Unidad 1: Almacenamiento de la información

Índice de contenidos

1 Introducción.

¿Te has preguntado alguna vez dónde y de qué manera se almacenan y gestionan los datos que utilizamos diariamente? Si pensamos **en cualquier acción de nuestra vida cotidiana**, o si analizamos la mayoría de los ámbitos de actividad, **nos encontramos que la utilización de las bases de datos está ampliamente extendida**. **Éstas, y los datos contenidos en ellas, serán imprescindibles para llevar a cabo multitud de acciones**.

¿Crees que no es para tanto? Piensa en las siguientes situaciones:

- ✓ Cuando seleccionamos nuestro canal favorito en la televisión.
- ✓ Al utilizar la agenda del móvil para realizar una llamada telefónica.
- ✓ Cuando operamos en el cajero automático.
- ✓ Al solicitar un certificado en un organismo público.
- ✓ Cuando acudimos a la consulta del médico.
- ✓ Al inscribirnos en un curso, plataforma on-line, etc.
- ✓ Cuando reservamos unas entradas para un evento deportivo o espectáculo.
- ✓ Si consumimos ocio digital.
- ✓ Cuando consultamos cualquier información en *Internet* (bibliotecas, enciclopedias, museos, etc.).
- ✓ Al registrarte en una página de juegos on-line, redes sociales o foros.
- ✓ Incluso, si tienes coche, puede ser que éste incorpore alguna base de datos.

No es necesario continuar más para darse cuenta de que **casi todo lo que nos rodea, en alguna medida, está relacionado con los datos, su almacenamiento y su gestión**. **El gran volumen de datos que actualmente se manejan y sus innumerables posibilidades requieren de la existencia de técnicos perfectamente formados y capaces de trabajar con ellos**.

Este módulo profesional se centra en el **estudio de las *Bases de Datos* (*BD*) y su uso en el desarrollo de aplicaciones**.

2 Los ficheros de información.

2.1 ¿Qué es un fichero de datos?

El **sistema de archivos** es un sistema lógico de almacenamiento y recuperación para nombrar y colocar archivos.

También incluye el formato de la ruta especificada para encontrar el archivo (o fichero) a través de la estructura de carpetas (o directorios).

Tipos: *FAT*, *NTFS*, *ext4*, *APFS*, *ZFS*, …

En la década de los setenta, los procesos básicos que se llevaban a cabo en las empresas se centraban en cuestiones relacionadas con la contabilidad y la facturación. Las necesidades de almacenamiento y gestión de información podían satisfacerse utilizando un número relativamente reducido de archivos en papel agrupados y ordenados, los típicos ficheros clásicos.

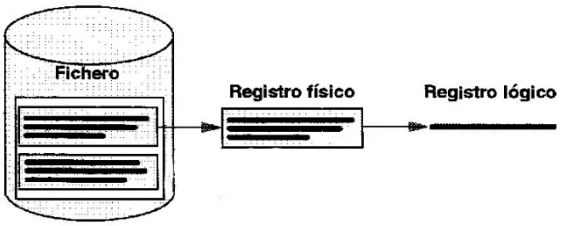
Al llevar a cabo una primera informatización, se pasó de tener los datos en formato papel a poder acceder a ellos de manera mucho más rápida a través del ordenador. En ese momento, **la informática adaptó sus herramientas para que los elementos que el usuario manejaba en el ordenador se parecieran a los que utilizaba manualmente**. Así en informática se sigue hablando de ficheros, formularios, carpetas, directorios, etc.

La información debía ser trasladada desde el papel al formato digital y por lo general, era necesario almacenarla para su posterior recuperación, consulta y procesamiento. De este modo, para llevar a cabo un tratamiento eficiente de ésta, era necesario establecer métodos adecuados para su almacenamiento. El **elemento que permitió llevar a cabo el almacenamiento de datos de forma permanente en dispositivos de memoria masiva fue el fichero o archivo**.

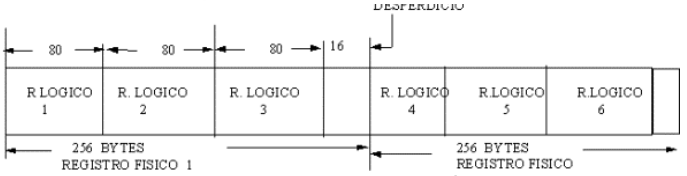
***Fichero o archivo de datos**: conjunto de información relacionada, tratada como un todo y organizada de forma estructurada de forma que se permita la búsqueda de datos individuales. Son identificados por un nombre y la descripción de la carpeta o directorio que lo contiene.*

Los **ficheros de datos** están **formados por registros lógicos** que **contienen datos relativos a un mismo elemento u objeto** (por ejemplo, los datos de usuarios de una plataforma educativa). A su vez, los **registros** están **divididos en campos que contienen cada una de las informaciones elementales** que forman un registro (por ejemplo, el nombre del usuario o su dirección de correo electrónico).

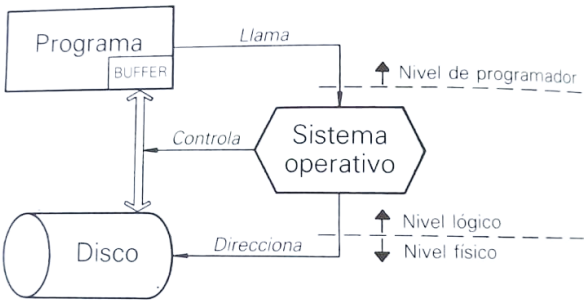
Se ha de resaltar que los datos están almacenados de tal forma que se puedan añadir, suprimir, actualizar o consultar individualmente en cualquier momento.



Como los **ficheros suelen ser muy voluminosos**, solo **se pueden llevar a la memoria principal partes de ellos** para poder procesarlos. La **cantidad de información que es transferida por el *Sistema Operativo* (*SO*) entre el soporte en el que se almacena el fichero, y la memoria principal del ordenador, en una sola operación de lectura/escritura**, recibe el nombre de **registro físico o bloque**.



Normalmente en cada operación de lectura/escritura se transfieren varios registros lógicos del fichero, es decir un bloque suele contener varios registros lógicos. Al **número de registros lógicos que entran en un bloque** se le conoce con el nombre de **factor de blocaje**, y a esta **operación de agrupar varios registros en un bloque se le llama bloqueo de registros**.



El *SO* realiza, además, la transformación de la dirección lógica (posición relativa que ocupa el registro en el fichero) usada en los programas de usuario en la dirección física (posición real o efectiva donde se encuentra el registro en el soporte de información) con la que se direcciona en el soporte.

Una **clave** o identificativo es un **campo (o conjunto de campos) que identifica cada registro del fichero**. En un mismo fichero puede haber una, varias o ninguna clave. Cuando una **clave se utiliza como campo de localización en el fichero** se denomina **llave** (*key*).

Los **ficheros pueden estar formados por registros de** uno de los siguientes tipos:

- ✓ **Longitud fija**: la suma de las longitudes de cada campo (en caracteres) es siempre la misma.
- ✓ **Longitud variable** (longitudes de los campos no siempre es la misma): el sistema reserva una **palabra al comienzo de cada registro para anotar su longitud**.
- ✓ **Delimitados**: el sistema **incluye un carácter especial al final de cada registro**.
- ✓ **Indefinido**: en este caso el *SO* no realiza ninguna gestión sobre la longitud de los registros del fichero. **Será el propio programa del usuario el que se encargue de localizar el principio y el final de cada registro**.

2.2 Tipos de ficheros.

En una aplicación informática se pueden utilizar ficheros para realizar funciones diversas. Conocer la función que va a desempeñar un fichero concreto es fundamental a la hora de decidir cómo se va a organizar éste. **Según la función que vayan a desempeñar los ficheros**, éstos **pueden ser clasificados** de varias maneras:

- ➢ **Ficheros permanentes**: contienen **información relevante para una aplicación** (los datos necesarios para el funcionamiento de ésta). **Su vida es larga** y generalmente **no pueden generarse de una forma inmediata a partir de otros ficheros**. Estos **se subdividen en**:
  - ✓ **Ficheros maestros**: contienen el **estado actual de los datos que pueden modificarse** desde la aplicación. Es la **parte central de la aplicación**, su núcleo. Podría ser un archivo con los datos de los usuarios de una plataforma educativa, un archivo de clientes de un banco, etc.
  - ✓ **Ficheros constantes**: son aquellos que **incluyen datos fijos para la aplicación**. **No suelen ser modificados** (o suele ser infrecuente) y **se accede a ellos para la realización de consultas**. Podría ser un archivo con códigos postales, la ubicación de estantes en una biblioteca, una tabla de números primos, etc.
  - ✓ **Ficheros históricos**: contienen **datos que fueron considerados como actuales en un periodo o situación**

anterior. Se utilizan para la reconstrucción de situaciones (actual o anteriores). Podría ser un archivo con los usuarios que han sido dados de baja en la plataforma educativa.

- **Ficheros temporales**: se utilizan **para almacenar información útil para una parte de la aplicación**, no para toda ella. Son **generados a partir de datos de ficheros permanentes**. **Tienen un corto periodo de existencia**. Estos **se subdividen en**:
  - ✓ **Ficheros intermedios**: almacenan **resultados de una aplicación que serán utilizados por otra**, dentro de una misma tarea.
  - ✓ **Ficheros de maniobras**: almacenan **datos de una aplicación que no pueden ser mantenidos en memoria** principal por falta de espacio.
  - ✓ **Ficheros de resultados**: almacenan **datos que van a ser transferidos a un dispositivo de salida**.

Autoevaluación

Suponer una aplicación informática para gestionar una biblioteca, existirá un fichero con el catálogo de libros disponibles, otro con las editoriales, otro con información sobre los libros que se han quedado obsoletos, etc. ¿A cuál de los siguientes tipos correspondería el fichero que almacena las editoriales?

- ○ Fichero maestro.
- ○ Fichero constante.
- ○ Fichero intermedio.

2.3 Los soportes de información.

Los ficheros se almacenan en soportes de información manejados por dispositivos periféricos del ordenador, que permiten leer y grabar datos en el soporte. Los **soportes más utilizados** para almacenar los ficheros son las **cintas magnéticas y** los **discos** (magnéticos, ópticos, magneto-ópticos o estado sólido). Dentro de estos tipos de soporte existen en el mercado una gran variedad de modelos. El **tamaño de los ficheros está limitado por el tamaño de los dispositivos que los albergan y el sistema de archivos implantado** (por ejemplo, *FAT32* no permite más de 4 *GB*).

Inicialmente, los primeros sistemas de almacenamiento físico eran los tambores de cinta magnética que tenían unas dimensiones parecidas a los discos de vinilo. Estos tambores funcionaban de manera similar a los antiguos casetes, pero sus mayores dimensiones les permitían almacenar gran cantidad de datos en formato digital, es decir con ceros y unos, en orden secuencial.

Posteriormente, los sistemas de almacenamiento de información comenzaron a cambiar de la mano de los avances en el hardware, en concreto con la aparición del disquete y el disco duro. Eran dispositivos de acceso aleatorio, no siendo necesario en ellos pasar por todos los datos desde el inicio hasta la zona donde se encuentra la información que nos interesa.

Por tanto, **se distinguen dos tipos de soportes para el almacenamiento de datos**:

- ✓ **Soportes de acceso secuencial o no direccionables** (ej.: cintas magnéticas): se suelen usar en copias de seguridad y **si se desea leer un dato** que está en la mitad, **se tendrá que leer todo lo que hay hasta llegar a esa posición**.
- ✓ **Soportes de acceso directo o direccionables** (ej.: discos): son los más empleados y **el acceso a los datos puede hacerse de forma directa, pudiendo acceder directamente a la posición que interese y leer a partir de ella** (no es necesario recorrer o leer otros bloques).

2.4 Parámetros de utilización.

En función del **uso que se le vaya a dar al fichero, serán adecuados unos tipos u otros de organización**. Mediante la **utilización de parámetros de referencia, podremos determinar el uso de un fichero**. Estos parámetros son:

a) **Capacidad o volumen**: es el **espacio**, en caracteres, **que ocupa el fichero**. La capacidad podrá calcularse multiplicando el número previsto de registros por la longitud media de cada registro.

b) **Actividad**: permite conocer la **cantidad de consultas y modificaciones que se realizan en el fichero**. Para poder especificar la actividad se deben tener en cuenta:

- **Tasa de consulta o modificación**: que es el **porcentaje de registros consultados o modificados en cada tratamiento del fichero**, respecto al número total de registros contenidos en él.
- **Frecuencia de consulta o modificación**: número de **veces que se accede al fichero para hacer una consulta o modificación en un periodo de tiempo fijo**.

c) **Volatilidad**: mide la cantidad de **inserciones y borrados que se efectúan en un fichero**. Para determinar la volatilidad es necesario conocer:

- **Tasa de renovación**: es el **porcentaje de registros renovados en cada tratamiento del fichero**, respecto al número total de registros contenidos en él.
- **Frecuencia de renovación**: es el número de **veces que se accede al fichero para renovarlo en un periodo de tiempo** fijo.

d) **Crecimiento**: es la **variación de la capacidad del fichero** y se mide con la **tasa de crecimiento**, que es el **porcentaje de registros en que aumenta el fichero en cada tratamiento**.

Autoevaluación

La volatilidad de un fichero es un parámetro que indica:

- La variación del volumen del fichero.
- La cantidad de veces que se abre o cierra el fichero.
- La cantidad de inserciones y borrados en dicho fichero.

2.5 Organización de ficheros.

A medida que la tecnología ha ido evolucionando, atendiendo principalmente a los **avances hardware**, el acceso a la información contenida en los diferentes tipos de ficheros ha cambiado mucho.

Los **objetivos fundamentales** de estas modificaciones pueden resumirse en los siguientes puntos:

- ✓ Proporcionar un **acceso rápido** a los registros.
- ✓ Conseguir **economizar el almacenamiento**.
- ✓ **Facilitar la actualización de los registros**.
- ✓ Permitir que la estructura refleje la organización real de la información.

Hay **distintas formas de estructurar u organizar los ficheros sobre un soporte de información**. Las características de utilización del fichero dependen de la organización que se adopte. Se debe, pues, **optar por una u otra organización, atendiendo a la forma en que se va a usar el fichero**. Las principales organizaciones de ficheros se detallan en los apartados siguientes.

2.5.1 Organización secuencial.

Un fichero con organización secuencial se caracteriza porque **sus registros están almacenados de forma contigua**, de manera que **la única forma de acceder a él, es leyendo un registro tras otro desde el principio hasta el final**. En los ficheros secuenciales suele haber una marca indicativa del fin del fichero, que suele denominarse *EOF* (*End of File*).

Este tipo de ficheros **pueden utilizar dispositivos** o soportes **no direccionables** o de acceso secuencial, como son las cintas magnéticas. También se utilizan en los *CD* de audio y los *DVD* de vídeo, en los que la música o las imágenes se almacenan a lo largo de una espiral continua.

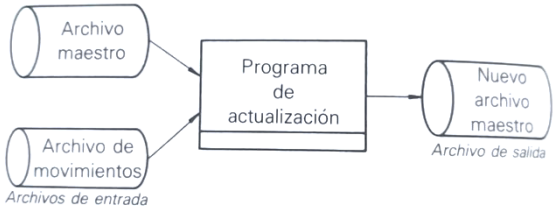
Los **registros almacenados se identifican por medio de una información ubicada en uno de sus campos**, a este campo se le denomina **clave** o **llave**. Si se ordena un archivo secuencial por su clave, es más rápido realizar cualquier operación de lectura o escritura.

