Funciones de los Sistemas Gestores de Bases de Datos*

Miguel-Angel Sicilia

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 2.0†

Abstract

Se introduce el concepto de Sistema Gestor de Bases de Datos (SGBD) como una pieza independiente de software que proporciona servicios de gestión de datos a las aplicaciones. Se describen las principales funciones de los SGBD.

1 Los Sistemas de Gestión de Bases de Datos son software especializado

El concepto de Sistema Gestor de Bases de Datos (SGBD) puede definirse de la siguiente forma:

Sistema Gestor de Bases de Datos (SGBD). Un software que proporciona servicos para la creación, el almacenamiento, el procesamiento y la consulta de la información almacenada en base de datos de forma segura y eficiente. Un SGBD actúa como un intermediario entre las aplicaciones y los datos.

Es decir, vamos a considerar aquí un SGDB como un software que proporciona funcionalidad añadida al sistema de ficheros para facilitar la gestión de datos.

Los SGBD pueden residir (y normalmente así lo hacen para mejorar el rendimiento) en una máquina diferente a la que ejecuta las aplicaciones. De hecho, las aplicaciones modernas se programan de forma que se puede utilizar esta característica de distribución física, aunque a la hora de instalar la aplicación no se utilice y se ubique todo el software en la misma máquina. Esto ha dado lugar a diferentes configuraciones de la arquitectura de las aplicaciones, todas ellas conocidas como arquitecturas **multi-capa**. La Figura 1 muestra arquitecturas típicas de dos, tres y cuatro capas, con ejemplos concretos de protocolos y piezas software, suponiendo que se utiliza Java como plataforma de desarrollo.

