

### 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 ir las 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. Antigüamente, 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.

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.

#### 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.

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

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

#### 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.

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 abonarlo 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

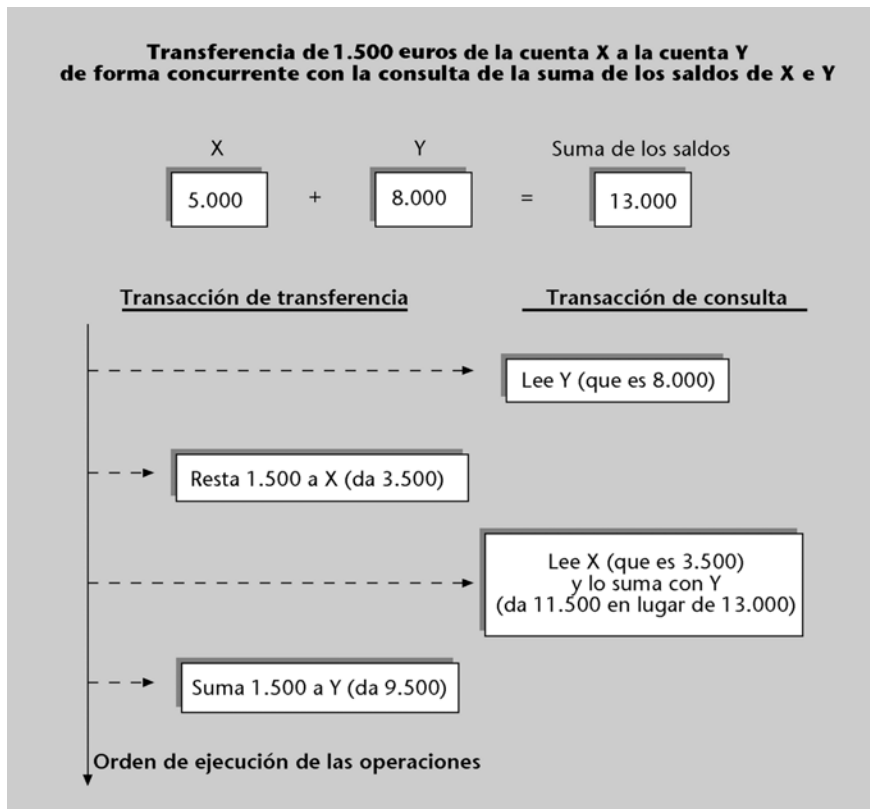
1) Imaginemos que una transacción que transfiere dinero de X a Y se ejecuta concurrentemente con una transacción que observa el saldo de las cuentas Y y X, en este orden, y nos muestra su suma. Si la ejecución de forma concurrente de las dos transacciones casualmente

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.

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.

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.

\* 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).

#### Derechos de acceso

Por ejemplo, el usuario SEC3 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.

### 3.7. Otros objetivos

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.

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