Otras **características** relevantes de los ficheros secuenciales son:

- ✓ La **lectura siempre se realiza hacia delante**.
- ✓ Tienen una **estructura rígida de campos**, es decir, la posición de los campos de cada registro siempre ha de ser la misma.
- ✓ Son **ficheros monousuario**, no permiten el acceso simultáneo de varios usuarios.
- ✓ **Aprovechan al máximo el soporte de almacenamiento**, al no dejar huecos vacíos.

- ✓ **Se pueden grabar en cualquier tipo de soporte**, tanto en **secuenciales** como **direccionables**.
- ✓ **Todos los lenguajes de programación disponen de instrucciones para trabajar con este tipo de ficheros**.
- ✓ **No se pueden insertar registros entre los que ya están grabados**.

No es posible realizar fácilmente las operaciones de inserción, modificación o borrado sobre un fichero secuencial. Si se necesita actualizar un archivo con organización secuencial se debe crear de nuevo el archivo (esto se realiza mediante un programa escrito para tal fin).

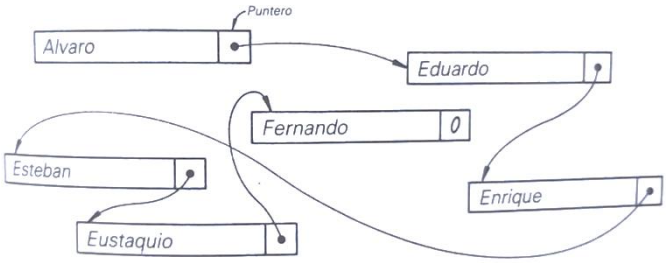


Tan sólo es posible añadir un nuevo registro de forma fácil al final del fichero.

La consulta se realiza en orden secuencial.

2.5.2 Organización secuencial encadenada.

En un fichero con organización secuencial encadenada, **junto a cada registro se almacena un puntero (enlace) con la dirección del registro siguiente, según el orden lógico del fichero**. Los ficheros con esta organización **sólo pueden ser gestionados en soportes direccionables**.



La **consulta es secuencial**, cada vez que se lee un registro se lee la posición del siguiente, lo que permite seguir la secuencia lógica del archivo.

Para **insertar un registro** es necesario, en primer lugar, **localizar la posición en que se desea insertar** (entre qué dos registros se quiere que aparezca). El **registro se crea en una zona libre y se modifican los punteros** (enlaces) para que el registro quede insertado entre los dos registros deseados.

Para **borrar un registro** se **asigna al puntero del registro anterior la dirección del registro siguiente** al que se desea borrar. El *SO* puede o no liberar el espacio ocupado por el registro borrado.

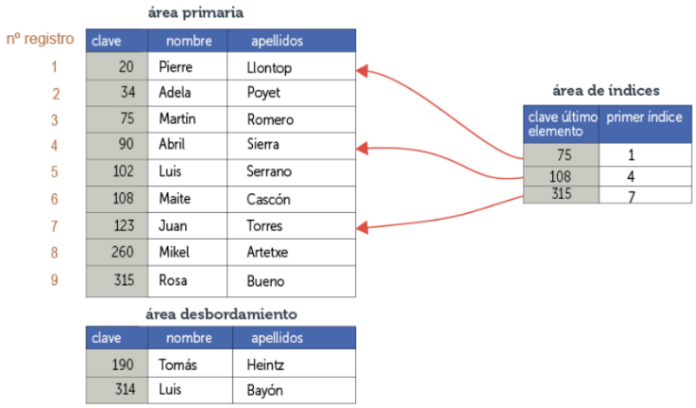
Al **modificar**, si **no implica un aumento de la longitud del registro, éste puede reescribirse en el mismo espacio**. En el **caso de que el registro aumente de longitud se debe insertar el registro y posteriormente borrar la versión anterior a la modificación**.

La principal ventaja de esta organización es su flexibilidad, y su principal inconveniente es su limitación a consulta secuencial.

2.5.3 Organización secuencial indexada.

Un fichero con organización secuencial indexada está **formado por tres áreas o zonas distintas**:

- ✓ **Zona de registros** (área primaria o de datos): los registros son grabados en un **soporte de almacenamiento directo**, en **secuencia ascendente, de acuerdo con los valores de la clave** y en bloques de longitud fija.
- ✓ **Zona de índices** (área de índices): se procesa de forma secuencial y contiene una **tabla que asocia las claves con las direcciones de los registros en la zona de registros**. Cada **entrada está formada por el valor más alto de la clave de cada conjunto de registros y un puntero con la dirección del primer registro del grupo**.
- ✓ **Zona de desbordamiento** (área de desbordamiento o excedentes): sitio donde **se graban los registros que no caben en la zona de registros**.



La zona de registros está dividida en tramos lógicos. Cada tramo está **formado por una serie de registros consecutivos**. **Por cada tramo de la zona de registros hay un registro en la zona de índices**, que indica el valor de la clave del último elemento del tramo y la posición por la que comenzar para recorrer el tramo.

Para permitir actualizaciones es necesario incluir en la estructura una zona de desbordamientos, en esta zona los registros están desordenados y cada nuevo registro se escribe al final en esta zona.

Esta organización es muy **utilizada**, tanto para **procesos en los que intervienen pocos registros** como para **aquellos en los que se maneja el fichero completo**.

Las principales **características** son:

- ✓ **Permite el acceso secuencial**: los registros se leen ordenados por el campo clave.
- ✓ **Permite el acceso directo a los registros**: realmente emula el acceso directo, primero busca la clave en el área de índices y luego va a leer al área de datos en la dirección que le indica la tabla.
- ✓ **Se pueden actualizar los registros en el mismo fichero**: no es necesario crear un nuevo fichero de copia en el proceso de actualización.
- ✓ **Ocupa más espacio en el disco que los ficheros secuenciales**: debido al uso del área de índices.
- ✓ **Únicamente se pueden utilizar con soportes direccionables**.

2.5.4 Organización directa o aleatoria.

En este tipo de ficheros **se puede acceder a un registro indicando la posición relativa del mismo dentro del archivo** o, más comúnmente, **a través de una clave** que forma parte del registro como un campo más. Estos archivos **deben almacenarse en dispositivos de almacenamiento de acceso directo**, como son los discos.

> ***Campo clave**: campo que permite identificar y localizar un registro de manera ágil y organizada.*

Cada uno de los registros se guarda en una posición física, que dependerá del espacio disponible en memoria masiva (disco), de ahí que **la distribución de los registros** sea **aleatoria dentro del soporte de almacenamiento**. **Para acceder a la posición física de un registro se utiliza una dirección o índice, no siendo necesario recorrer todo el fichero para encontrar un determinado registro**.

Algunas **características** fundamentales de los ficheros de acceso directo o aleatorio son:

- ✓ **Posicionamiento inmediato**.
- ✓ **Registros de longitud fija**.
- ✓ **Permiten múltiples usuarios** utilizándolos.
- ✓ Los **registros se borran colocando un cero en la posición que ocupan**.
- ✓ Permiten la **utilización de algoritmos de compactación de huecos**.
- ✓ Esta organización **sólo es posible en soportes direccionables**.
- ✓ Los archivos **se crean con un tamaño definido**, es decir, **con un máximo de registros** establecido durante la creación.
- ✓ Se usan cuando el acceso a los datos de un registro se hace siempre empleando la misma clave y la **velocidad de acceso a un registro es lo que más importa**.
- ✓ **Permiten la actualización de los registros en el mismo fichero**, sin necesidad de copiar el fichero.
- ✓ Permiten realizar **procesos de actualización en tiempo real**.

A través de una transformación específica aplicada a la clave, se obtendrá la dirección física en la que se encuentra el registro. El **problema fundamental de esta organización es la elección de la transformación que se ha de usar**, y en algunos casos **aparecen dos situaciones no deseadas**: **direcciones que no corresponden a ninguna clave** (zonas de disco sin utilizar) y **direcciones que corresponden a más de una clave** (da lugar a lo que se conoce como **sinónimos**).

Hay **dos formas de resolver el problema de los sinónimos**, siempre a costa de complicar la estructura del fichero:

1. Cuando se asocia a una clave una dirección ya ocupada por un registro distinto, **se busca en el archivo, por algún procedimiento, una posición libre donde escribir el registro**.
2. **Se reserva una zona de desbordamiento donde se escribirán los registros que no puedan escribirse en la posición que les corresponde** según la transformación.

Según la forma de realizar esta transformación, existen diferentes **modos de acceso**:

2.5.4.1 Direccionamiento directo.

Es **factible cuando la clave es numérica y su rango de valores no es mayor que el rango de direcciones en un archivo**. **Se utiliza como dirección la propia clave**.

En algunos casos **pueden quedar lagunas de direcciones sin utilizar**, en lugares conocidos de antemano.

Un archivo aleatorio con direccionamiento directo **siempre está ordenado por la clave**.

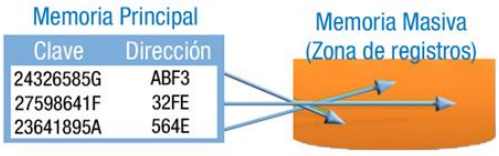
Ejemplo: suponer que se tiene un fichero de datos para almacenar información de empleados. Cada empleado tiene un número de identificación único y se utilizará ese número como clave numérica para el direccionamiento directo.

- Estructura del fichero: el fichero de datos tiene una estructura predefinida en la que cada registro ocupa una posición fija en el fichero. Por ejemplo, si cada registro ocupa 100 bytes, el primer registro estará en la posición 0-99, el segundo registro en la posición 100-199, y así sucesivamente.
- Asignación de claves: se asigna a cada empleado un número de identificación único que servirá como clave numérica para acceder a su registro correspondiente en el fichero. Por ejemplo, el empleado con número de identificación 1002 tendrá su registro en la posición 100100-100199.
- Direccionamiento directo: para acceder a un registro de empleado en el fichero, se utiliza su número de identificación como clave para calcular su ubicación física en el fichero. Por ejemplo, si se quiere acceder al empleado con número de identificación 1001, se utiliza la siguiente fórmula:

$$posición = (número\_identificación - 1) * 100$$

El direccionamiento directo con clave numérica permite acceder directamente a un registro en el fichero utilizando la clave numérica como índice para calcular su ubicación física. Esto evita la necesidad de buscar secuencialmente a través de los registros, lo que proporciona un acceso rápido y eficiente a los datos. Es importante tener en cuenta que el fichero debe estar organizado de manera que los registros tengan una ubicación fija y predecible con base en la clave numérica para que el direccionamiento directo funcione correctamente.

2.5.4.2 Direccionamiento asociado.



Se puede utilizar para cualquier tipo de clave. Si se utiliza este método **debe construirse una tabla en la que figurarán todas las claves y la dirección donde se encuentra el registro correspondiente**.

Al **añadir nuevos registros** las **claves se colocan al final de la tabla**, normalmente desordenada. Por tanto, habrá que localizar las claves en ella por lectura secuencial, lo que ralentiza el acceso, a menos que la tabla se cargue en memoria principal o se ordene.

Ejemplo: suponer que se tiene un fichero de datos de empleados y cada registro en el fichero tiene una clave única, como el número de identificación del empleado.

- Creación del fichero: al crear el fichero de datos, se asigna un identificador único a cada registro. Además, se mantiene una tabla de índices donde se almacena la relación entre la clave y la posición del registro en el fichero.
- Inserción de registros: cuando se inserta un nuevo registro en el fichero, se le asigna una clave única y se guarda en la siguiente posición disponible. Luego, se actualiza la tabla de índices para asociar la clave con la posición del registro en el fichero.
- Búsqueda de registros: para buscar un registro en el fichero, se utiliza la clave asociada al registro. Se busca la clave en la tabla de índices para obtener la posición del registro en el fichero. A continuación, se accede directamente a esa posición y se recupera el registro correspondiente.
- Actualización de registros: si se necesita actualizar un registro en el fichero, se utiliza la clave para buscar la posición del registro en la tabla de índices. Una vez localizada la posición, se actualiza el registro directamente en esa posición en el fichero.

El direccionamiento asociado permite un acceso directo a los registros utilizando la clave asociada a cada uno. Esto evita la necesidad de recorrer todo el fichero para buscar el registro deseado, lo que resulta en un acceso más eficiente y rápido. Además, la tabla de índices proporciona una forma de mapear las claves a las posiciones físicas en el fichero, facilitando la recuperación y actualización de los registros.

Es importante destacar que el direccionamiento asociado puede requerir una estructura adicional, como la tabla de índices, para mantener la relación entre las claves y las posiciones de los registros en el fichero.

2.5.4.3 Direccionamiento calculado (hashing).

Cuando se utilizan ficheros con direccionamiento asociado es necesario siempre tener que consultar una tabla para obtener la dirección de almacenamiento a partir de la clave. La técnica del acceso calculado o *hash*, **permite accesos más rápidos**, ya que, en lugar de consultar una tabla, **se utiliza una transformación o función matemática (función de *hashing*) conocida, que a partir de la clave genera la dirección de cada registro del archivo**. Si la clave es alfanumérica, deberá previamente ser transformada en un número.

El mayor **problema** que presenta este tipo de ficheros es que **a partir de diferentes claves se obtenga la misma dirección al aplicar la función matemática** o transformación. A este problema se le denomina **colisión**, y las claves que generan la misma dirección se conocen por sinónimos. **Para resolver este problema se aplican diferentes métodos: tener un bloque de excedentes o zona de sinónimos, crear un archivo de sinónimos, etc.**

Para llevar a cabo la **transformación** existen multitud de **métodos**:

- ✓ **Módulo**: la dirección será igual al **resto de la división entera entre la clave y el número de registros**.
- ✓ **Extracción**: la dirección será igual a una **parte de las cifras que se extraen de la clave**.
- ✓ **Elevación al cuadrado**: se **eleva al cuadrado la clave y se toman los dígitos centrales**.
- ✓ **Plegamiento**: se **descompone la clave en segmentos de cifras contiguas del mismo tamaño, y se suman**.

Una **buena** transformación o **función de hash**, será **aquella que produzca el menor número de colisiones**. En este caso hay que buscar una función, a ser posible biunívoca, que relacione los posibles valores de la clave con el conjunto de números correlativos de dirección. Esta función consistirá en realizar una serie de cálculos matemáticos con el valor de la clave hasta obtener un número entre 1 y n, siendo n el número de direcciones que tiene el fichero.

Ejemplo: suponer que se tiene un fichero de datos que almacena información de productos. Cada registro del fichero contiene un campo de clave única, como el código de producto, y otros campos con información relevante sobre el producto.

- Función de *hash*: se define una función de *hash* que toma el código de producto y genera un número *hash* único correspondiente a ese producto. Por ejemplo, la función de *hash* podría ser el cálculo del módulo del código de producto por el tamaño del fichero:

  valor_hash = código_producto % tamaño_fichero

- Direccionamiento: para insertar un nuevo producto en el fichero de datos, se calcula su número *hash* utilizando la función de *hash* y se utiliza ese valor como dirección para ubicar el registro en el fichero. Por ejemplo, si el código de producto es "P12345", se aplica la función de *hash*:

  valor_hash = hash("P12345") % tamaño_fichero

  Luego, se inserta el registro del producto en la posición calculada en el fichero de datos.

