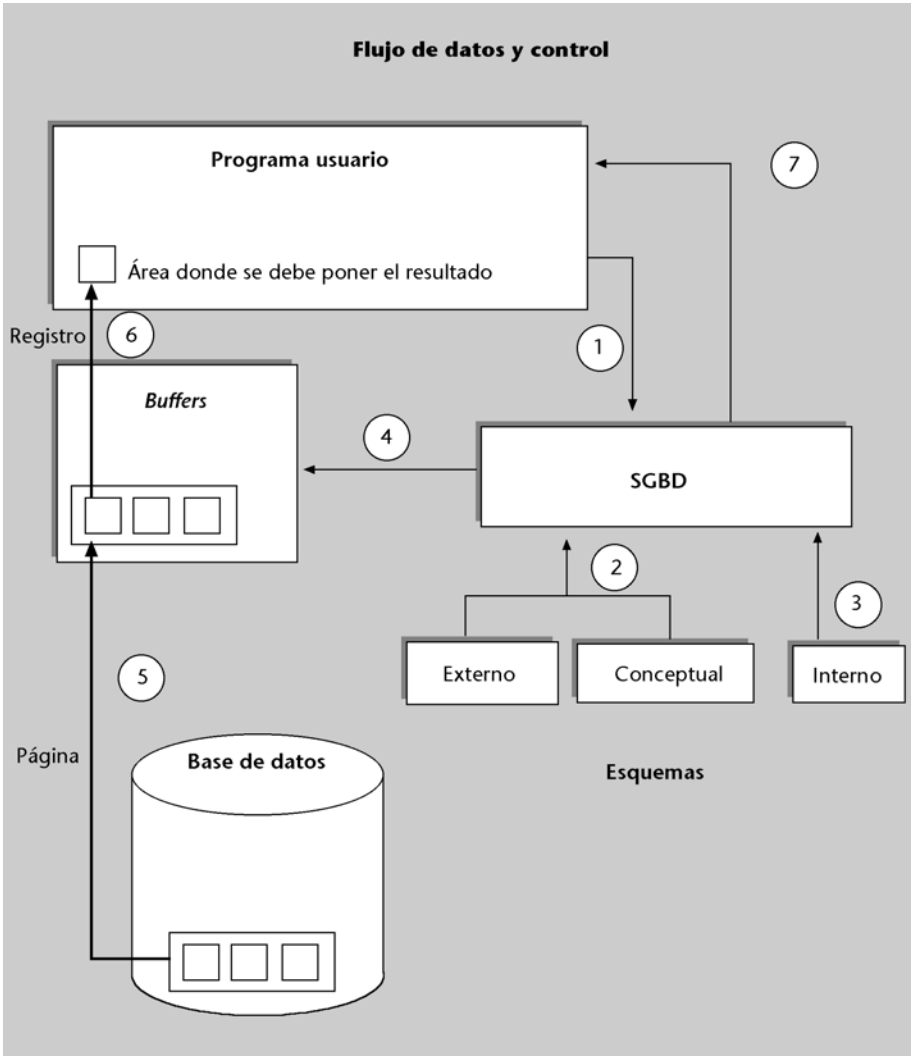
Para entender el funcionamiento de un SGBD, a continuación veremos los principales pasos de la ejecución de una consulta sometida al SGBD por un programa de aplicación. Explicaremos las líneas generales del flujo de datos y de control entre el SGBD, los programas de usuario y la BD.

Recordad que el SGBD, con la ayuda del SO, lee páginas (bloques) de los soportes donde está almacenada la BD física, y las lleva a un área de *buffers* o memorias caché en la memoria principal. El SGBD pasa registros desde los *buffers* hacia el área de trabajo del mismo programa.

Supongamos que la consulta pide los datos del alumno que tiene un determinado DNI. Por lo tanto, la respuesta que el programa obtendrá será un solo registro y lo recibirá dentro de un área de trabajo propia*.

*** Por ejemplo, una variable con estructura de tupla.**

Figura 7



Ejecución de una consulta

En la figura vemos representada la BD, los tres niveles de esquemas, el área de los *buffers*, el SGBD y el programa de aplicación que le hace la consulta.