*Version 1.1: Sep 23, 2008 9:52 am -0500

†http://creativecommons.org/licenses/by/2.0/

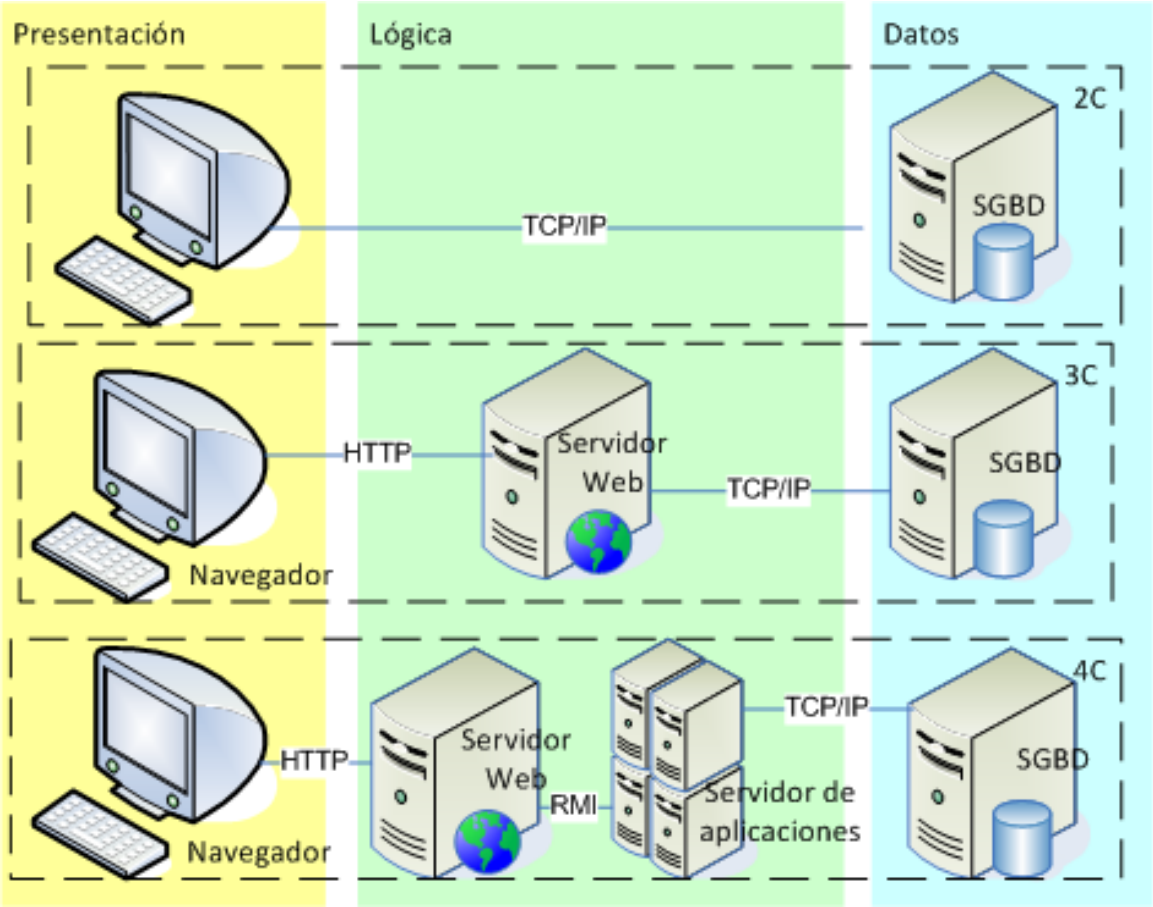

Figure 1

La división en capas generalmente se estrucuta en tes partes: presentación, lógica y datos. En las arquitecturas de dos capas (2C), la capa de presentación tiene la interfaz de usuario y todo el código para el procesamiento de los datos. La parte que se separa es la gestión de los datos que proporciona un SGBD. La interfaz entre las dos capas suele hacerse a través de una interfaz de programación de aplicaciones, por ejemplo, JDBC[1] en el caso de Java. Para ello, en el lado de la aplicación se cuenta con un manejador (*driver*) que se encarga de gestionar las comunicaciones. Estas normalmente se hacen vía TCP/IP, de acuerdo al protocolo propio de cada SGBD. En las aplicaciones de tres capas (3C) aparece una capa intermedia que suele denominarse de "lógica" o "lógica del negocio". La idea es que en esa capa estén las aplicaciones, y éstas se puedan acceder mediante algún cliente ligero. El ejemplo típico de uno de estos clientes son los navegadores Web. Así, el navegador es la interfaz (la presentación), y en el servidor Web está el código de acceso a la base de datos (lógicamente, entre el cliente y el servidor se utiliza HTTP como protocolo). De este modo se evita que en la máquina del usuario sea necesario instalar nada más que el navegador. En Java, estas aplicaciones 2C utilizan tecnología de servidor Web como JSP. Un elemento adicional de separación puede darse en la capa intermedia, en el caso de que se utilice un servidor de aplicaciones.

[1] http://java.sun.com/javase/technologies/database/ (<http://java.sun.com/javase/technologies/database/>)

2 Los SGBD nos abstraen de la representación de los datos

Los SGBD pueden considerarse como intermediarios entre las aplicaciones y la representación de los datos. Así, los desarrolladores ven los datos desde una perspectiva de más alto nivel. Esa perspectiva es la del **modelo de bases de datos** utilizado. De esta forma, el desarrollo de aplicaciones separa dos aspectos:

- El diseño de la(s) bases de datos, que finaliza con una base de datos creada en un SGBD terminado y diseñada de acuerdo a un modelo.
- El diseño y desarrollo de las aplicaciones, que se hace utilizando el diseño de la base de datos.