- Búsqueda: para buscar un producto en el fichero de datos, se calcula su número *hash* utilizando la función de *hash* y se accede a esa dirección para obtener el registro correspondiente. Por ejemplo, si se quiere buscar el producto con código "P67890", se aplica la función de *hash*:

  valor_hash = hash("P67890") % tamaño_fichero

  Luego, se accede al registro en la posición calculada en el fichero y se obtiene la información del producto.

El direccionamiento calculado *hashing* en ficheros de datos utiliza la función de *hash* para calcular la ubicación de los registros en el fichero, permitiendo un acceso eficiente a través de la dirección calculada en lugar de recorrer todo el fichero en busca del registro deseado.

Autoevaluación

En los ficheros de acceso directo los registros siempre se encuentran en posiciones contiguas dentro del soporte de almacenamiento. ¿Verdadero o falso? Razona la respuesta.

2.6 Inconvenientes de los sistemas de gestión de ficheros.

Antes de aparecer los *SGBD* (*Sistemas Gestores de Bases de Datos*), como se ha visto, **la información se gestionaba utilizando los sistemas de gestión de ficheros**. **Los datos se guardaban en ficheros y los programas**

manejaban esos ficheros para obtener la información. Si la estructura de los datos cambia en los ficheros, todos los programas que los manejan se deben modificar. En estos sistemas de gestión de ficheros, **la definición de los datos se encuentra codificada dentro de los programas** en lugar de almacenarse de forma independiente, y además **el control del acceso y la manipulación de los datos vienen impuestos por los programas**.

Esto supone un gran inconveniente a la hora de tratar grandes volúmenes de información. **Surge así la idea de separar los datos contenidos en los ficheros de los programas que los manipulan** (es decir, que se pueda modificar la estructura de los datos de los ficheros sin que por ello se tengan que modificar los programas).

Inconvenientes de los sistemas de gestión de ficheros:

- ✓ **Redundancia e inconsistencia de los datos**: los ficheros son creados por distintos programas y van cambiando con el tiempo, por lo que pueden tener distintos formatos y los datos pueden estar duplicados.
- ✓ **Dependencia de los datos física-lógica**: la estructura física de los datos (definición de ficheros y registros) se encuentra codificada en los programas.
- ✓ **Dificultad para tener acceso a los datos**: cada vez que se necesite una consulta no prevista es necesario codificar el programa correspondiente.
- ✓ **Separación y aislamiento de los datos**: al estar repartidos en varios ficheros con diferentes formatos, es difícil escribir nuevos programas que aseguren la manipulación correcta de los datos.
- ✓ **Dificultad para el acceso concurrente**: es complicado que los usuarios actualicen los datos simultáneamente.
- ✓ **Dependencia de la estructura del fichero con el lenguaje de programación**: la estructura se define dentro de los programas, la incompatibilidad entre ficheros generados por distintos lenguajes de programación hace que los datos sean difíciles de procesar.
- ✓ **Problemas en la seguridad de los datos**: difícil implantar restricciones de seguridad ya que las aplicaciones se van añadiendo al sistema según se van necesitando.
- ✓ **Problemas de integridad de datos**: los valores almacenados en los ficheros deben cumplir con restricciones de consistencia (no se puede crear un pedido si el cliente no existe).

Todos estos inconvenientes hacen posible el fomento y desarrollo de las *BD* y los *SGBD*.

TAREA

1. **Imaginar que se va a desarrollar una aplicación para llevar el control de las ventas de ordenadores de una tienda de informática y en principio no se va a utilizar ningún gestor de *BD*. ¿Se podría crear dicha aplicación? ¿de qué elemento se dispone en el *SO* para guardar la información? ¿qué problemas va a plantear este desarrollo? ¿de qué tipos pueden ser los elementos utilizados para guardar la información? Poner un ejemplo de la información que se guardaría en cada uno de los tipos.**
2. **¿Qué es un fichero de datos? Crear una tabla con las ventajas e inconvenientes que presentan las distintas formas de organizar los archivos o ficheros de datos.**
3. **Investigar en *Internet* sobre los sistemas lógicos de almacenamiento (sistemas de archivos o ficheros). ¿Qué son? ¿qué características presentan? ¿cuáles son los más utilizados hoy en día?, …**

3 Bases de datos.

Como se ha visto anteriormente, **los ficheros permiten organizar y almacenar conjuntos de datos del mismo tipo o naturaleza con una determinada estructura**, siendo un medio para el almacenamiento de los datos o resultados de una aplicación específica. **Pero si las aplicaciones, al ser diseñadas, deben depender directamente de sus ficheros o archivos, se pierde independencia y surgen serios inconvenientes: como información duplicada, incoherencia de datos, fallos de seguridad, etc.**

Estos problemas debían ser solucionados, es cuando aparece el concepto de *BD*. **Una *BD* permitirá reunir toda la información relacionada en un único sistema de almacenamiento, pudiendo cualquier aplicación utilizarla de manera independiente y ofreciendo una mejora en el tratamiento de la información, así como una evolución para el desarrollo de aplicaciones**.

Las *BD* son instrumentos **de gran utilidad para gestionar grandes ficheros y facilitar la consulta de información**. En muchas, además, **puede definirse un esquema de permisos que establece qué personas o programas pueden acceder a los datos, y a cuáles**, con el objetivo de presentar el contenido de forma adecuada y clara.

Los distintos sistemas de *BD* se diferencian conceptualmente entre sí y tienen, por lo tanto, sus propias ventajas y desventajas. Pero, antes que nada, es conveniente diferenciar entre la *BD* en sí y el sistema que la gestiona. Como ***BD*** se designa al **conjunto de los datos que se ha de almacenar**, mientras que el ***SGBD*** es **responsable de su administración**, determinando así su estructura, el orden, los permisos de acceso, las dependencias, etc.

La gestión de las *BD* ha experimentado gran cantidad de cambios, partiendo de aplicaciones especializadas **hasta llegar a convertirse en el núcleo de los entornos informáticos modernos**. Con la llegada de *Internet* en los noventa, el número de usuarios de *BD* creció exponencialmente, y aunque muchos de ellos no sean conscientes de ello, el acceso a dichas *BD* forma parte de la vida cotidiana de muchos de nosotros.

Conocer los sistemas que gestionan las *BD*, sus conceptos fundamentales, el diseño, lenguajes y la implementación de éstas, se puede considerar imprescindible para alguien que se está formando en el campo de la informática.

3.1 Historia y necesidad del uso de las *BD*.

Para aumentar la eficiencia estructural del tratamiento electrónico de los datos, ya en la década de los 60, se empezó a desarrollar el concepto de la *BD* electrónica como capa separada de software entre el *SO* y el programa de aplicación. Esto fue el resultado de la experiencia del día a día, pues tanto manipular los archivos como supervisar y repartir los permisos adquirió tal complejidad que el procesamiento electrónico de los datos no significó un avance real. Así, la idea del sistema de *BD* electrónica se convirtió en una de las innovaciones más relevantes en el desarrollo del ordenador.

Los primeros modelos que se desarrollaron fueron las *BD* en red y jerárquicas, si bien pronto demostraron ser demasiado simples y estar limitadas técnicamente. *IBM* fue la empresa que revolucionó el sector, con el desarrollo del modelo relacional de *BD* en los años setenta, con mucho el más potente, que pronto encontró un campo de cultivo favorable en el mundo laboral. Los productos que más éxito tuvieron en ese momento, fueron el lenguaje de consultas a *BD SQL* de *Oracle* y los sucesores de *IBM*, *SQL/DS* y *DB2*.

Hasta bien entrados los años 2000, cuando algunos proyectos de código libre metieron algo de aire fresco al sector, el mercado del software de *BD* estuvo gobernado por los pesos pesados. Entre los sistemas libres más populares se encuentran *MySQL* y *PostgreSQL*. La tendencia iniciada en 2001 hacia los sistemas *NoSQL* también contribuyó a la debilitación de la posición de los sistemas de *BD* de los grandes fabricantes.

Hoy en día, los sistemas de *BD* son imprescindibles en numerosos campos. **Cualquier tipo de software concebido para las empresas se basa en robustas *BD*** con un gran número de opciones y herramientas para los administradores del sistema. La seguridad de los datos, además, ha ido ganando importancia con el tiempo, y es que en las *BD* se almacenan y cifran contraseñas, datos personales e incluso divisa digital.

El sistema financiero moderno, no es más que una red de *BD*, en la cual, la mayor parte de las cuantías monetarias solo existen como unidades electrónicas de información, cuya protección, por medio de *BD* seguras es una de las tareas principales de las instituciones financieras. Aunque no solo por esto son cruciales las *BD* electrónicas para la civilización moderna.

3.2 Conceptos y ventajas del uso de *BD*.

A finales de los setenta, la aparición de nuevas tecnologías de manejo de datos a través de los sistemas de *BD* supuso un considerable cambio. Los sistemas basados en ficheros separados dieron paso a la utilización de ***SGBD***, que son **sistemas software centralizados o distribuidos que ofrecen facilidades para la definición de *BD*, selección de estructuras de datos y búsqueda de forma interactiva o mediante lenguajes de programación**.

La *BD* no sólo contiene los datos de la organización, también almacena una descripción de dichos datos. Esta descripción es lo que se denomina metadatos, se almacena en el diccionario de datos o catálogo y es lo que permite

que exista independencia de datos lógica-física.

> ***Base de datos (BD)****: es una colección de datos relacionados lógicamente entre sí, con una definición y descripción comunes y que están estructurados de una determinada forma, almacenados con la mínima redundancia y posibilitando el acceso a ellos eficientemente por parte de varias aplicaciones y usuarios.*

Las **ventajas fundamentales que ofrece el uso de *BD*** se resumen a continuación:

- ✓ **Acceso múltiple**: diversos usuarios o aplicaciones podrán acceder a la *BD*, sin que existan problemas en el acceso o los datos.
- ✓ **Utilización múltiple**: cada uno de los **usuarios o aplicaciones podrán disponer de una visión particular de la estructura de la *BD***, de tal manera que cada uno de ellos accederá sólo a la parte que realmente le corresponde.
- ✓ **Flexibilidad**: la forma de acceder a la información puede ser establecida de diferentes maneras, ofreciendo tiempos de respuesta muy reducidos.
- ✓ **Confidencialidad y seguridad**: el control del acceso a los datos podrá ser establecido para que unos usuarios o aplicaciones puedan acceder a unos datos y a otros no, impidiendo a los usuarios no autorizados la utilización de la *BD*.
- ✓ **Protección contra fallos**: en caso de errores en la información, existen mecanismos bien definidos que permiten la recuperación de los datos de forma fiable.
- ✓ **Independencia física**: un cambio de soporte físico de los datos (por ejemplo, el tipo de discos), no afectaría a la *BD* o a las aplicaciones que acceden a ellos.
- ✓ **Independencia lógica**: los cambios en la *BD* no afectan a las aplicaciones que la usan.
- ✓ **Redundancia**: los datos se almacenan, por lo general, una única vez. Aunque si es necesario, se podría repetir información de manera controlada.
- ✓ **Interfaz de alto nivel**: mediante la utilización de lenguajes de alto nivel puede utilizarse la *BD* de manera sencilla y cómoda.
- ✓ **Consulta directa**: existen herramientas para poder acceder a los datos interactivamente.

Autoevaluación

Una *BD* es:

- Un programa para gestionar archivos muy grandes.
- El conjunto de datos de los usuarios almacenados en un único disco duro.
- Conjunto de datos de distinto tipo relacionados entre sí, almacenados con la mínima redundancia y posibilitando el acceso a ellos eficientemente.
- Un conjunto de programas que permiten la gestión de los datos.

3.3 ¿Quién utiliza las *BD* y para qué?

Ya se ha visto lo que es una *BD* y sus características principales, pero es necesario conocer quien las usa y para qué. **¿Quién utiliza las *BD*?**

Tipo	Funciones y características
El administrador	Es la persona **encargada de la creación o implementación física de la *BD***. Es quien **escoge los tipos de ficheros, los índices que hay que crear, la ubicación de éstos, etc.** En general, es quien toma las decisiones relacionadas con el funcionamiento físico del almacenamiento de información, siempre teniendo en cuenta las posibilidades del sistema de información con el que trabaje. Junto a estas tareas, el administrador establecerá la política de seguridad y de acceso para garantizar el menor número de problemas.
Los diseñadores	Son las **personas encargadas de diseñar cómo será la *BD***. Llevarán a cabo la **identificación de los datos, las relaciones entre ellos, sus restricciones, etc.** Para ello han de conocer a fondo los datos y procesos a representar en la *BD*. Si estamos hablando de una empresa, será necesario que conozcan las reglas de negocio en la que esta se mueve. Para obtener un buen resultado, el diseñador de la *BD* debe implicar en el proceso a todos los usuarios de la *BD*, tan pronto como sea posible.

Tipo	Funciones y características
Los programadores de aplicaciones	Una vez diseñada y construida la *BD*, los programadores **se encargarán de implementar los programas de aplicación que servirán a los usuarios finales**. Estos programas de aplicación ofrecerán la posibilidad de realizar consultas de datos, inserción, actualización o eliminación de los mismos. Para desarrollar estos programas se utilizan lenguajes de programación.
Los usuarios finales	Son los **clientes finales de la *BD***. Al diseñar, implementar y mantener la *BD* se busca cumplir los requisitos establecidos por el cliente para la gestión de su información.

¿Para qué se utilizan las *BD*?

Enumerar todos y cada uno de los campos donde se utilizan las *BD* es complejo, aunque seguro que quedarán muchos en el tintero, a continuación, se recopilan **algunos de los ámbitos donde se aplican**:

- ✓ **Banca**: información de clientes, cuentas, transacciones, préstamos, …
- ✓ **Líneas aéreas**: información de clientes, horarios, vuelos, destinos, …
- ✓ **Universidades**: información de estudiantes, carreras, horarios, materias, …
- ✓ **Transacciones de tarjeta de crédito**: para comprar con tarjetas de crédito y la generación de los extractos mensuales.
- ✓ **Telecomunicaciones**: para guardar registros de llamadas realizadas, generar facturas mensuales, mantener el saldo de las tarjetas telefónicas de prepago y almacenar información sobre las redes.
- ✓ **Medicina**: información hospitalaria, biomedicina, genética, …
- ✓ **Justicia y seguridad**: delincuentes, casos, sentencias, investigaciones, …
- ✓ **Legislación**: normativa, registros, …
- ✓ **Organismos públicos**: datos ciudadanos, certificados, …
- ✓ **Sistemas de posicionamiento geográfico**.
- ✓ **Hostelería y turismo**: reservas de hotel, vuelos, excursiones, …
- ✓ **Ocio digital**: juegos online, apuestas, …
- ✓ **Cultura**: gestión de bibliotecas, museos virtuales, …
- ✓ …

3.4 Ubicación de la información.

Las *BD* se utilizan a diario, pero ¿dónde se encuentra realmente almacenada la información? Las *BD* pueden tener un tamaño muy reducido (1 *MByte* o menos) o bien, ser muy voluminosas y complejas (del orden de *TBytes*). Sin embargo, **todas las *BD* normalmente se almacenan y localizan en discos duros** y otros dispositivos de almacenamiento, a los que se accede a través de un ordenador. **Una gran *BD* puede necesitar servidores en lugares diferentes**, y viceversa, **pequeñas *BD* pueden existir como ficheros en el disco duro de un único equipo**.

A continuación, se exponen los **sistemas de almacenamiento de información más utilizados para el despliegue de *BD***, comenzando por aquellos en los que pueden alojarse *BD* de tamaño pequeño y mediano, para después analizar los sistemas de alta disponibilidad de grandes servidores.