El proceso que se sigue es el siguiente:

a) Empieza con una llamada (1) del programa al SGBD, en la que se le envía la operación de consulta. El SGBD debe verificar que la sintaxis de la operación

recibida sea correcta, que el usuario del programa esté autorizado a hacerla, etc. Para poder llevar a cabo todo esto, el SGBD se basa (2) en el esquema externo con el que trabaja el programa y en el esquema conceptual.

b) Si la consulta es válida, el SGBD determina, consultando el esquema interno (3), qué mecanismo debe seguir para responderla. Ya sabemos que el programa usuario no dice nada respecto a cómo se debe hacer físicamente la consulta. Es el SGBD el que lo debe determinar. Casi siempre hay varias formas y diferentes caminos para responder a una consulta*. Supongamos que ha elegido aplicar un *hashing* al valor del DNI, que es el parámetro de la consulta, y el resultado es la dirección de la página donde se encuentra (entre muchos otros) el registro del alumno buscado.

*** Por ejemplo, siempre tiene la posibilidad de hacer una búsqueda secuencial.**

c) Cuando ya se sabe cuál es la página, el SGBD comprobará (4) si por suerte esta página ya se encuentra en aquel momento en el área de los *buffers* (tal vez como resultado de una consulta anterior de este usuario o de otro). Si no está, el SGBD, con la ayuda del SO, la busca en disco y la carga en los *buffers* (5). Si ya está, se ahorra el acceso a disco.

d) Ahora, la página deseada ya está en la memoria principal. El SGBD extrae, de entre los distintos registros que la página puede contener, el registro buscado, e interpreta la codificación y el resultado según lo que diga el esquema interno.

e) El SGBD aplica a los datos las eventuales transformaciones lógicas que implica el esquema externo (tal vez cortando la dirección por la derecha) y las lleva al área de trabajo del programa (6).

Diferencias entre SGBD

Aunque entre diferentes SGBD puede haber enormes diferencias de funcionamiento, suelen seguir el esquema general que acabamos de explicar.

f) A continuación, el SGBD retorna el control al programa (7) y da por terminada la ejecución de la consulta.

5. Modelos de BD

Una BD es una representación de la realidad (de la parte de la realidad que nos interesa en nuestro SI). Dicho de otro modo, una BD se puede considerar un modelo de la realidad. El componente fundamental utilizado para modelar en un SGBD relacional son las tablas (denominadas *relaciones* en el mundo teórico). Sin embargo, en otros tipos de SGBD se utilizan otros componentes.

Las tablas o relaciones se estudiarán en la unidad didáctica "El modelo relacional y el álgebra relacional" de este curso.

El conjunto de componentes o herramientas conceptuales que un SGBD proporciona para modelar recibe el nombre de **modelo de BD**. Los cuatro modelos de BD más utilizados en los SI son el **modelo relacional**, el **modelo jerárquico**, el **modelo en red** y el **modelo relacional con objetos**.

¡Cuidado con las confusiones!

Popularmente, en especial en el campo de la informática personal, se denomina *BD* a lo que aquí denominamos *SGBD*. Tampoco se debe confundir la BD considerada como modelo de la realidad con lo que aquí denominamos *modelo de BD*. El modelo de BD es el conjunto de herramientas conceptuales (piezas) que se utilizan para construir el modelo de la realidad.

Todo modelo de BD nos proporciona tres tipos de herramientas:

a) **Estructuras de datos** con las que se puede construir la BD: tablas, árboles, etc.

b) Diferentes tipos de **restricciones (o reglas) de integridad** que el SGBD tendrá que hacer cumplir a los datos: dominios, claves, etc.

c) Una serie de **operaciones** para trabajar con los datos. Un ejemplo de ello, en el modelo relacional, es la operación SELECT, que sirve para seleccionar (o leer) las filas que cumplen alguna condición. Un ejemplo de operación típica del modelo jerárquico y del modelo en red podría ser la que nos dice si un determinado registro tiene "hijos" o no.