Ambos tipos de actividades no se hacen en secuencia, sino de forma simultánea, pero en la mayoría de las aplicaciones actuales se utiliza un SGBD de un tipo u otro. Así, el desarrollo de las aplicaciones en general requiere menos esfuerzo, ya que los desarrolladores no tienen que preocuparse de diseñar una gestión de datos eficiente, dado que el SGBD se supone que se encarga de eso. Los desarrolladores solo tratan con el SGBD mediante lenguajes normalizados (y a veces estandarizados) de alto nivel que ocultan muchos detalles de almacenamiento y permiten al programador centrarse en definir qué datos quiere obtener o guardar, y no en los detalles de cómo estos datos están almacenados y cómo se puede acceder de forma eficiente a ellos.

En otras palabras, es una función esencial de los SGBD en proporcionar **independencia de los datos y las aplicaciones**. Así, se puede diseñar la base de datos incluso antes que las aplicaciones, y ese diseño permitirá que se desarrollen múltiples aplicaciones posteriormente.

3 Podemos ver las bases de datos en tres niveles

Los SGBD ocultan la representación interna de los datos ofreciendo un conjunto de funciones de más alto nivel. No obstante, en ocasiones es necesario "descender" un poco en los detalles de almacenamiento. Por ejemplo, esto a veces hace falta cuando se necesita ajustar el rendimiento de las consultas al máximo. En otros casos, una base de datos puede guadar información de cientos de entidades. Esto sucede por ejemplo en algunos sistemas de gestión empresarial. Pero muchas aplicaciones no requiren manipular o consultar todas las tablas, sino que a veces sólo necesitan algunos datos de unas pocas entidades.

Los diferentes niveles de abstracción a menudo se denominan **niveles**:

- El **nivel físico** permite ver (algunos) detalles de la representación de la información en ficheros. Este nivel sólo es de interés para los desarrolladores que se encargan de afinar el rendimiento de las consultas y actualizaciones. Requiere conocimientos técnicos muy específicos del SGBD concreto que se esté utilizando, y no están normalizados ni estandarizados.
- El **nivel lógico** es el que se utiliza en el desarrollo (a excepción de las tareas que acabamos de mencionar), y considera la base de datos como un conjunto de informaciones y sus relaciones. Lo que se maneja es el qué información se almancena y no cómo está almacenada. Este nivel lógico suele ajustarse a un determinado modelo de bases de datos, y en ocasiones estos modelos tienen un cierto nivel de estandarización. Por ejemplo, los SGBD que implementan el modelo relacional son a nivel lógico prácticamente iguales, y cuentan con lenguajes de consulta como SQL que han sido estandarizados.
- El **nivel de vistas** permite que se oculte a nivel lógico una parte de la base de datos, y que sólo veamos algunas informaciones determinadas. Una vista es un subconjunto de la información de una base de datos que se ha definido para abstraer una parte concreta de una base de datos. Las vistas se utilizan para simplificar su uso por parte de desarrolladores y usuarios.

Es importante notar que una vista de una base de datos para sus usuarios es como una base de datos (más pequeña) a nivel lógico.

En una base de datos, las definiciones de la información se denominan **esquemas** (es decir, el esquema son las definiciones, no los datos en sí). El más importante de los esquemas es el esquema lógico, que realmente define la información a almacenar. El esquema físico define cómo se almacena esa información. También se habla a veces del esquema de una vista, haciendo referencia a la definición de la parte de la definición de la base de datos a la que la vista da acceso. Las diferentes vistas se definen mediante un conjunto de lenguajes.

3.1 Los lenguajes de definición de datos

El esquema de una base de datos se define mediante un lenguaje de definición de datos (LDD). El lenguaje SQL incluye un LDD para bases de datos diseñadas de acuerdo al modelo relacional. Por ejemplo, la siguiente definición SQL crea una tabla en un SGBD relacional.

```
CREATE TABLE clientes (
id CHAR(10) NOT NULL,
nombre VARCHAR(20),
fechanac DATE,
telefono CHAR(15),
PRIMARY KEY(id)
);
```

En el ejemplo se crea una tabla relacional `clientes`, y se definen algunos de sus atributos. El LDD define ciertos tipos de datos como `CHAR`, `VARCHAR` o `DATE`. Además, incluye Cuando se ejecuta la sentencia anterior en un SGBD se crea una tabla vacía con la estuctura especificada, en la base de datos que indiquemos. Además de la definición de los datos, tenemos algunos ejemplos de restricciones. El atributo id no puede tener un valor vacío (`NOT NULL`) y además ese mismo atributo identifica a cada una de las filas de la tabla, es decir, es su clave primaria (`PRIMARY KEY`).