3.4.1 Discos.

Uno de los principales medios para almacenar información es mediante el uso de discos. En la actualidad existen **dos tipos** de discos:

- ✓ **Discos duros**: los discos *HDD* (*Hard Drive Disk*) son **dispositivos mecánicos que utilizan el magnetismo para grabar los datos**. Se componen de varios discos rígidos que giran a gran velocidad y cuyo cabezal de lectura/escritura se encarga de grabar o leer la información.
- ✓ **Discos de estado sólido**: las unidades de estado sólido ***SSD*** (*Solid State Drive*) **almacenan la información en chips con memorias flash** interconectadas (memorias *NAND* que mantienen la información cuando se corta el suministro eléctrico). Dentro de este tipo de dispositivos se pueden diferenciar tres tipos según su conexión: *SATA*, *M.2* y *PCIe NVME*.

Los discos suelen ser parte de cualquier sistema de almacenamiento de datos. El **sistema de disco duro *NAS*** (*Network Attached Storage*) permite colocar tantas unidades como se necesiten. Es decir, cuando se termina la capacidad, solo se debe añadir un disco duro más.

Los **sistemas *NAS*** son en sí mismos **pequeños ordenadores conectados a la red que permiten una gestión de datos empresarial muy eficiente y garantizan la seguridad** en caso de un ciberataque.

3.4.2 Cintas magnéticas.

Este tipo de dispositivo de almacenamiento **graba los datos en pistas sobre una banda plástica que cuenta con material magnético**. Existen **distintos tipos de cintas** de almacenamiento según su composición química o formatos de grabación que utilicen.

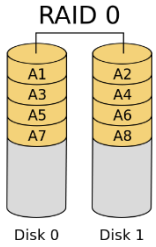
Cada sistema de almacenamiento en cinta dispone de sus propias características, como su tamaño, el tipo de contenedor del sistema, la capacidad de almacenamiento o las características magnéticas de la cinta, por ejemplo.

A pesar de que se trata de un **sistema de almacenamiento de datos antiguo**, tiene una gran utilidad en el entorno actual donde se manejan datos masivos. La **gran durabilidad** y su **reducido coste** las han convertido en un medio de almacenamiento que **es aún utilizado en la actualidad para copias de seguridad**.

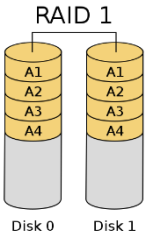
3.4.3 *RAID*.

Acrónimo de ***Redundant Array of Independent Disks*** o matriz de discos independientes, es un contenedor de almacenamiento redundante. Se basa en el montaje en **conjunto de dos o más discos duros, formando un bloque de trabajo, para obtener desde una ampliación de capacidad a mejoras en velocidad y seguridad de almacenamiento**.

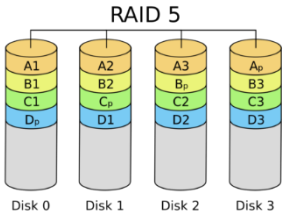
Según las características, se establecen distintos **sistemas de *RAID***:



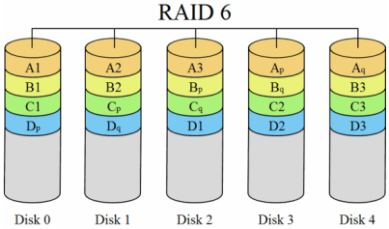
- ✓ ***RAID 0***: la **información se escribe de forma alterna en cada uno de los discos** que forman la matriz. Esto hace que los datos se envíen de forma paralela, y que la velocidad a la que fluyen los datos sea muy superior a la de un sólo disco.
  La parte negativa es que **los datos no se van a duplicar**, y al no haber redundancia, **si hay un fallo en un disco o en algún fichero interno**, se perderán los datos sin poder recuperarlos.



- ✓ ***RAID 1***: se necesitan **al menos dos discos duros** para utilizarlo, y los **datos se almacenan por igual en cada uno de los discos duros**. Por lo tanto, **si hay algún error en alguno de los discos o en alguno de los ficheros, siempre se tendrá otra copia a la que recurrir**.
  El **tamaño de la matriz es el mismo que el del disco duro de menor capacidad**, ya que los datos tienen que estar siempre por duplicado.
- ✓ ***RAID 5***: se deben tener un **mínimo de 3 discos duros**. Los **datos se distribuyen entre todos los discos** que se tengan en la matriz **salvo en uno, que almacenará una copia de los datos a forma de copia de seguridad**.
  En cuanto al resto de discos más allá del de copia de seguridad, la información se divide en bloques que se distribuyen de forma equitativa por ellos. Así, se multiplica exponencialmente la velocidad, y cuantos más discos duros se tengan mayor será la velocidad.

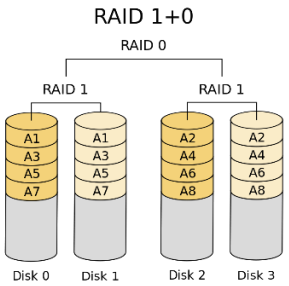


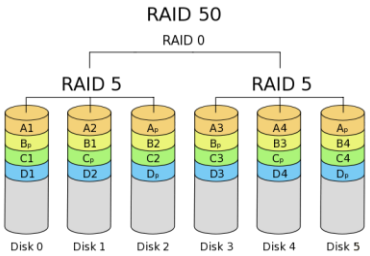
- ✓ **Otros tipos de *RAID***:
  Más allá de los tres anteriores (principales), también hay otros tipos de *RAID* que suelen ser combinaciones, pero que pueden ser **útiles sobre todo cuando se tienen cuatro o más discos duros** disponibles para la configuración de la matriz. Estos son los otros tipos más comunes:
  - ➢ ***RAID 6***: esta no es una combinación de los anteriores, sino una **variante del *RAID 5***. La diferencia es que **los datos** no se duplican en un solo disco duro, mientras se reparten en el resto, sino que **se**



duplican en dos.

- ***RAID 0+1*** o ***RAID 01***: requiere **por lo menos cuatro discos duros**, con los que **crear al menos dos matrices *RAID 0*** con cada uno de los pares de discos. Entonces, **luego se compone una matriz de *RAID 1* utilizando las dos matrices de *RAID 0***, por lo que **se tiene la velocidad de las 0 pero con los datos duplicados**.
- ***RAID 1+0*** o ***RAID 10***: es la inversa a la anterior. Se tiene que **crear dos matrices *RAID 1***, y **combinarlas para crear entre las dos una matriz *RAID 0***. También requiere de un **mínimo de 4 discos duros**.
- ***RAID 5+0*** o ***RAID 50***: se necesitan **al menos seis discos duros**, con los que se creará un **mínimo de dos matrices *RAID 5***. Estas matrices, a su vez, **se conectarán entre ellas formando una *RAID 0***.





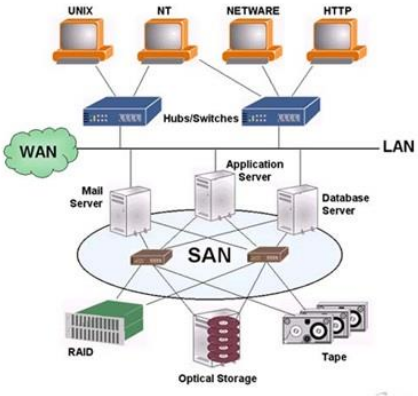
3.4.4 Almacenamiento en red.

Las redes han evolucionado con el paso del tiempo y en la actualidad una red empresarial tiene una capacidad de transferencia de al menos 1000 *Mbps* (en caso de red de fibra óptica, de 10 *Gbps*), lo que **permite transferir mucha información** en poco tiempo. Esta velocidad de transferencia ha hecho populares **métodos de almacenamiento en red** como *NAS* o *SAN* (destinados sobre todo a almacenamiento empresarial o personal):

- ***NAS*** (*Network Attached Storage*): es un **sistema de almacenamiento de datos que utiliza un único dispositivo accesible desde la red para guardar y compartir información**. Un servidor *NAS* **cuenta con su propio hardware y *SO* que gestiona el acceso de usuarios y las operaciones con datos que pueden realizar en el mismo**.
- ***SAN*** (*Storage Area Network*): su uso principal es **en servidores de aplicaciones**. Es una **solución de almacenamiento en la red donde múltiples dispositivos actúan como bloques de disco, permitiendo el almacenamiento y acceso a la información desde cualquier punto de la red**.

  La arquitectura de este tipo de sistemas permite que los recursos de almacenamiento estén disponibles para varios servidores en una red de área local (*LAN*) o amplia (*WAN*).

  Debido a que la **información almacenada no reside directamente en ninguno de los servidores**, se optimiza el poder de procesamiento para aplicaciones comerciales y la capacidad de almacenamiento se puede proporcionar en el servidor donde más se necesite.



3.4.5 Almacenamiento en la nube.

Hoy en día los sistemas de almacenamiento de datos tienden a migrar al almacenamiento *cloud*. Se trata de contratar un **espacio externo** (en los servidores de un proveedor de almacenamiento en la nube) **donde es posible almacenar toda la información y acceder a ella** independientemente del *SO*, ubicación, hora o dispositivo que se utilice.

Según ha aumentado la tasa de transmisión de datos a través de *Internet*, ha aumentado también la facilidad con la que es posible conectarse a los servicios de almacenamiento en la nube.

Ahora mismo es posible contratar suscripciones a servicios en la nube que ofrecen grandes capacidades de almacenamiento de datos a costes muy reducidos para las empresas.

Las **ventajas** del almacenamiento en la nube en cuanto a **seguridad, portabilidad y accesibilidad** lo han convertido en el sistema de almacenamiento de datos más utilizado por las empresas en la actualidad.

El almacenamiento en la nube **implica disponer de una constante conexión a *Internet*** por lo que es recomendable utilizar distintos tipos de almacenamiento para garantizar el acceso a la información en todo momento.

Autoevaluación

Rellena los huecos con los conceptos adecuados.

Un tipo de red donde se optimiza el poder de procesamiento para aplicaciones comerciales, pudiendo proporcionarse la capacidad de almacenamiento en el servidor donde más se necesite, se denomina sistema _____.

TAREA

4. **Citar las ventajas de las *BD* sobre los sistemas de gestión de datos basados en ficheros.**
5. **En función de la ubicación de la información, ¿qué tipos de almacenamiento se pueden tener? ¿qué ventajas e inconvenientes presenta cada uno de ellos?**

4 Evolución de los modelos de *BD*.

Las diferencias entre los modelos de *BD* más habituales es resultado de la evolución técnica de la transmisión electrónica de datos, que no solo perseguía la eficiencia y la manejabilidad, sino también, el empoderamiento de los fabricantes más renombrados.

***Modelar**: definir un mundo abstracto y teórico tal que las conclusiones que se pueden sacar de él coinciden con las manifestaciones aparentes del mundo real.*

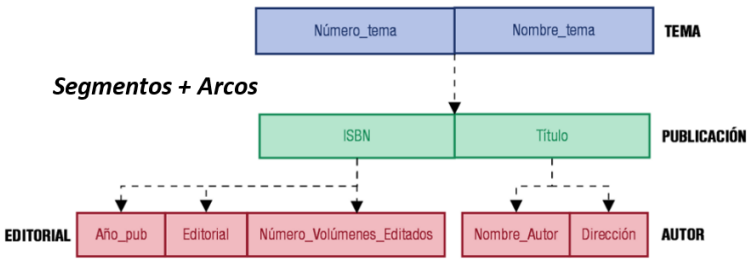
***Modelo**: instrumento que se aplica a una parcela del mundo real para obtener una estructura de datos a la que denominamos esquema. El modelo será un conjunto de conceptos, reglas y convenciones que nos permiten describir los datos del mundo real.*

***Modelo de datos**: conjunto de conceptos o herramientas conceptuales que sirven para describir la estructura de una BD (los datos, las relaciones y las restricciones que se deben cumplir sobre los datos).*

***Esquema de la BD**: descripción de una BD mediante un modelo de datos.*

4.1 Modelo jerárquico.

Cuando ***IBM* creó** su *Sistema Administrador de Información* o ***IMS***, se establecieron las bases para que la gran mayoría de sistemas de gestión de información de los años sesenta utilizaran el modelo jerárquico. También recibe el nombre de **modelo en árbol**, ya que **utiliza una estructura en árbol invertido para la organización de los datos**.

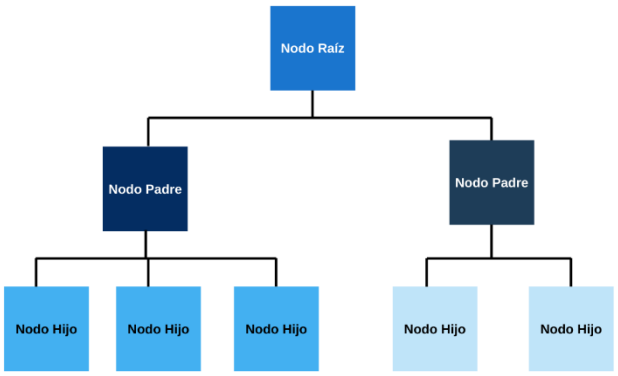


La información se organiza con una jerarquía en la que la relación entre las entidades de este modelo siempre es del tipo padre/hijo. De tal manera que existen nodos que contienen atributos o campos y que se relacionarán con sus nodos hijos, pudiendo tener cada nodo más de un hijo, pero un nodo siempre tendrá un sólo padre.

Los datos de este modelo se almacenan en estructuras lógicas llamadas segmentos. Los segmentos se relacionan entre sí utilizando arcos. La forma visual de este modelo es de árbol invertido, en la parte superior están los padres y en la inferior los hijos.

Las **principales características** de las *BD* jerárquicas son las siguientes:

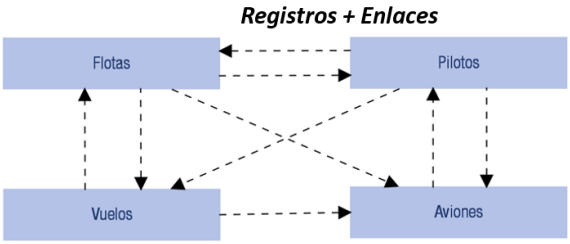
- ✓ Se **organizan** en forma de **árbol invertido**, con un nodo raíz, nodos padre e hijos.
- ✓ El árbol se organiza en un conjunto de **niveles**.
- ✓ El **nivel 0** se corresponde al **nodo raíz** y es el nivel más alto de la jerarquía.
- ✓ Los **arcos** (enlaces) **representan las asociaciones jerárquicas entre dos nodos**, carecen de nombre porque entre dos conjuntos de datos solo puede existir una interrelación.



- ✓ Un **nodo padre puede tener un número ilimitado de nodos hijos**, pero **a un nodo hijo solo le puede corresponder un padre**.
- ✓ **Todo nodo debe tener un padre**, a **excepción del nodo raíz**.
- ✓ Los **nodos sin descendientes** se llaman **hojas**.
- ✓ Los **niveles de la estructura jerárquica** se denominan **altura**.
- ✓ El **número de nodos** se llama **momento**.
- ✓ El **árbol** siempre **se recorre en un orden prefijado**.
- ✓ Solo pueden existir **relaciones** de **uno a uno** o de **uno a varios**.
- ✓ La **estructura del árbol no se puede modificar cuando ha quedado establecida**.

Hoy en día, debido a sus limitaciones, **el modelo jerárquico está en desuso**.