Evolución de los modelos de BD

De los cuatro modelos de BD que hemos citado, el que apareció primero, a principios de los años sesenta, fue el **modelo jerárquico**. Sus estructuras son registros interrelacionados en forma de árboles. El SGBD clásico de este modelo es el IMS/DL1 de IBM.

A principios de los setenta surgieron SGBD basados en un **modelo en red**. Como en el modelo jerárquico, hay registros e interrelaciones, pero un registro ya no está limitado a ser "hijo" de un solo registro tipo. El comité CODASYL-DBTG propuso un estándar basado en este modelo, que fue adoptado por muchos constructores de SGBD*. Sin embargo, encontró la oposición de IBM, la empresa entonces dominante. La propuesta de CODASYL-DBTG ya definía tres niveles de esquemas.

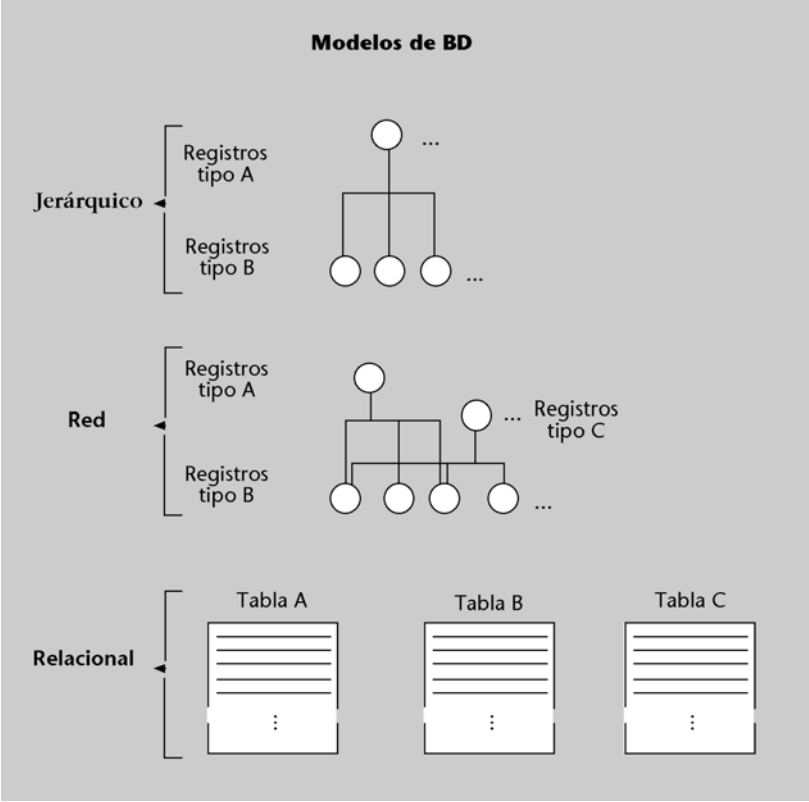
*** Por ejemplo, IDS de Bull, DMS de Univac y DBMS de Digital.**

Durante los años ochenta apareció una gran cantidad de SGBD basados en el **modelo relacional** propuesto en 1969 por E.F. Codd, de IBM, y prácticamente todos utilizaban como lenguaje nativo el SQL**. El modelo relacional se basa en el concepto matemático de *relación*, que aquí podemos considerar de momento equivalente al término *tabla* (formada por filas y columnas). La mayor parte de los SI que actualmente están en funcionamiento utilizan SGBD relacionales, pero algunos siguen utilizando los jerárquicos o en red (especialmente en SI antiguos muy grandes).

** Por ejemplo, Oracle, DB2 de IBM, Informix, Ingres, Allbase de HP y SQL-Server de Sybase.

El modelo relacional se estudia con detalle en la unidad didáctica "El modelo relacional y el álgebra relacional" de este curso.

Figura 8



Así como en los modelos prerrelacionales (jerárquico y en red), las estructuras de datos constan de dos elementos básicos (los registros y las interrelaciones), en el modelo relacional constan de un solo elemento: la tabla, formada por filas y columnas. Las interrelaciones se deben modelizar utilizando las tablas.