Veamos ahora la misma sentencia pero ampliada con dos cláusulas adicionales.

```
CREATE TABLE clientes (
id CHAR(10) NOT NULL,
...,
INDEX(id)
) ENGINE = InnoDB;
```

Las dos cláusulas adicionales corresponden a aspectos del modelo físico. La primera indica que se debe crear un estructura de índice (`INDEX`) sobre el atributo `id`. Un índice es una estructura de datos especializada en hacer más rápido el acceso en las consultas por el atributo sobre el que se construye el índice.

La cláusla `ENGINE` especifica qué forma de almacenamiento se debe utilizar para la tabla que se crea. `InnoDB` es una forma de almacenamiento específica del SGBD MySQL. El siguiente es un extracto de la documentación técnica de MySQL 5.0 que describe InnoDB:

[..] el motor de almacenamiento InnoDB mantiene su propio pool de buffers como caché para los datos y los índices en memoria principal. InnoDB almacena sus tablas e índices en un espacio de tablas (tablespace), que puede consistir en varios ficheros (o en varias particiones del disco sin formatear). Esto es diferente de, por ejemplo, las tablas MyISAM en las que cada tabla se almacena en ficheros diferentes. Las tablas InnoDB pueden ser de cualquier tamaño incluso en sistemas operativos en los que el tamaño de fichero está limitado a 2GB.

Como se puede apreciar, estamos tratando con aspectos físicos del almacenamiento, que no tienen que ver con la definición de los datos en sí. Los índices de tablas `InnoDB`, por ejemplo, por defecto utilizan una estructura `BTREE`, es decir, de árbol balanceado (balanced tree). Esto ocupa espacio en disco ya que es una estructura adicional, pero hace más eficiente las consultas. Por eso, también en la documentación técnica se recomienda que los índices se hagan sobre campos de longitud reducida si es posible. En conclusión, las sentencias de definición mezclan aspectos de la vista lógica y la vista física. Lo normal es que los aspectos físicos se añadan a posteriori, después de haber creado el diseño lógico sin prestarles atención.

3.2 Los lenguajes de manipulación de datos

Un segundo tipo son los conocidos como lenguajes de manipulación de datos (LMD). Por "manipulación" se entiende incluir nuevos datos, actualizar los existentes, borrar algunos de ellos y, por supuesto, consultar datos. Estos lenguajes afectan a los datos, y no a los esquemas, que fueron definidos previamente con algún tipo de LDD.

Siguiendo el ejemplo anterior, la siguiente sería una sentencia de inserción de una fila en la tabla `clientes`.

```
INSERT INTO clientes (id, nombre)
```

```
VALUES ('500234L', 'John Doe');
```

La anterior sentencia inserta una fila con el identificador y nombre especificados. Si intentamos volver a ejecutarla, el SGBD no nos lo permitirá, ya que estaríamos creando otra fila con el mismo valor la misma `PRIMARY KEY`. Esta es la forma en que las restricciones nos permiten evitar situaciones de incoherencia en los datos. Sin embargo, ya que no indicamos ninguna restricción, por ejemplo, en la fecha de nacimiento, la nueva fila tendrá un valor nulo (NULL) como fecha de nacimiento (lo cual no representa ningún problema, dado que es bastante común que algunos clientes vanidosos no quieran revelarnos su edad). Pero es más, con la definición que hicimos es perfectamente válido hacer lo siguiente:

```
INSERT INTO clientes (id)
VALUES ('234567M');
```

Con lo cual hemos insertado un cliente del que sólo sabemos el identificador (dato que difícilmente nos será de utilidad por sí solo). En otras palabras, las restricciones son realmente importantes ya que evitan muchas situaciones no deseables en cuanto a la utilidad o coherencia de los datos.