4.2 Modelo en red.



El modelo de datos en red **aparece** a mediados de los sesenta **como respuesta a las limitaciones del modelo jerárquico en cuanto a representación de relaciones más complejas**. Se puede considerar a ***IDS*** (*Integrated Data Store*) **de *Bachman*** como el primer sistema de *BD* en red. Tras él **se intentó crear un estándar de modelo de red por parte de *CODASYL***, siendo un modelo que tuvo gran aceptación a principios de los setenta.

El modelo en red **organiza la información en registros** (también llamados nodos) **y enlaces**. **En los registros se almacenan los datos, mientras que los enlaces permiten relacionar estos datos**. Las *BD* en red son parecidas a las jerárquicas sólo que en ellas puede haber más de un padre.

En este modelo **se puede representar perfectamente cualquier tipo de relación entre los datos, pero hace muy complicado su manejo**. Al no tener que duplicar la información se ahorra espacio de almacenamiento.

El sistema de gestión de información basado en el modelo en red más popular es el sistema *IDMS*.

Entre las principales **ventajas** de las *BD* en red está la **posibilidad de establecer relaciones de muchos a muchos**. Por ejemplo, una tienda online que quiere relacionar a los productos con los pedidos. Un producto puede ser objeto de múltiples pedidos, pero a su vez, un pedido puede contener diversos productos. Una *BD* de red permite establecer este tipo de relaciones cruzadas, lo cual no es posible con las *BD* jerárquicas estándar.

4.3 Modelo relacional.

Este modelo es posterior a los dos anteriores y fue **desarrollado por *Codd* en 1970**. **Hoy en día las *BD* relacionales son las más utilizadas**.

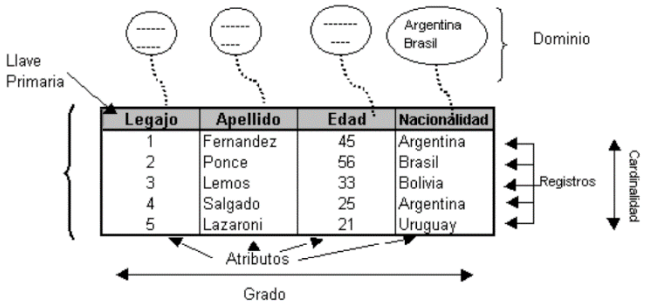
En el modelo relacional **la *BD* es percibida por el usuario como un conjunto de tablas**. Esta percepción es sólo a nivel lógico, ya que a nivel físico puede estar implementada mediante distintas estructuras de almacenamiento.

El modelo relacional **utiliza tablas bidimensionales (relaciones) para la representación lógica de los datos y las relaciones entre ellos**. Cada relación (tabla) posee un nombre que es único y contiene un conjunto de columnas (atributos).

Se llamará **registro, entidad o tupla a cada fila de la tabla** y **campo o atributo a cada columna de la tabla**.

A los **conjuntos de valores que puede tomar un determinado atributo**, se le denomina **dominio**.

Una **clave** (llave primaria) será un **atributo o conjunto de atributos que identifique de forma única a una tupla**.

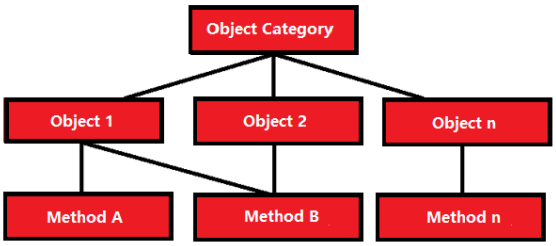


Las tablas deben cumplir una serie de **requisitos**:

- ✓ Todos los **registros son del mismo tipo**.
- ✓ La **tabla sólo puede tener un tipo de registro**.
- ✓ **No existen campos o atributos repetidos**.
- ✓ **No existen registros duplicados**.
- ✓ **No existe orden en el almacenamiento de los registros**.

El **lenguaje habitual para hacer consultas** a *BD* relacionales es ***SQL***, *Structured Query Language* o *Lenguaje Estructurado de Consultas*, un **estándar implementado por los principales motores o *SGBD* relacionales**.

4.4 Modelo orientado a objetos.



El modelo orientado a objetos **define una *BD* en términos de objetos, sus propiedades y sus operaciones**. Los **objetos con la misma estructura y comportamiento pertenecen a una clase, y las clases se organizan en jerarquías**. Las operaciones de cada clase se especifican en términos de procedimientos predefinidos denominados métodos. Algunos **sistemas existentes en el mercado, basados en el modelo relacional, han sufrido evoluciones incorporando conceptos orientados a objetos. A estos modelos se les conoce como sistemas objeto-relacionales**.

El **objetivo** del modelo orientado a objetos es **cubrir las limitaciones del modelo relacional**. Gracias a este modelo **se incorporan mejoras como la herencia entre tablas, los tipos definidos por el usuario, disparadores almacenables en la *BD* (*triggers*), soporte multimedia, etc.**

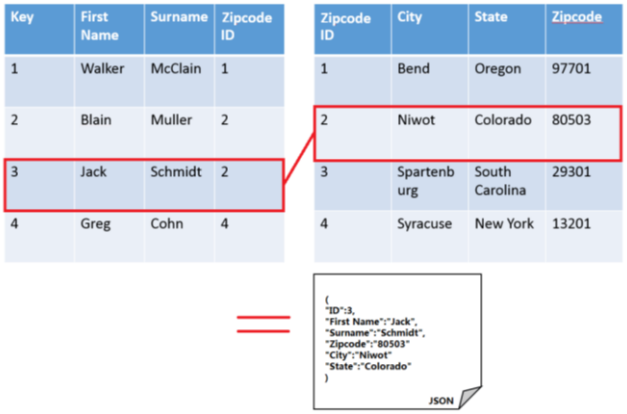
Los **conceptos más importantes del paradigma de objetos** que el modelo orientado a objetos incorpora son:

- ✓ **Encapsulación**: propiedad que permite **ocultar la información al resto de los objetos**, impidiendo así accesos incorrectos o conflictos.
- ✓ **Herencia**: propiedad a través de la cual los **objetos heredan comportamiento** (sus propiedades y sus operaciones) dentro de una jerarquía de clases.
- ✓ **Polimorfismo**: propiedad de **una operación** mediante la cual **puede ser aplicada a distintos tipos de objetos**.

Desde la aparición de la *Programación Orientada a Objetos* (*POO* u *OOP*) se empezó a pensar en *BD* adaptadas a estos lenguajes. Este modelo es considerado como el fundamento de las *BD* de tercera generación, siendo consideradas las *BD* en red como la primera y las *BD* relacionales como la segunda generación, aunque **no han reemplazado a las *BD* relacionales**.

4.5 Modelo orientado a documentos.

En este modelo, **los documentos son la unidad básica para el almacenamiento de datos**. **Estas unidades son las que estructuran los datos** y no deben confundirse con los documentos de los programas de procesamiento de texto. Aquí, **los datos se guardan en los llamados pares clave-valor**, comprendiendo así, una *clave* y un *valor*. Como **no están definidos ni la estructura ni el número de pares, los documentos que integran una *BD* orientada a documentos pueden resultar muy dispares entre sí**. Cada **documento es una unidad cerrada en sí misma y establecer relaciones entre documentos no resulta fácil**, pero en este modelo no es necesario.



En el modelo relacional, varias relaciones (tablas) se conectan entre sí para seleccionar un registro común. En el modelo documental, un único documento basta para guardar toda la información. Aquí no se está obligado a utilizar un determinado esquema porque, mientras se use siempre el mismo lenguaje de *BD*, **este modelo está conceptualmente libre de esquemas**.

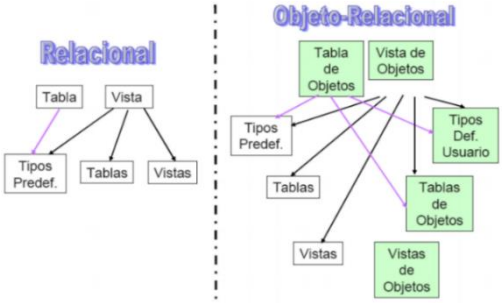
Una idea fundamental de las *BD* documentales es que los datos que guardan relación entre sí siempre se guardan juntos en un lugar (en el documento). Mientras que las *BD* relacionales suelen representar y mostrar la información relacionada conectando varias tablas, en este modelo es suficiente con consultar un solo documento.

Estos sistemas **son especialmente interesantes para las aplicaciones web**, puesto que permiten guardar formularios *HTML* completos. Fue sobre todo con el avance de la *Web 2.0* cuando estas *BD* vieron aumentar su popularidad. Con todo, es necesario remarcar que **entre los diversos sistemas basados en documentos se dan diferencias notables, desde la sintaxis hasta la estructura interna, por lo que no todas las *BD* orientadas a documentos son apropiadas para cualquier escenario**. Es debido a estas diferencias por lo que hoy se dispone de algunos sistemas de *BD* orientados a documentos de la reputación de *Lotus Notes*, *Amazon SimpleDB*, ***MongoDB***, *CouchDB*, *Riak*, *ThruDB*, *OrientDB*, etc.

4.6 Otros modelos.

Además de los modelos vistos, se van a detallar a continuación las particularidades de otros modelos de *BD* existentes y que, en algunos casos, son una evolución de los anteriores.

4.6.1 Modelo objeto-relacional.



Las *BD* pertenecientes a este modelo, **son un híbrido entre las *BD* del modelo relacional y el orientado a objetos**. El mayor inconveniente de las *BD* orientadas a objetos radica en los costes de la conversión de las *BD* relacionales a *BD* orientadas a objetos.

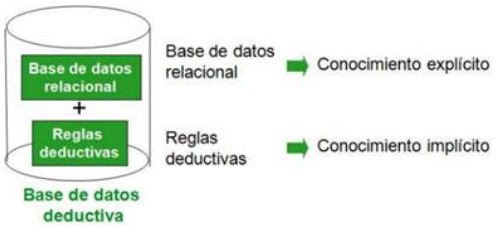
En una *BD* objeto-relacional (*BDOR*) siempre **se busca obtener lo mejor del modelo relacional, incorporando las mejoras ofrecidas por la orientación a objetos**. En este modelo se siguen almacenando tuplas, aunque la estructura de las tuplas no está restringida, sino que las relaciones pueden ser definidas en función de otras, que es lo que denominamos herencia directa.

El **estándar en el que se basa este modelo es *SQL99***. Este estándar **ofrece la posibilidad de añadir a las *BD* relacionales procedimientos almacenados de usuario, triggers, tipos definidos por el usuario, consultas recursivas, bases de datos *OLAP*, tipos *LOB*** (*BLOB* y *CLOB*)**, ...**

Otra característica a destacar es la capacidad para incorporar funciones que tengan un código en algún lenguaje de programación como, por ejemplo: *SQL*, *Java*, *C*, etc.

La **gran mayoría de las *BD* relacionales** clásicas de gran tamaño, como ***Oracle***, ***PosgreSQL***, *SQL Server*, etc., **son objeto-relacionales**.

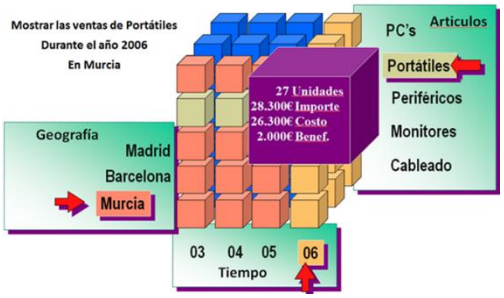
4.6.2 Modelo de *BD* deductivas o lógicas.



En este modelo **las *BD* almacenan la información y permiten realizar deducciones a través de inferencias**. Es decir, se derivan nuevas informaciones a partir de las que se han introducido explícitamente en la *BD* por parte del usuario.

Las *BD* deductivas **son también llamadas *BD* lógicas**, al basarse en lógica matemática. **Surgieron para contrarrestar las limitaciones del modelo relacional para la respuesta a consultas recursivas y la deducción de relaciones indirectas entre los datos almacenados**.

4.6.3 *BD* multidimensionales.



Son ***BD* ideadas para desarrollar aplicaciones muy concretas**. Son *BD* de estructura basada en dimensiones orientada a consultas complejas y alto rendimiento. En una *BD* multidimensional, **la información se representa como matrices multidimensionales, cuadros de múltiples entradas o funciones de varias variables sobre conjuntos finitos**. Cada una de estas matrices se denomina cubo.

4.6.4 *BD* transaccionales.

Son ***BD* caracterizadas por su velocidad para gestionar el intercambio de información, se utilizan sobre todo en sistemas bancarios, análisis de calidad y datos de producción industrial**. Son *BD* **muy fiables**, ya que en ellas cada una de las operaciones de inserción, actualización o borrado se realizan completamente o se descartan.

4.7 Tabla comparativa de los principales modelos.

Modelo de *BD*	Desarrollo	Ventajas	Inconvenientes	Ámbitos de aplicación	Marcas
Jerárquico (segmentos y arcos)	Década de 1960	Acceso de lectura muy rápido, estructura clara, técnicamente simple	Estructura fija en árbol que no permite conexiones entre árboles	Bancos, compañías de seguros, sistemas operativos	*IMS/DB*
En red (registros y enlaces)	Mediados década de 1960	Admite varias formas de acceder a un registro, sin jerarquía estricta	En *BD* más grandes no se tiene una vista general	Grandes ordenadores	*UDS* (Siemens), *DMS* (Sperry Univac)
Relacional (tablas)	1970	Simple, flexible. Creación y edición fácil y flexible, fácil de ampliar, rápida puesta en marcha, contexto de competencia muy dinámica	Inmanejable con cantidades grandes de datos, segmentación deficiente, atributos de clave artificiales, interfaz de programación externa, no refleja bien las propiedades y la conducta de los objetos	Control de gestión, facturación, sistemas de control de inventario, sistemas de gestión de contenido, etc.	*MySQL*, *PostgreSQL*, *Oracle*, *SQLite*, *DB2*, *Ingres*, *MariaDB*, *Microsoft Access*
Orientado a objetos (objetos)	Final de la década de 1980	Soporta mejor los lenguajes de programación orientados a objetos, permite almacenar contenido multimedia	El rendimiento empeora con grandes volúmenes de datos, pocas interfaces compatibles	Inventario (museos, comercio minorista)	*db4o*
Orientado a documentos (documentos sin esquema definido)	Década de 1980	Los datos relacionados se guardan de forma centralizada en documentos independientes, estructura libre, concepción multimedia	El trabajo de organización es relativamente alto, a menudo requiere conocimientos de programación	Aplicaciones web, buscadores, bases de datos de texto	*Lotus Notes*, *Amazon SimpleDB*, *MongoDB*, *CouchDB*, *Riak*, *ThruDB*, *OrientDB*

TAREA

6. **Teniendo en cuenta los distintos modelos de *BD*, ¿qué tipo de *BD* es el más utilizado en la actualidad? ¿qué tipos de *BD* no se utilizan actualmente? ¿qué tipo de *BD noSQL* están apareciendo en la actualidad y en qué tipo de aplicaciones se suelen utilizar?**
7. **Visitar la *URL* https://db-engines.com/en/ranking e indicar el modelo de *BD* más utilizado en las primeras 20 posiciones. ¿Qué otros modelos de *BD* aparecen en las primeras 20 posiciones? Investigar y describir los modelos que aparecen en las primeras 20 posiciones.**

5 Sistemas Gestores de Bases de Datos (*SGBD*).

Para poder tratar la información contenida en las *BD* se utilizan los *Sistemas Gestores de Bases de Datos* o *SGBD*, también llamados *DBMS* (*DataBase Management System*), que **ofrecen un conjunto de programas que permiten acceder y gestionar** dichos **datos**.