Otra diferencia importante entre los modelos prerrelacionales y el modelo relacional es que el modelo relacional se limita al nivel lógico (no hace absolutamente ninguna consideración sobre las representaciones físicas). Es decir, nos da una independencia física de datos total. Esto es así si hablamos del modelo teórico, pero los SGBD del mercado nos proporcionan una independencia limitada.

Estos últimos años se está extendiendo el **modelo de BD relacional con objetos**. Se trata de ampliar el modelo relacional, añadiéndole la posibilidad de que los tipos de datos sean tipos abstractos de datos, TAD. Esto acerca los sistemas relacionales al paradigma de la OO. Los primeros SGBD relacionales

que dieron esta posibilidad fueron Oracle (versión 8), Informix (versión 9) e IBM/DB2/UDB (versión 5).

Hablamos de modelos de BD, pero de hecho se acostumbran a denominar *modelos de datos*, ya que permiten modelarlos. Sin embargo, hay modelos de datos que no son utilizados por los SGBD del mercado: sólo se usan durante el proceso de análisis y diseño, pero no en las realizaciones.

Los más conocidos de estos tipos de modelos son los **modelos semánticos** y los **funcionales**. Éstos nos proporcionan herramientas muy potentes para describir las estructuras de la información del mundo real, la semántica y las interrelaciones, pero normalmente no disponen de operaciones para tratarlas. Se limitan a ser herramientas de descripción lógica. Son muy utilizados en la etapa del diseño de BD y en herramientas CASE. El más extendido de estos modelos es el conocido como **modelo ER** (*entity-relationship*), que estudiaremos más adelante.

La evolución de los modelos...

... a lo largo de los años los ha ido alejando del mundo físico y los ha acercado al mundo lógico; es decir, se han alejado de las máquinas y se han acercado a las personas.

Actualmente, la práctica más extendida en el mundo profesional de los desarrolladores de SI es la utilización del modelo ER durante el análisis y las primeras etapas del diseño de los datos, y la utilización del modelo relacional para acabar el diseño y construir la BD con un SGBD.

En esta asignatura hablamos sólo de BD con modelos de datos estructurados, que son los que normalmente se utilizan en los SI empresariales. Sin embargo, hay SGBD especializados en tipos de aplicaciones concretas que no siguen ninguno de estos modelos. Por ejemplo, los SGBD documentales o los de BD geográficas.

6. Lenguajes y usuarios

Para comunicarse con el SGBD, el usuario, ya sea un programa de aplicación o un usuario directo, se vale de un lenguaje. Hay muchos lenguajes diferentes, según el tipo de usuarios para los que están pensados y el tipo de cosas que los usuarios deben poder expresar con ellos:

a) Habrá usuarios informáticos muy expertos que querrán escribir procesos complejos y que necesitarán lenguajes complejos.

b) Sin embargo, habrá usuarios finales no informáticos, ocasionales (esporádicos), que sólo harán consultas. Estos usuarios necesitarán un lenguaje muy sencillo, aunque dé un rendimiento bajo en tiempo de respuesta.

c) También podrá haber usuarios finales no informáticos, dedicados o especializados. Son usuarios cotidianos o, incluso, dedicados exclusivamente a trabajar con la BD*. Estos usuarios necesitarán lenguajes muy eficientes y compactos, aunque no sea fácil aprenderlos. Tal vez serán lenguajes especializados en tipos concretos de tareas.

¿Qué debería poder decir el usuario al SGBD?

Por un lado, la persona que hace el diseño debe tener la posibilidad de describir al SGBD la BD que ha diseñado. Por otro lado, debe ser posible pedirle al sistema que rellene y actualice la base de datos con los datos que se le den. Además, y obviamente, el usuario debe disponer de medios para hacerle consultas.

*** Por ejemplo, personas dedicadas a introducir datos masivamente.**

Hay lenguajes especializados en la escritura de esquemas; es decir, en la descripción de la BD. Se conocen genéricamente como **DDL** o *data definition language*. Incluso hay lenguajes específicos para esquemas internos, lenguajes para esquemas conceptuales y lenguajes para esquemas externos.