Como ejemplo de la consulta de datos, consideremos la siguiente sentencia SQL.

```
SELECT id, nombre FROM clientes WHERE nombre LIKE "John%"
```

Esta sentencia recuperará el cliente que se insertó con la operación anterior. Los leguajes de consulta nos proporcionan un arsenal de posibles combinaciones sintácticas para obtener la parte de los datos que nuestra aplicación necesita.

4 Los SGBD permiten implementar restricciones

Como ya hemos visto, los SGBD nos permiten implementar diferentes tipos de **restricciones**. En los ejemplos hemos visto restricciones de unicidad (no puede haber dos filas con el mismo valor en la clave primaria), o de valores nulos (algunos campos pueden tener valores nulos y otros no). También implicitamente el limitar el tamaño de un campo (como en `CHAR(10)`) es una restricción que afecta a los tipos de datos. Hay otros tipos de restricciones, pero lo importante es entender que un buen diseño de bases de datos pasa por una consideración detallada y acertada de las restricciones de la definición de los datos.

5 Los SGBD protegen los datos ante actualizaciones fallidas

Los ordenadores no son máquinas perfectas, y en ocasiones ocurren desastres. Sea por desatención o por los duendes, en ocasiones un ordenador deja de funcionar temporalmente (por ejemplo, cuando falta la corriente eléctrica).

¿Qué es lo que ocurre si hay una de esas caídas en la máquina que contiene el SGBD? Si tenemos suerte, el SGBD no estaría haciendo ninguna operación de acceso a los datos en esos momentos, pero la fortuna es sólo para unos pocos. En la mayoría de los casos, el SGBD estaría haciendo algún tipo de actualización, de modo que parte de los dato estuvieran en memoria principal. Los SGBD modernos proporcionan mecanismos de **recuperación** que impiden que la información en la base de datos quede en un estado incoherente.

El resultado si no hubiese esos mecanismos de recuperación sería que los datos en la base de datos podrían quedar inconsistentes. Incluso la propia base de datos podría quedar dañada en su estructura.

Los mecanismos de recuperación funcionan considerando **transacciones**. Una transacción es un conjunto de operaciones que deben ejecutarse "a la vez". "A la vez" no quiere decir aquí exactamente en el mismo instante de tiempo, sino que o bien se ejecutan todas o bien ninguna. Por ejemplo, si tenemos una transacción que representa un intercambio en el horario y aulas de dos exámenes dentro de una base de datos con ese calendario, el cambio podría hacerse mediante dos sentecias SQL `REPLACE`: una para cambiar cada una de las filas en una supuesta tabla `examenes`. Pues bien, si el sistema cae cuando se ha ejecutado una de ellas y no la otra, el resultado sería que los dos exámenes coincidirían en aula y hora (un estado incoherente). Pero esto no ocurrirá si el programador tienen el buen juicio de colocar las dos sentencias `REPLACE` como operaciones de una misma transacción. En ese caso, si se realizó el primer `REPLACE` y no el segundo, el SGBD deshará (`rollback`) los cambios del primero al volver a estar en funcionamiento.

6 Los SGDB permiten el acceso y actualización concurrente

Además de las caídas a las que hemos mencionado, las bases de datos pueden quedar en estados incoherentes si varios programas intentan actualizar los datos a la vez. Los SGBD para estos casos también consideran las transacciones como secuencias de operaciones "todo o nada". Además, los SGBD proporcionan diferentes mecanismos de **control de concurrencia**.