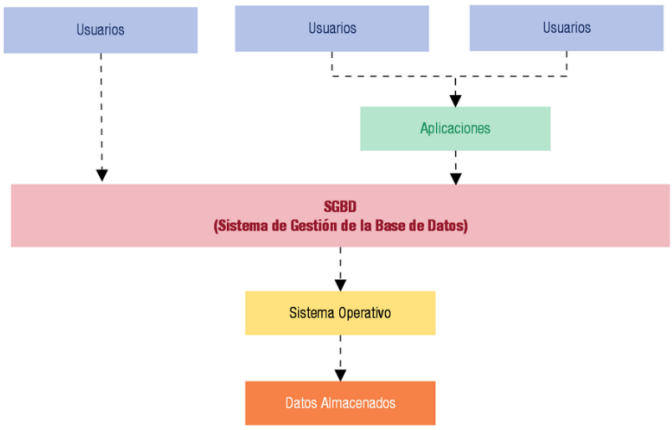
El **objetivo fundamental** de los *SGBD* es **proporcionar eficiencia y seguridad a la hora de recuperar o insertar información en las *BD***. Estos sistemas están diseñados para la manipulación de grandes bloques de información.

***Sistema Gestor de Base de Datos**: conjunto coordinado de programas, procedimientos, lenguajes, etc., que suministra, tanto a los usuarios no informáticos, como a los analistas programadores, o al administrador, los medios necesarios para describir y manipular los datos contenidos en la BD, manteniendo su integridad, confidencialidad y seguridad.*

El *SGBD* **permite a los usuarios la creación y el mantenimiento de las *BD*, facilitando la definición, construcción y manipulación de la información contenida en éstas**. Definir una *BD* consistirá en especificar los tipos de datos, las estructuras y las restricciones que los datos han de cumplir a la hora de almacenarse en dichas *BD*. Por otro lado, la construcción de una *BD* será el proceso de almacenamiento de datos concretos en algún medio o soporte de almacenamiento que esté supervisado por el *SGBD*. Finalmente, la manipulación de la *BD* incluirá la posibilidad de realización de consultas para recuperar información específica, la actualización de los datos (inserción, borrado o modificación) y la generación de informes a partir de su contenido.

Las **ventajas del uso de *SGBD*** son:

- ✓ **Proporcionan al usuario una visión abstracta de los datos, ocultando parte de la complejidad** relacionada con cómo se almacenan y mantienen los datos.
- ✓ **Ofrecen independencia física**, es decir, la visión que tiene de la información el usuario, y la manipulación de los datos almacenados en la *BD*, es independiente de cómo estén almacenados físicamente.
- ✓ **Ofrecen independencia lógica**.
- ✓ **Disminuyen la redundancia y la inconsistencia de los datos**.
- ✓ **Aseguran la integridad de los datos**.
- ✓ **Facilitan el acceso a los datos**, aportando rapidez y evitando la pérdida de datos.
- ✓ **Aumentan la seguridad y privacidad de los datos**.
- ✓ **Mejoran la eficiencia**.
- ✓ **Permiten compartir datos y accesos concurrentes**.
- ✓ **Facilitan el intercambio de datos entre distintos sistemas**.
- ✓ **Incorporan mecanismos de copias de seguridad y recuperación** para restablecer la información en caso de fallos en el sistema.



El *SGBD* interacciona con otros elementos software existentes en el sistema, concretamente con el *SO*. Los datos almacenados de forma estructurada en la *BD* son utilizados indistintamente por otras aplicaciones, será el *SGBD* quien ofrecerá una serie de facilidades a éstas para el acceso y manipulación de la información, basándose en las funciones y métodos propios del *SO*.

5.1 Funciones.

El *SGBD* es una aplicación (conjunto de programas) **que permite a los usuarios definir, crear y mantener la *BD* y proporciona un acceso controlado a la misma**. Debe prestar los siguientes **servicios**:

- ✓ **Creación y definición de la *BD***: especificación de la estructura, el tipo de los datos, las restricciones y relaciones entre ellos mediante lenguajes de definición de datos (toda esta información se almacena en el diccionario de datos).
- ✓ **Manipulación de los datos**: realizando consultas, inserciones y actualizaciones utilizando lenguajes de manipulación de datos.
- ✓ **Acceso controlado a los datos de la *BD***: mediante mecanismos de seguridad de acceso de usuarios.
- ✓ **Mantener la integridad y consistencia de los datos**: utilizando mecanismos para evitar que los datos sean perjudicados por cambios no autorizados.
- ✓ **Acceso compartido a la *BD***: controlando la interacción entre usuarios concurrentes.
- ✓ **Mecanismos de respaldo y recuperación**: para restablecer la información en caso de fallos del sistema.

Un *SGBD* desarrolla **tres funciones fundamentales**:

1. **Función de descripción o definición**: **permite al diseñador de la *BD* crear las estructuras apropiadas para integrar adecuadamente los datos**. Esta función es la que **permite definir las tres estructuras de la *BD*: estructura interna, estructura conceptual y estructura externa**.

   Esta función **se realiza mediante el lenguaje de descripción de datos** o ***DDL***. Mediante este lenguaje se definen las estructuras de datos, las relaciones entre los datos y las reglas (restricciones) que han de cumplir los datos.

Se especificarán las características de los datos a cada uno de los tres niveles:

- **A nivel interno** (estructura interna): se ha de **indicar el espacio de disco reservado para la *BD*, la longitud de los campos, su modo de representación**, …
- **A nivel conceptual** (estructura conceptual): se proporcionan herramientas para la **definición de las entidades y su identificación, atributos de las mismas, interrelaciones entre ellas, restricciones de integridad, …**
- **A nivel externo** (estructura externa): se deben **definir las vistas de los distintos usuarios** (vistas parciales que cada usuario tiene de los datos).

Además, el *SGBD* se ocupará de la transformación de las estructuras externas orientadas a los usuarios a las estructuras conceptuales y de la relación de ésta con la estructura interna.

2. **Función de manipulación**: **permite a los usuarios de la *BD* buscar, añadir, suprimir o modificar los datos de la misma, siempre de acuerdo con las especificaciones y las normas de seguridad dictadas por el administrador**. Se llevará a cabo **por medio de un lenguaje de manipulación de datos** (***DML***) que proporciona los instrumentos necesarios para la realización de estas tareas. Por **manipulación de datos** se entiende:
   - La recuperación de información almacenada en la *BD*, lo que se conoce como **consultas**.
   - La **inserción** de información nueva en la *BD*.
   - El **borrado** de información de la *BD*.
   - La **modificación** de información almacenada en la *BD*.
3. **Función de control**: **permite al administrador de la *BD* establecer mecanismos de protección de las diferentes visiones de los datos asociadas a cada usuario, proporcionando elementos de creación y modificación de dichos usuarios**. Adicionalmente, **incorpora sistemas para la creación de copias de seguridad, carga de ficheros, auditoría, configuración del sistema, etc.** El lenguaje que implementa esta función es el **lenguaje de control de datos** o ***DCL***.

¿Y a través de qué lenguaje se podrán desarrollar estas funciones? Se hará utilizando el ***Lenguaje Estructurado de Consultas*** (***SQL***: *Structured Query Language*). Este lenguaje **proporciona sentencias para realizar operaciones de *DDL*, *DML* y *DCL***. *SQL* fue publicado por el *ANSI* (*American National Standard Institute*) en 1986 y ha ido evolucionando a lo largo del tiempo. Además, los *SGBD* suelen proporcionar otras herramientas que complementan a estos lenguajes como generadores de formularios, informes, interfaces gráficas, generadores de aplicaciones, etc.

Autoevaluación

El *DDL* de una *BD* sirve para:

- La introducción de los datos en una *BD*.
- Definir la estructura lógica de la *BD*.
- Interrogar a la *BD* (consultar la información de dicha *BD*).

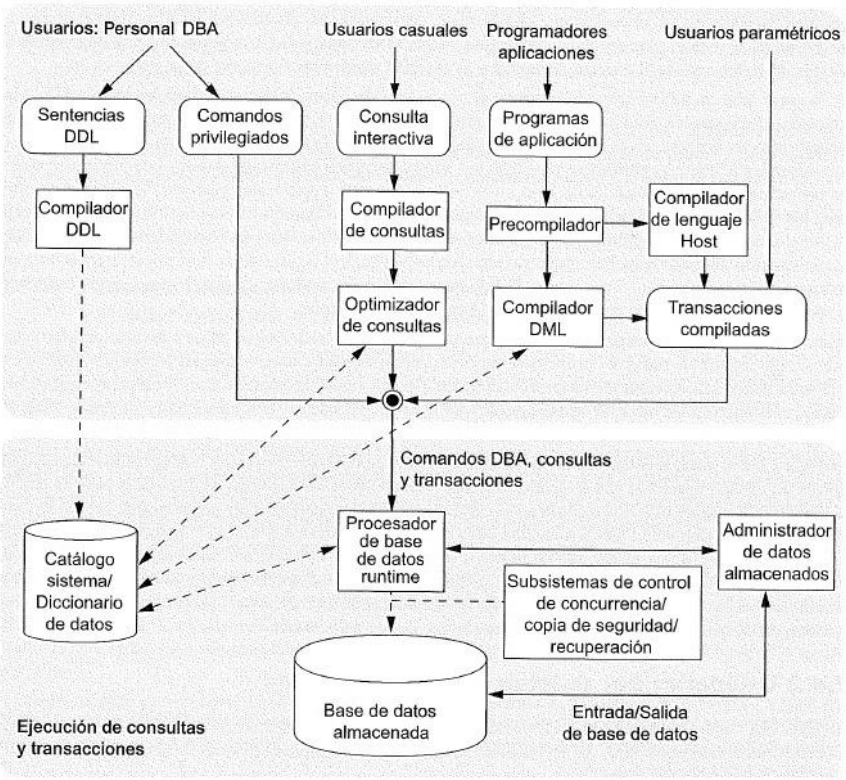
5.2 Componentes.

Una vez descritas las funciones que un *SGBD* debe llevar a cabo, es fácil imaginar que **un *SGBD* es un paquete de software complejo que ha de proporcionar servicios relacionados con el almacenamiento y la explotación de los datos** de forma eficiente. Para ello, cuenta con una serie de **componentes** que se detallan a continuación:

1. **Lenguajes de la *BD***: cualquier *SGBD* ofrece la posibilidad de utilizar lenguajes e interfaces adecuadas para sus diferentes tipos de usuarios. A través de los lenguajes se pueden especificar los datos que componen la *BD*, su estructura, relaciones, reglas de integridad, control de acceso, características físicas y vistas externas de los usuarios. Los lenguajes del *SGBD* son: ***Lenguaje de Definición de los Datos* (*DDL*), *Lenguaje de Manejo de Datos* (*DML*) y *Lenguaje de Control de Datos* (*DCL*)**.
2. **El diccionario de datos**: **descripción de los datos almacenados**. Se trata de información útil para los programadores de aplicaciones. **Es el lugar donde se deposita la información sobre la totalidad de los datos que forman la *BD*. Contiene las características lógicas de las estructuras que almacenan los datos, su nombre, descripción, contenido y organización**. En una *BD* relacional, el diccionario de datos aportará información sobre:
   - ✓ Estructura lógica y física de la *BD*.

- ✓ Definición de tablas, vistas, índices, disparadores, procedimientos, funciones, etc.
- ✓ Cantidad de espacio asignado y utilizado por los elementos de la *BD*.
- ✓ Descripción de las restricciones de integridad.
- ✓ Información sobre los permisos asociados a cada perfil de usuario.
- ✓ Auditoría de acceso a los datos, utilización, etc.

3. **El gestor de *BD***: es la **parte de software encargada de garantizar el correcto, seguro, íntegro y eficiente acceso y almacenamiento de los datos**. Este componente es el encargado de proporcionar una interfaz entre los datos almacenados y los programas de aplicación que los manejan (intermediario entre el usuario y los datos). Es el encargado de garantizar la privacidad, seguridad e integridad de los datos, controlando los accesos concurrentes e interactuando con el *SO*.
4. **Usuarios de la *BD***: en los *SGBD* existen **diferentes perfiles de usuario**, cada uno de ellos con una serie de permisos sobre los objetos de la *BD*. Generalmente existirán:
   - ✓ El **administrador de la *BD*** o *Database Administrator* (***DBA***), que será la **persona o conjunto de ellas encargadas de la función de administración de la *BD***. Tiene el control centralizado de la *BD* y es el responsable de su buen funcionamiento. Es el encargado de autorizar el acceso a la *BD*, de coordinar y vigilar su utilización y de adquirir los recursos software y hardware que sean necesarios.
   - ✓ Los **usuarios de la *BD***, que serán **diferentes usuarios de la *BD* con diferentes necesidades** sobre los datos, así como diferentes accesos y privilegios. Se puede establecer la siguiente clasificación:
     - Operadores y personal de mantenimiento.
     - Analistas y programadores de aplicaciones.
     - Usuarios finales: ocasionales, simples, avanzados y autónomos.
     - Diseñadores.



5. **Herramientas de la *BD***: son un **conjunto de aplicaciones que permiten a los administradores la gestión de la *BD*, de los usuarios y permisos, generadores de formularios, informes, interfaces gráficas, generadores de aplicaciones, etc.**

5.3 Arquitectura.

Un *SGBD* cuenta con una arquitectura a través de la que se simplifica su trabajo a los diferentes usuarios de la *BD*. El **objetivo fundamental es separar los programas de aplicación de la *BD* física**.

Encontrar un estándar para esta arquitectura no es una tarea sencilla, aunque los tres estándares que más importancia han cobrado en el campo de las *BD* son *ANSI/SPARC/X3*, *CODASYL* (para *BD* en red) y *ODMG* (para las *BD* orientadas a objetos). Tanto *ANSI* (*EEUU*), como *ISO* (resto del mundo), son el referente en cuanto a estandarización de *BD*, conformando un único modelo de *BD*.

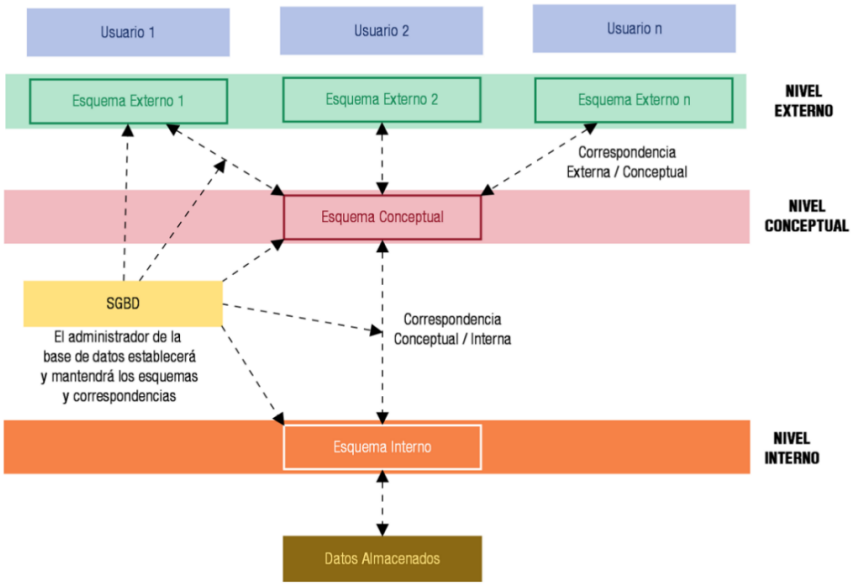
La **arquitectura propuesta por *ANSI-SPARC***** proporciona **tres niveles de abstracción**:

- ✓ **Nivel interno o físico** (perspectiva del almacenamiento físico): en este nivel se **describe la estructura física de la *BD* a través de un esquema interno encargado de detallar el sistema de almacenamiento de la *BD* y sus métodos de acceso**. Es el nivel **más cercano al almacenamiento físico**. A través del esquema físico **se indican**, entre otros, los **ficheros que contienen la información**, su **organización**, los **métodos de acceso** a

los registros, los **tipos de registros**, la **longitud**, los **campos** que los componen, las **unidades de almacenamiento**, etc.