Otros lenguajes están especializados en la utilización de la BD (consultas y mantenimiento). Se conocen como **DML** o *data management language*. Sin embargo, lo más frecuente es que el mismo lenguaje disponga de construcciones para las dos funciones, DDL y DML.

El **lenguaje SQL**, que es el más utilizado en las BD relacionales, tiene verbos –instrucciones– de tres tipos diferentes:

El lenguaje SQL se explicará en la unidad didáctica "El lenguaje SQL" de este curso.

1) Verbos del tipo DML; por ejemplo, `SELECT` para hacer consultas, e `INSERT`, `UPDATE` y `DELETE` para hacer el mantenimiento de los datos.

2) Verbos del tipo DDL; por ejemplo, `CREATE TABLE` para definir las tablas, sus columnas y las restricciones.

3) Además, SQL tiene **verbos de control del entorno**, como por ejemplo `COMMIT` y `ROLLBACK` para delimitar transacciones.

En cuanto a los **aspectos DML**, podemos diferenciar dos tipos de lenguajes:

Lenguajes declarativos y procedimentales

El aprendizaje y la utilización de los lenguajes procedimentales acostumbran a ser más difíciles que los declarativos, y por ello sólo los utilizan usuarios informáticos. Con los procedimentales se pueden escribir procesos más eficientes que con los declarativos.

a) Lenguajes muy **declarativos** (o implícitos), con los que se especifica qué se quiere hacer sin explicar cómo se debe hacer.

b) Lenguajes más explícitos o **procedimentales**, que nos exigen conocer más cuestiones del funcionamiento del SGBD para detallar paso a paso cómo se deben realizar las operaciones (lo que se denomina *navegar* por la BD).

Como es obvio, los **aspectos DDL** (las descripciones de los datos) son siempre declarativos por su propia naturaleza.

Los lenguajes utilizados en los SGBD prerrelacionales eran procedimentales. SQL es básicamente declarativo, pero tiene posibilidades procedimentales.

Aunque casi todos los SGBD del mercado tienen SQL como lenguaje nativo, ofrecen otras posibilidades, como por ejemplo 4GL y herramientas visuales:

1) Lenguajes 4GL (*4th Generation Languages*)***** de muy alto nivel, que suelen combinar elementos procedimentales con elementos declarativos. Pretenden facilitar no sólo el tratamiento de la BD, sino también la definición de menús, pantallas y diálogos.

*** Empezaron a aparecer al final de los años ochenta.**

2) Herramientas o interfaces visuales** muy fáciles de utilizar, que permiten usar las BD siguiendo el estilo de diálogos con ventanas, iconos y ratón, puesto de moda por las aplicaciones Windows. No sólo son útiles a los usuarios no informáticos, sino que facilitan mucho el trabajo a los usuarios informáticos: permiten consultar y actualizar la BD, así como definirla y actualizar su definición con mucha facilidad y claridad.

**** Han proliferado en los años noventa.**

Tanto los 4GL como las herramientas visuales (con frecuencia unidas en una sola herramienta) traducen lo que hace el usuario a instrucciones SQL por distintas vías:

- En el caso de los 4GL, la traducción se suele hacer mediante la compilación.
- En el caso de las herramientas visuales, se efectúa por medio del intérprete de SQL integrado en el SGBD.

Si queremos escribir un programa de aplicación que trabaje con BD, seguramente querremos utilizar nuestro lenguaje habitual de programación*****. Sin embargo, generalmente estos lenguajes no tienen instrucciones para realizar el acceso a las BD. Entonces tenemos las dos opciones siguientes:

*** Pascal, C, Cobol, PL/I, Basic, MUMPS, Fortran, Java, etc.**

1) Las **llamadas a funciones**: en el mercado hay librerías de funciones especializadas en BD (por ejemplo, las librerías ODBC). Sólo es preciso incluir llamadas a las funciones deseadas dentro del programa escrito con el lenguaje habitual. Las funciones serán las que se encargarán de enviar las instrucciones (generalmente en SQL) en tiempo de ejecución al SGBD.