Hay diferentes políticas de control de concurrencia, unas más liberales y otras más restrictivas, con sus ventajas e inconvenientes. Una política restrictiva típica es la de **bloquear** la parte de la base de datos que se está actualizando. Así, lo que se hace es **serializar** (es decir, hacer pasar "uno a uno") el acceso de los diferentes programas (de las diferentes transacciones) a esa determinada porción de los datos.

7 Los SGBD proporcionan mecanismos de control de acceso

Una base de datos es por tanto un almacén separado de las aplicaciones. Típicamente, una base de datos en una empresa será utilizada por diferentes Departamentos para diferentes usos. Por ello, es lógico que diferentes personas (**usuarios**) tengan diferentes posibilidades (derechos, **privilegios** o permisos) de uso y manipulación, no solo de los datos, sino del esquema que los define.

Típicamente, en un SGBD existe la figura del Administrador de Base de Datos (DBA, Database administrator) que tiene todos las posibilidades sobre las bases de datos albergadas en el SGBD, y que puede comenzar a crearlas. Así, el sistema de privilegios actuará por delegación, es decir, el DBA creará una base de datos y repartirá ciertos privilegios sobre ella a ciertor usuarios (o grupos de usuarios). A su vez, algunos de estos usuarios tendrán derecho a delegar a otros ciertos privilegios sobre ciertas partes de la base de datos, etc.

A todo este sistema de privilegios se denomina habitualmente control de acceso. Además del control de acceso, un SGBD proporciona un registro de todos los accesos a las bases de datos, permitiendo así auditar los cambios, es decir, inspecionar quién hizo qué en qué momento.

8 ... y los SGBD proporcionan otros servicios interesantes

Hay muchos otros servicios que habitualmente proporcionan los SGBD. Entre ellos podemos contar los servicios de copia de seguridad (**backup**), la compresión de datos o la felxibilidad en añadir nuevos tipos de datos.

9 Pero los SGBD no pueden hacer algunas cosas

A pesar de que los SGDB nos proporcionan muchas herramientas para una gestión eficaz y eficiente de los datos, hay aspectos fundamentales que no dependen del uso de un SGBD, sino de un **buen diseño de la base de datos**. Es decir, dependen del trabajo de los diseñadores. Por ejemplo, spongamos que tenemos el siguiente esquema de base de datos:

`PEDIDOS(id-cliente, nombre-cliente, id-producto, nombre-producto, cantidad-pedida, fecha, stock-producto)`

`PRODUCTOS(id, nombre, precio)`

`CLIENTES(id, nombre, ...)`

El diseño en este caso causará diferentes problemas. Entre ellos podemos mencionar:

- Existen una redundancia innecesaria. No es necesario colocar en la tabla de `PEDIDOS` los nombres del producto y del cliente (que ya aparecen en sus respectivas tablas). Esto da lugar a problemas de inconsistencia en la actualización. Por ejemplo, si cambiamos el nombre de un producto en `PRODUCTOS`, estaremos obligados a cambiarlo en todos los `PEDIDOS` (cosa que tiene un gran coste), o tendremos información inconsistente.
- El colocar el `precio` dentro de la tabla productos tiene un efecto de pérdida de información. Si cambiamos el precio allí, dado que no está en la tabla `PEDIDOS`, ya nunca más sabremos a qué precio

se vendió el producto en cada `PEDIDO`. La otra alternativa es que el precio nunca cambie, lo cual es bueno para la inflación pero poco realista.

- El colocar el `stock-producto` que quedaría tras el pedido en esa tabla mantiene la información necesaria, pero en el sitio incorrecto (lo que obligaría para saber o decrementar el stock a consultar el último pedido, lo cual será más ineficiente, dado que las tablas de pedidos suelen ser mucho más grandes que las de clientes), y generando otra vez mucha redundancia. El stock debería quedar en la tabla `PRODUCTOS`.

En resumen, el SGBD nos garantiza lo que incluyamos como restricciones, y se preocupa de la concurencia y las caídas, pero no es capaz de mitigar el daño de un mal diseño.