- ✓ **Nivel lógico o conceptual** (perspectiva del administrador): en este nivel se **describe la estructura completa de la *BD* a través de un esquema conceptual que detalla las entidades, atributos, relaciones, operaciones de los usuarios y restricciones**. Los detalles relacionados con las estructuras de almacenamiento se ocultan, permitiendo realizar una abstracción a más alto nivel.
- ✓ **Nivel externo o de visión del usuario** (perspectiva del programador/usuario): en este nivel **se describen las diferentes vistas que los usuarios percibirán de la *BD***. Cada tipo de usuario o grupo de ellos verá sólo la parte de la *BD* que le interesa, ocultando el resto.

Para una *BD*, sólo existirá un único esquema interno, un único esquema conceptual y podrían existir varios esquemas externos definidos para uno o varios usuarios.



Los *SGBD* basados en esta arquitectura permiten que cada grupo de usuarios haga referencia a su propio esquema externo. El *SGBD* debe de transformar cualquier petición de usuario (esquema externo) a una petición expresada en términos del esquema conceptual para finalmente ser una petición expresada en términos del esquema interno que se procesará sobre la *BD* almacenada. El **proceso de transformar peticiones y resultados de un nivel a otro se denomina correspondencia o transformación**, el *SGBD* es capaz de interpretar una solicitud de datos y realiza los siguientes pasos:

- El usuario solicita unos datos y crea una consulta.
- El *SGBD* verifica y acepta el esquema externo para ese usuario.
- Transforma la solicitud al esquema conceptual.
- Verifica y acepta el esquema conceptual.
- Transforma la solicitud al esquema físico o interno.
- Selecciona la o las tablas implicadas en la consulta y ejecuta la consulta.
- Transforma del esquema interno al conceptual, y del conceptual al externo.
- Finalmente, el usuario ve los datos solicitados.

Gracias a esta arquitectura se consigue la independencia de datos a dos niveles:

- ✓ **Independencia lógica**: se puede **modificar el esquema conceptual sin alterar los esquemas externos ni los programas de aplicación**.
- ✓ **Independencia física**: se puede **modificar el esquema interno sin necesidad de modificar el conceptual o el externo**. Es decir, **se puede cambiar el sistema de almacenamiento, reorganizar los ficheros, añadir nuevos, etc., sin que esto afecte al resto de esquemas**.

Autoevaluación

El esquema conceptual de la totalidad de la *BD* puede obtenerse de la unión de todos los esquemas externos definidos para cada usuario de la *BD*. ¿Verdadero o falso? Razona la respuesta.

5.4 Tipos.

¿Qué tipos de *SGBD* existen? Para responder a esta pregunta se puede realizar la siguiente **clasificación, atendiendo a diferentes criterios**:

a) El primer criterio que se suele utilizar es por el **modelo lógico en que se basan**. Actualmente, el modelo lógico que más se utiliza es el relacional. Los modelos en red y jerárquico han quedado obsoletos. Otro de los

modelos que más se está utilizando actualmente es el modelo orientado a documentos.

Ver punto 4 de la unidad para tener acceso a la clasificación.

b) El segundo criterio de clasificación se centra en el **número de usuarios a los que da servicio el sistema**:
   - ✓ **Monousuario**: sólo **atienden a un usuario a la vez**, y su principal uso se da en los ordenadores personales.
   - ✓ **Multiusuario**: entre los que se encuentran la mayor parte de los *SGBD*, **atienden a múltiples usuarios al mismo tiempo**.

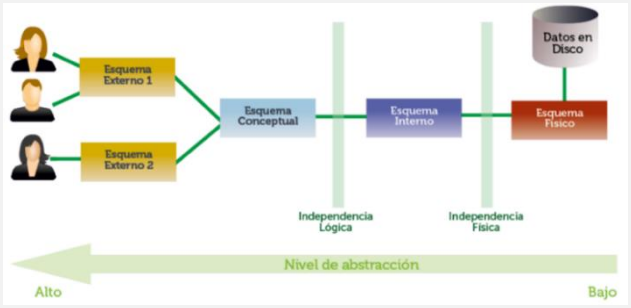
c) El tercer criterio se basa en el **número de sitios en los que está distribuida la *BD***:
   - ✓ **Centralizados**: **sus datos se almacenan en un solo ordenador**. Los *SGBD* centralizados pueden atender a varios usuarios, pero el *SGBD* y la *BD* en sí residen por completo en una sola máquina.
   - ✓ **Distribuidos** (homogéneos o heterogéneos): **la *BD* real y el propio software del *SGBD* pueden estar distribuidos en varios sitios conectados por una red**. Los **sistemas homogéneos** utilizan el **mismo *SGBD* en múltiples sitios**. Una tendencia reciente consiste en crear software para tener acceso a varias *BD* autónomas preexistentes almacenadas en **sistemas distribuidos heterogéneos**. Esto da lugar a los *SGBD* federados o **sistemas multibase de datos en los que los *SGBD* participantes tienen cierto grado de autonomía local**.

d) El cuarto criterio toma como referencia el **coste**. La mayor parte de los paquetes cuestan entre 10.000 y 100.000 €. Los sistemas monousuario más económicos para microordenadores cuestan entre 0 y 3.000 €. En el otro extremo, los paquetes más completos cuestan más de 100.000 €.

e) El quinto, y último, criterio establece su clasificación **según el propósito**:
   - ✓ **Propósito general**: pueden ser utilizados **para el tratamiento de cualquier tipo de *BD* y aplicación**.
   - ✓ **Propósito específico**: cuando el rendimiento es fundamental, se puede diseñar y construir un **software de propósito especial para una aplicación específica**, y este sistema **no sirve para otras aplicaciones**. Muchos sistemas de reservas de líneas aéreas son de propósito especial y pertenecen a la categoría de sistemas de procesamiento de transacciones en línea, que deben atender un gran número de transacciones concurrentes sin imponer excesivos retrasos.

TAREA

8. **¿Qué funciones realiza un *SGBD*? Describir cada una de ellas.**
9. **¿Cuáles son los componentes de un *SGBD*? ¿qué función realiza cada uno de ellos? Describir y comentar la imagen del punto 5.2.**
10. **¿Cuántos modelos conceptuales se pueden tener de una *BD*? ¿y externos? ¿qué diferencia hay entre ellos?**
11. **Dada la imagen siguiente, describir los pasos que sigue el *SGBD* para interpretar una solicitud de datos de un usuario determinado.**



6 *SGBD* comerciales y libres.

Actualmente, en el mercado de software existen multitud de *SGBD* comerciales. En este punto se desglosan las características fundamentales de los más importantes y extendidos hasta la fecha. Pero, como se puede observar, **la elección de un *SGBD* es una decisión muy importante a la hora de desarrollar proyectos**. A veces, el sistema más avanzado, "*el mejor*" según los entendidos, puede no serlo para el tipo de proyecto que se esté desarrollando. **Se ha de tener en cuenta qué volumen de carga debe soportar la *BD*, qué *SO* se utilizará como soporte, cuál es el presupuesto, plazos de entrega, etc.**

En la siguiente tabla se exponen los ***SGBD* comerciales más utilizados y sus características más relevantes**:

SGBD	Descripción
Oracle	Reconocido como **uno de los mejores a nivel mundial**. Es **multiplataforma, confiable y seguro**. Es Cliente/Servidor. Basado en el modelo de datos relacional. De gran potencia, aunque **con un precio elevado hace que sólo se vea en empresas muy grandes y multinacionales**. Ofrece una versión gratuita *Oracle Database Express Edition*.
MySQL Standard / Enterprise / Clúster	Sistema muy extendido que **se ofrece bajo dos tipos de licencia, comercial o libre**. Para aquellas **empresas que deseen incorporarlo en productos privativos, deben comprar una licencia específica**. Es **relacional, multihilo, multiusuario y multiplataforma**. Su **gran velocidad** lo hace ideal para consulta de *BD* y plataformas web.
DB2	Multiplataforma, el motor de *BD* relacional **integra *XML* de manera nativa**, lo que *IBM* ha llamado *pureXML*, que **permite almacenar documentos completos para realizar operaciones y búsquedas de manera jerárquica dentro de éste, e integrarlo con búsquedas relacionales**.
Informix	Otra opción de *IBM* para el mundo empresarial para quien necesita un *SGBD* **sencillo y confiable**. Es un gestor de *BD* relacional basado en *SQL*. **Multiplataforma**. Consume menos recursos que *Oracle*, con **utilidades muy avanzadas** respecto a conectividad y funciones relacionadas con tecnologías de *Internet*/*Intranet*, *XML*, etc.
Microsoft SQL SERVER	*SGBD* **producido por *Microsoft***. Es relacional, sólo **funciona bajo *Windows***, utiliza arquitectura *Cliente/Servidor*. Constituye la **alternativa a otros potentes *SGBD*** como son *Oracle*, *PostgreSQL* o *MySQL*.
SyBase | SAP ASE	Un *SGBD* con bastantes años en el mercado, fue adquirido por *SAP*. Es un sistema relacional, **altamente escalable, de alto rendimiento, con soporte a grandes volúmenes de datos, transacciones y usuarios, y de bajo costo**.

Otros *SGBD* comerciales importantes son: *dBase*, *Access*, *Interbase* y *FoxPro*.

La alternativa a los *SGBD* comerciales se encuentra en los ***SGBD* de código abierto o libres, también llamados *Open Source***. En la siguiente tabla se relacionan los más utilizados actualmente y sus principales características:

SGBD	Descripción
MySQL Community	Es un *SGBD* **relacional, multihilo y multiusuario** con varios millones de instalaciones. **Distribuido bajo dos tipos de licencias, comercial y libre**. Multiplataforma, posee varios motores de almacenamiento, **accesible a través de múltiples lenguajes de programación y muy ligado a aplicaciones web**.
MariaDB	Es un *SGBD* **derivado de *MySQL* con licencia *GPL***. Es desarrollado por *Michael (Monty) Widenius* -fundador de *MySQL*-, la fundación *MariaDB* y la comunidad de desarrolladores de software libre. Tiene una **alta compatibilidad con *MySQL*** ya que posee las mismas órdenes, interfaces, *API* y bibliotecas, siendo **su objetivo poder cambiar un servidor por otro directamente**.
PostgreSQL	**Sistema relacional Orientado a Objetos**. Considerado como el ***SGBD* de código abierto más avanzado del mundo**. Desarrollado por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyados por organizaciones comerciales. Es **multiplataforma y accesible desde múltiples lenguajes de programación**.
Firebird	*SGBD* **relacional, multiplataforma**, con **bajo consumo de recursos**, excelente gestión de la **concurrencia**, **alto rendimiento** y potente soporte para diferentes lenguajes. **Basado en la versión 6 de *Interbase***, cuyo código fue liberado por *Borland* en 2000. Su código fue reescrito de *C* a *C++*.
Apache Derby	*SGBD* escrito en *Java*, de **reducido tamaño** (pocos *MBytes*), con **soporte multilenguaje, multiplataforma, altamente portable, puede funcionar embebido o en modo cliente/servidor**.
SQLite	Sistema relacional, basado en una **biblioteca escrita en *C* que interactúa directamente con los programas**, reduce los tiempos de acceso siendo más rápido que *MySQL* o *PostgreSQL*, es multiplataforma y con soporte para varios lenguajes de programación.

7 *BD* centralizadas.

Si nos preguntamos cómo es la arquitectura de un sistema de *BD*, hemos de saber que todo depende del sistema informático que la sustenta. Tradicionalmente, la arquitectura centralizada **fue la que se utilizó inicialmente, aunque hoy en día es de las menos utilizadas**.

***Sistema de BD centralizado**: es aquella estructura en la que el SGBD está implantado en una sola plataforma u ordenador desde donde se gestiona directamente, de modo centralizado, la totalidad de los recursos. Es la arquitectura de los centros de proceso de datos tradicionales. Se basa en tecnologías sencillas, muy experimentadas y de gran robustez.*

Los sistemas de los años sesenta y setenta eran totalmente centralizados, como corresponde a los *SO* 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.

Las principales **características** de las *BD* centralizadas son:

- ✓ **Se almacena completamente en una ubicación central**, es decir, todos los componentes del sistema residen en un solo ordenador o sitio.
- ✓ **No posee múltiples elementos de procesamiento ni mecanismos de intercomunicación** como las *BD* distribuidas.
- ✓ Los **componentes** de las *BD* centralizadas son: **los datos, el software de gestión de *BD* y los dispositivos de almacenamiento** secundario asociados.
- ✓ Son **sistemas en los que su seguridad puede verse comprometida más fácilmente**.

Ventajas	Inconvenientes
Se evita la redundancia debido a la posibilidad de inconsistencias y al desperdicio de espacio.	Un **mainframe en comparación de un sistema distribuido no tiene mayor poder de cómputo**.
Se evita la inconsistencia. Ya que si un hecho específico se representa por una sola entrada, la no-concordancia de datos no puede ocurrir.	Cuando un sistema de ***BD* centralizado falla, se pierde toda disponibilidad de procesamiento** y sobre todo de información confiada al sistema.
La seguridad se centraliza.	En **caso de un desastre o catástrofe, la recuperación es difícil de sincronizar**.
Puede conservarse la integridad.	Las **cargas de trabajo no se pueden difundir entre varios ordenadores**, ya que los trabajos siempre se ejecutarán en la misma máquina.
El **procesamiento de los datos ofrece un mejor rendimiento**.	Los departamentos de sistemas retienen el control de toda la organización.
Mantenimiento más barato. Mejor uso de los recursos y menores recursos humanos.	Los sistemas centralizados **requieren un mantenimiento central de datos**.

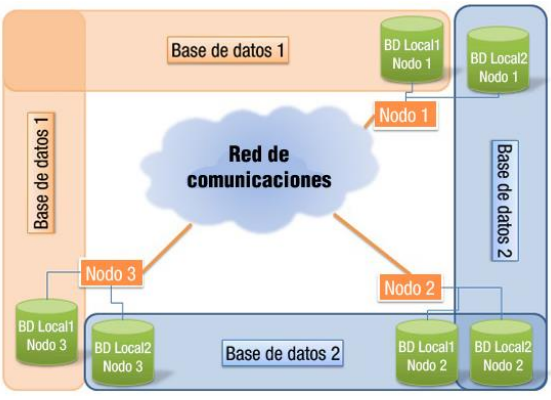
8 *BD* distribuidas.