2) El **lenguaje hospedado**: otra posibilidad consiste en incluir directamente las instrucciones del lenguaje de BD en nuestro programa. Sin embargo, esto exige utilizar un precompilador especializado que acepte en nuestro lenguaje de programación habitual las instrucciones del lenguaje de BD. Entonces se dice que este lenguaje (casi siempre SQL) es el lenguaje hospedado o incorporado (*embedded*), y nuestro lenguaje de programación (Pascal, C, Cobol, etc.) es el lenguaje anfitrión (*host*).

7. Administración de BD

Hay un tipo de usuario especial: el que realiza tareas de administración y control de la BD. Una empresa o institución que tenga SI construidos en torno a BD necesita que alguien lleve a cabo una serie de funciones centralizadas de gestión y administración, para asegurar que la explotación de la BD es la correcta. Este conjunto de funciones se conoce con el nombre de **administración de BD (ABD)**, y los usuarios que hacen este tipo especial de trabajo se denominan *administradores de BD*.

Los **administradores de BD** son los responsables del correcto funcionamiento de la BD y velan para que siempre se mantenga útil. Intervienen en situaciones problemáticas o de emergencia, pero su responsabilidad fundamental es velar para que no se produzcan incidentes.

A continuación damos una lista de tareas típicas del ABD: 

1) Mantenimiento, administración y control de los esquemas. Comunicación de los cambios a los usuarios.

2) Asegurar la máxima disponibilidad de los datos; por ejemplo, haciendo copias (*back-ups*), administrando diarios (*journals* o *logs*), reconstruyendo la BD, etc.

3) Resolución de emergencias.

4) Vigilancia de la integridad y de la calidad de los datos.

5) Diseño físico, estrategia de caminos de acceso y reestructuraciones.

6) Control del rendimiento y decisiones relativas a las modificaciones en los esquemas y/o en los parámetros del SGBD y del SO, para mejorarlo.

7) Normativa y asesoramiento a los programadores y a los usuarios finales sobre la utilización de la BD.

8) Control y administración de la seguridad: autorizaciones, restricciones, etc.

La tarea del ABD no es sencilla.

Los SGBD del mercado procuran reducir al mínimo el volumen de estas tareas, pero en sistemas muy grandes y críticos se llega a tener grupos de ABD de más de diez personas. Buena parte del *software* que acompaña el SGBD está orientado a facilitar la gran diversidad de tareas controladas por el ABD: monitores del rendimiento, monitores de la seguridad, verificadores de la consistencia entre índices y datos, reorganizadores, gestores de las copias de seguridad, etc. La mayoría de estas herramientas tienen interfaces visuales para facilitar la tarea del ABD.

Resumen

En esta unidad hemos hecho una introducción a los conceptos fundamentales del mundo de las BD y de los SGBD. Hemos explicado la evolución de los SGBD, que ha conducido de una estructura centralizada y poco flexible a una distribuida y flexible, y de una utilización procedimental que requería muchos conocimientos a un uso declarativo y sencillo.

Hemos revisado los **objetivos de los SGBD actuales** y algunos de los **servicios** que nos dan para conseguirlos. Es especialmente importante el concepto de **transacción** y la forma en que se utiliza para velar por la integridad de los datos.

La **arquitectura de tres niveles** aporta una gran flexibilidad a los cambios, tanto a los físicos como a los lógicos. Hemos visto cómo un SGBD puede funcionar utilizando los tres esquemas propios de esta arquitectura.

Hemos explicado que los **componentes de un modelo de BD** son las estructuras, las restricciones y las operaciones. Los diferentes modelos de BD se diferencian básicamente por sus estructuras. Hemos hablado de los modelos más conocidos, especialmente del modelo relacional, que está basado en tablas y que estudiaremos más adelante.

El modelo relacional se estudiará en la unidad"El modelo relacional y el álgebra relacional" de este curso.

Cada tipo de usuario del SGBD puede utilizar un lenguaje apropiado para su trabajo. Unos usuarios con una tarea importante y difícil son los **administradores de las BD**.

Actividades

1. Comparad la lista de los objetivos de los SGBD que hemos dado aquí con la que se da en otros libros, y haced otra lista con las diferencias que encontréis.

2. Buscad información técnica y comercial de los fabricantes de SGBD sobre sus productos y a partir de aquí intentad reconocer los conceptos que hemos introducido en esta unidad.

3. Leed algún informe "Estado del arte" sobre SGBD de los que se publican (normalmente cada año) en la revista *Byte* y en *Datamation*.

Lectura recomendada

Consultad las obras de la bibliografía que encontraréis al final de esta unidad.

Ejercicios de autoevaluación

1. ¿Qué ventajas aportaron los SGBD relacionales con respecto a los prerrelacionales?

2. Para mejorar la disponibilidad y el coste, hemos decidido que una cierta parte de una BD que está situada en el ordenador central de la empresa estará duplicada (replicada) en un ordenador situado en una oficina alejada (conectado permanentemente por vía telefónica). Los programas que actualizan la BD, ¿tendrían que preocuparse de actualizar también la réplica? ¿Por qué?

3. Hemos programado una transacción para consultar cuántos alumnos cursan una asignatura. Si este número es inferior a quince, se nos informará de cuántos hay y en una lista, en una hoja de papel o en la pantalla nos aparecerán todos ellos. Sin embargo, si es superior o igual a quince, simplemente dirá cuántos hay. Supongamos que de forma concurrente con esta transacción se podrán estar ejecutando otras que inserten nuevos alumnos o que los supriman. ¿Qué problema se podrá producir si el SGBD no aísla totalmente las transacciones?

4. De las siguientes afirmaciones, decid cuáles son ciertas y cuáles son falsas:
a) El modelo ER es más conocido como *modelo relacional*.
b) Los SGBD no permiten la redundancia.
c) El DML es un lenguaje declarativo.
d) El DDL es un lenguaje pensado para escribir programas de consulta y actualización de BD.
e) En un ordenador que actúa como servidor de BD, con dos RAID y tres discos duros y con un SGBD actual, no es necesario que los encargados de realizar los programas para consultar esta BD sepan en qué discos está.
f) Cuando un programa quiere acceder a unos datos mediante un índice, lo debe decir al SGBD.

Solucionario

Ejercicios de autoevaluación

1. Los SGBD relacionales aportaron una programación más sencilla: los lenguajes son más sencillos y no dependen tanto de las características físicas de la BD. Se da más flexibilidad a los cambios (más independencia física de los datos). El programador se debe preocupar mucho menos de las cuestiones de rendimiento, pues de ello ya se ocupa el SGBD. Incluyen lenguajes declarativos de consulta para usuarios no informáticos.

2. Si la actualización no se hace en los dos lugares, la redundancia nos puede comportar problemas de consistencia de los datos. El administrador de la BD debería poder describir qué quiere que esté replicado y cómo quiere que se haga el mantenimiento de la réplica. El SGBD debería encargarse de mantener la réplica actualizada correctamente. Si la actualización de la réplica la tuviesen que hacer los programas de aplicación, podría suceder que alguno de ellos no lo hiciese, o incluso que la actualización la hiciese (mal) un usuario directo, sin escribir un programa.
Y todavía más, los programas de aplicación y los usuarios directos deberían ser totalmente ajenos a estos temas físicos (rendimiento, disponibilidad, etc.), ya que de este modo se podrían cambiar las decisiones, como por ejemplo variar la política de réplicas, sin que se tuviesen que modificar los programas ni avisar a nadie. Simplemente debe intervenir el ABD mediante el lenguaje de descripción del nivel físico. Actualmente, los SGBD del mercado ya dan este nivel de independencia.

3. Supongamos que la transacción consulta al SGBD cuántos alumnos hay, y el SGBD cuenta trece. El programa preparará una línea de cabecera de lista que indique que hay trece alumnos, y a continuación los mostrará. Sin embargo, entre el momento en que fabrica esta cabecera y el momento de empezar a leer a los alumnos uno por uno para mostrarlos, otras transacciones (que se ejecutan concurrentemente con ésta) eliminan a dos alumnos. Entonces, el programa nos mostrará sólo a once, a pesar de que había anunciado trece.
Y si las transacciones concurrentes que hacen actualizaciones hubiesen insertado tres alumnos, la cabecera diría que hay trece, y mostraría en realidad dieciséis (¡y no tendría que mostrar nunca más de quince!).
Estos problemas de concurrencia pueden surgir si el SGBD no lleva un control que evite las interferencias.

4. a) Falsa, **b)** Falsa, **c)** Falsa, **d)** Falsa, **e)** Cierta, **f)** Falsa.

Glosario

administrador de BD
Tipo de usuario especial que realiza funciones de administración y control de la BD, que aseguran que la explotación de la BD es correcta.

base de datos
Conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir diversas utilizaciones.
sigla: *BD*

BD
Ved: *base de datos*.

cliente/servidor
Tecnología habitual para distribuir datos. La idea es que dos procesos diferentes, que se pueden ejecutar en un mismo sistema o en sistemas separados, actúan de modo que uno actúa de cliente o peticionario de un servicio y el otro, como servidor. Un proceso cliente puede pedir servicios a distintos servidores. Un servidor puede recibir peticiones de muchos clientes. En general, un proceso *A* que actúa como cliente pidiendo un servicio a otro proceso *B* puede hacer también de servidor de un servicio que le pida otro proceso *C*.
sigla: *C/S*

C/S
Ved: *cliente/servidor*.

data definition language
Lenguaje especializado en la escritura de esquemas; es decir, en la descripción de BD.
Sigla: *DDL*

data manipulation language
Lenguaje especializado en la utilización de BD (consultas y mantenimiento).
Sigla: *DML*

DDL
Ved: *data definition language*.

DML
Ved: *data manipulation language*.

esquema
Descripción o definición de la BD. Esta descripción está separada de los programas y es utilizada por el SGBD para saber cómo es la BD con la que debe trabajar. La arquitectura ANSI/SPARC recomienda tres niveles de esquemas: el externo (visión de los usuarios), el conceptual (visión global) y el físico (descripción de características físicas).

SGBD
Ved: *sistema de gestión de BD*.

sistema de gestión de BD
Software que gestiona y controla BD. Sus principales funciones son facilitar la utilización de la BD a muchos usuarios simultáneos y de tipos diferentes, independizar al usuario del mundo físico y mantener la integridad de los datos.
sigla: SGBD

SQL
Ved: *structured query language*.

structured query language
Lenguaje especializado en la descripción (DDL) y la utilización (DML) de BD relacionales. Creado por IBM al final de los años setenta y estandarizado por ANSI-ISO en 1985 (el último estándar de SQL es de 1999). En la actualidad lo utilizan prácticamente todos los SGBD del mercado.
sigla: SQL

transacción
Conjunto de operaciones (de BD) que queremos que se ejecuten como un todo (todas o ninguna) y de forma aislada (sin interferencias) de otros conjuntos de operaciones que se ejecuten concurrentemente.

Bibliografía

Bibliografía básica

Date, C.J. (2001). *Introducción a los sistemas de bases de* datos (7ª ed.). Prentice Hall.

Silberschatz, A; Korth, H.F; Sudarshan, S. (1998). *Fundamentos de bases de datos* (3.ª ed.). Madrid: McGraw-Hill.
La introducción de este libro es similar a la que hemos hecho aquí.

Bibliografía complementaria

Para actualizar vuestra visión global de los SGBD del mercado, podéis consultar en Internet las páginas de los fabricantes de SGBD; por ejemplo:

- *Informix*: http://www.informix.com
- *Oracle*: http://www.oracle.com
- *IBM*: http://www.ibm.com
- *Computer Associates*: http://www.ca.com
- *Microsoft*: http://www.microsoft.com
- *NCR*: http://www.ncr.com