***BD distribuida (BDD)**: es un conjunto de múltiples BD lógicamente relacionadas las cuales se encuentran distribuidas entre diferentes nodos interconectados por una red de comunicaciones.*

***Sistema de BD distribuida (SBDD)**: es un sistema en el cual múltiples sitios de BD están ligados por un sistema de comunicaciones, de tal forma que, un usuario en cualquier sitio puede acceder los datos en cualquier parte de la red exactamente como si los datos estuvieran almacenados en su sitio propio.*

***SGBD distribuido (SGBDD)**: es aquel que se encarga del manejo de la BDD y proporciona un mecanismo de acceso que hace que la distribución sea transparente a los usuarios (la aplicación trabajaría, desde un punto de vista lógico, como si un solo SGBD ejecutado en una sola máquina, administrara esos datos).*

Un *SGBDD* desarrollará su trabajo a través de un conjunto de sitios o nodos, que poseen un sistema de procesamiento de datos completo con una *BD* local, un *SGBD* e interconectados entre sí. Si estos nodos están dispersos geográficamente se interconectarán a través de una red de área amplia o *WAN*, pero si se encuentran en edificios relativamente cercanos, pueden estar interconectados por una red de área local o *LAN*. Este tipo de sistemas es utilizado en organizaciones con estructura descentralizada, industrias de manufactura con múltiples sedes, aplicaciones militares, líneas aéreas, cadenas hoteleras, servicios bancarios, etc.



Ventajas	Inconvenientes
El **acceso y procesamiento de los datos es más rápido** ya que varios nodos comparten carga de trabajo.	La **probabilidad de violaciones de seguridad es creciente** si no se toman las precauciones debidas.
Desde una ubicación puede accederse a información alojada en diferentes lugares.	Existe una **complejidad añadida que es necesaria para garantizar la coordinación apropiada entre los nodos**.
Los **costes son inferiores a los de las *BD* centralizadas**.	La inversión inicial es menor, pero el **mantenimiento y control puede resultar costoso**.
Existe **cierta tolerancia a fallos**. Mediante la replicación, si un nodo deja de funcionar el sistema completo no deja de funcionar.	Dado que los datos pueden estar replicados, el **control de concurrencia y los mecanismos de recuperación son mucho más complejos** que en un sistema centralizado.
El enfoque distribuido de las *BD* **se adapta más naturalmente a la estructura de las organizaciones**. Permiten la incorporación de nodos de forma flexible.	El **intercambio de mensajes y el cómputo adicional necesario para conseguir la coordinación entre los distintos nodos constituyen una forma de sobrecarga** que no surge en los sistemas centralizados.
Aunque los nodos están interconectados, tienen independencia local.	Dada la complejidad del procesamiento entre nodos es difícil asegurar la corrección de los algoritmos, el funcionamiento correcto durante un fallo o la recuperación.

8.1 Fragmentación.

Se sabe que **en los sistemas de *BDD* la información se encuentra repartida en varios lugares**. La forma de extraer los datos consultados puede realizarse mediante la fragmentación de distintas tablas pertenecientes a distintas *BD* que se encuentran en diferentes servidores. **El problema de la fragmentación se refiere al particionamiento de la información para distribuir cada parte a los diferentes sitios de la red**.

La fragmentación en BD se refiere al proceso de dividir una BD grande en fragmentos más pequeños y distribuirlos en varios sistemas de almacenamiento o nodos de una red. Esto se hace para mejorar la eficiencia y el rendimiento de la BD, así como para aumentar la disponibilidad y la capacidad de escalabilidad.

Pero hay que tener en cuenta el **grado de fragmentación** que se aplicará, ya que éste es un **factor determinante a la hora de la ejecución de consultas**. Si no existe fragmentación, se tomarán las relaciones o tablas como la unidad de fragmentación. Pero también **puede fragmentarse a nivel de tupla (fila o registro) o a nivel de atributo (columna o campo) de una tabla**. No será adecuado un grado de fragmentación nulo, ni tampoco un grado de fragmentación demasiado alto. El **grado de fragmentación deberá estar equilibrado y dependerá de las particularidades de las aplicaciones que utilicen dicha *BD***. Concretando, **el objetivo de la fragmentación es encontrar un nivel de particionamiento adecuado en el rango que va desde tuplas o atributos hasta relaciones completas**.

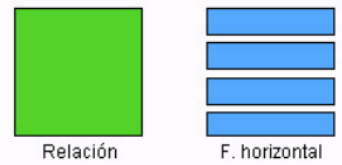
Cuando se lleva a cabo una fragmentación, existen **tres reglas fundamentales a cumplir**:

- ✓ **Completitud**: si una relación *R* se descompone en fragmentos *R1*, *R2*, ..., *Rn*, **cada elemento de datos que pueda encontrarse en *R* deberá poder encontrarse en uno o varios fragmentos *Ri***.
- ✓ **Reconstrucción**: si una relación *R* se descompone en una serie de fragmentos *R1*, *R2*, ..., *Rn*, **la reconstrucción de la relación a partir de sus fragmentos asegura que se preservan las restricciones definidas sobre los datos**.
- ✓ **Disyunción**: si una relación *R* se descompone verticalmente, sus **atributos primarios clave normalmente se repiten en todos sus fragmentos**.

La fragmentación puede proporcionar varios **beneficios**, como una **mejor utilización de los recursos de almacenamiento y procesamiento**, una **mayor disponibilidad** y **resistencia a fallos**, una **mejor escalabilidad** y un **rendimiento mejorado** al realizar consultas distribuidas. Sin embargo, también puede introducir **desafíos adicionales**, como la necesidad de **coordinar y sincronizar los fragmentos y garantizar la integridad de los datos entre ellos**.

Es importante tener en cuenta que la **fragmentación en *BD* generalmente se realiza en entornos distribuidos** o paralelos, donde múltiples nodos o sistemas de almacenamiento trabajan juntos como un conjunto coherente de *BD*

distribuidas.



Hay distintos **métodos de fragmentación** que se utilizan en las *BD*, entre ellos:

- ✓ **Fragmentación horizontal**: **se realiza sobre las tuplas o filas de la relación**, **dividiendo la relación en subrelaciones** que contienen un subconjunto de las tuplas que alberga la primera. Los **fragmentos se definen mediante una operación de selección**. Su **reconstrucción se realizará mediante la unión** de los fragmentos componentes. Existen **dos variantes**:
  - **Primaria**: particionar las tuplas de una relación en subconjuntos, donde cada subconjunto tenga propiedades comunes.
  - **Derivada**: consiste en dividir una relación partiendo de las condiciones definidas sobre alguna otra.

SNUM	NOMBRE	CD
S1	X	L
S2	Y	L
S3	Z	P
S4	A	P
S5	B	L

FH1 *CD='L'*

SNUM	NOMBRE	CD
S1	X	L
S2	Y	L
S5	B	L

FH2 *CD='P'*

SNUM	NOMBRE	CD
S3	Z	P
S4	A	P

En la imagen se muestra una fragmentación horizontal por la columna *CD*. Se puede comprobar que se tienen dos fragmentos primarios (*FH1* y *FH2*). En caso de que en la columna *CD* se tuvieran más valores distintos, se generarían tantos fragmentos como posibles valores.

> *Los datos se dividen en filas y se distribuyen en múltiples nodos. Cada nodo contendrá un subconjunto de las filas de la tabla original.*

Ejemplo: dada la tabla de la imagen realizar una fragmentación horizontal por provincia.

En este ejemplo, se crean tres fragmentos horizontales de la tabla dada basados en la provincia. Los usuarios que pertenecen a la misma provincia se agrupan en el mismo nodo. Cada nodo es responsable de almacenar y administrar los datos de los usuarios de una provincia específica.

ID	Nombre	Apellido	Correo electrónico	Fecha de registro	Provincia
1	Juan	Pérez	juan@example.com	2022-01-01	Madrid
2	Ana	García	ana@example.com	2022-01-05	Barcelona
3	Carlos	Torres	carlos@example.com	2022-01-10	Valencia
4	María	López	maria@example.com	2022-02-15	Madrid
5	Luis	Martínez	luis@example.com	2022-02-20	Barcelona
6	Laura	Sánchez	laura@example.com	2022-02-25	Valencia
7	Pedro	Ramírez	pedro@example.com	2022-03-10	Madrid
8	Andrea	Rodríguez	andrea@example.com	2022-03-15	Barcelona
9	Martín	Vargas	martin@example.com	2022-03-20	Valencia

Nodo 1:

ID	Nombre	Apellido	Correo electrónico	Fecha de registro	Provincia
1	Juan	Pérez	juan@example.com	2022-01-01	Madrid
4	María	López	maria@example.com	2022-02-15	Madrid
7	Pedro	Ramírez	pedro@example.com	2022-03-10	Madrid

Nodo 2:

ID	Nombre	Apellido	Correo electrónico	Fecha de registro	Provincia
2	Ana	García	ana@example.com	2022-01-05	Barcelona
5	Luis	Martínez	luis@example.com	2022-02-20	Barcelona
8	Andrea	Rodríguez	andrea@example.com	2022-03-15	Barcelona

Nodo 3:

ID	Nombre	Apellido	Correo electrónico	Fecha de registro	Provincia
3	Carlos	Torres	carlos@example.com	2022-01-10	Valencia
6	Laura	Sánchez	laura@example.com	2022-02-25	Valencia
9	Martín	Vargas	martin@example.com	2022-03-20	Valencia

Ejemplo: fragmentación horizontal derivada por provincia en dos tablas relacionadas: usuarios y pedidos.

Tabla "Usuarios":

ID	Nombre	Apellido	Correo electrónico	Provincia
1	Juan	Pérez	juan@example.com	Madrid
2	Ana	García	ana@example.com	Barcelona
3	Carlos	Torres	carlos@example.com	Valencia
4	María	López	maria@example.com	Madrid
5	Luis	Martínez	luis@example.com	Barcelona
6	Laura	Sánchez	laura@example.com	Valencia
7	Pedro	Ramírez	pedro@example.com	Madrid
8	Andrea	Rodríguez	andrea@example.com	Barcelona
9	Martín	Vargas	martin@example.com	Valencia

Tabla "Pedidos":

ID	Usuario_ID	Producto	Cantidad
1	1	A	5
2	2	B	3
3	3	C	2
4	4	A	4
5	5	B	1
6	6	C	2
7	7	A	6
8	8	B	2
9	9	C	3

Para realizar la fragmentación horizontal derivada, se va a dividir los datos de las tablas "Usuarios" y "Pedidos" en tres nodos basados en la provincia. A continuación, se muestra cómo quedarían los fragmentos:

Nodo 1:

Tabla "Usuarios":

ID	Nombre	Apellido	Correo electrónico	Provincia
1	Juan	Pérez	juan@example.com	Madrid
4	María	López	maria@example.com	Madrid
7	Pedro	Ramírez	pedro@example.com	Madrid

Tabla "Pedidos":

ID	Usuario_ID	Producto	Cantidad
1	1	A	5
4	4	A	4
7	7	A	6

Nodo 2:

Tabla "Usuarios":

ID	Nombre	Apellido	Correo electrónico	Provincia
2	Ana	García	ana@example.com	Barcelona
5	Luis	Martínez	luis@example.com	Barcelona
8	Andrea	Rodríguez	andrea@example.com	Barcelona

Tabla "Pedidos":

ID	Usuario_ID	Producto	Cantidad
2	2	B	3
5	5	B	1
8	8	B	2

Nodo 3:

Tabla "Usuarios":

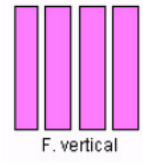
ID	Nombre	Apellido	Correo electrónico	Provincia
3	Carlos	Torres	carlos@example.com	Valencia
6	Laura	Sánchez	laura@example.com	Valencia
9	Martín	Vargas	martin@example.com	Valencia

Tabla "Pedidos":

ID	Usuario_ID	Producto	Cantidad
3	3	C	2
6	6	C	2
9	9	C	3

En esta fragmentación derivada, cada nodo contiene una combinación de usuarios y pedidos basados en la provincia. Esto permite una distribución equilibrada de los datos relacionados entre los nodos, considerando tanto la información de usuarios como los pedidos asociados a ellos en función de la provincia a la que pertenecen.

- ✓ **Fragmentación vertical**: la fragmentación vertical, en cambio, **se basa en los atributos de la relación para efectuar la división**. Una **relación *R* produce fragmentos *R1*, *R2*, ..., *Rr*, cada uno de los cuales contiene un subconjunto de los atributos de *R*, así como la llave primaria de *R***.



El objetivo de la fragmentación vertical es particionar una relación en un conjunto de relaciones más pequeñas de manera que varias de las aplicaciones de usuario se ejecutarán sobre un fragmento. En este contexto, una fragmentación óptima es aquella que produce un esquema de fragmentación que minimiza el tiempo de ejecución de las consultas de usuario.

La fragmentación vertical es más complicada que la horizontal, ya que existe un gran número de alternativas para realizarla.

En la imagen se muestra una fragmentación vertical de una relación/tabla empleados por los atributos/columnas *NOMBRE/JEFE/DEPT* y *SALARIO/IMPTO*. Se puede comprobar que se tienen dos fragmentos (*FV1* y *FV2*). Resaltar que el atributo (o atributos) clave debe aparecer en todos los fragmentos, en el ejemplo *EMP#*.

EMP#	NOMBRE	SALARIO	IMPTO	JEFE	DEPT
e1	X	1000	100	J1	D1
e2	Y	1500	300	J1	D1
e3	Z	500	20	J2	D2
e4	A	4000	1000	J3	D3
e5	B	2000	350	J2	D2

EMP#	NOMBRE	JEFE	DEPT
e1	X	J1	D1
e2	Y	J1	D1
e3	Z	J2	D2
e4	A	J3	D3
e5	B	J2	D2

FV1= E[EMP#, NOMBRE., JEFE, DEPT]

EMP#	SALARIO	IMPTO
e1	1000	100
e2	1500	300
e3	500	20
e4	4000	1000
e5	2000	350

FV2= E[EMP#, SALARIO.,IMPTO]

Los datos se dividen en columnas y se distribuyen en múltiples nodos. Cada nodo contendrá un subconjunto de las columnas de la tabla original.

Ejemplo: fragmentación vertical de la tabla usuarios.

ID	Nombre	Apellido	Dirección
1	Juan	Pérez	Calle 123
2	María	López	Avenida 456
3	Carlos	Rodríguez	Calle 789
4	Ana	Gómez	Avenida 012

En este ejemplo, el Fragmento vertical 1 contiene las columnas "ID", "Nombre" y "Apellido", mientras que el Fragmento vertical 2 contiene las columnas "ID" y "Dirección". Cada fragmento muestra una parte de la información original de la tabla.

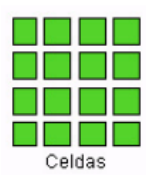
Fragmentación vertical 1: Fragmento 1

ID	Nombre	Apellido
1	Juan	Pérez
2	María	López
3	Carlos	Rodríguez
4	Ana	Gómez

Fragmentación vertical 2: Fragmento 2

ID	Dirección
1	Calle 123
2	Avenida 456
3	Calle 789
4	Avenida 012

- ✓ **<u>Fragmentación híbrida o mixta</u>**: se pueden **combinar ambas** (horizontal y vertical), utilizando por ello la denominada fragmentación mixta. **Tipos**:
  - **Fragmentación mixta *HV***: es una fragmentación horizontal seguida de una fragmentación vertical, sobre cada uno de los fragmentos horizontales.
  - **Fragmentación mixta *VH***: si tras una fragmentación vertical se lleva a cabo otra horizontal.
  - **Celdas**: se aplican ambas fragmentaciones simultáneamente sobre la relación, formando una red de celdas. Es la única en la que todos los fragmentos obtenidos son iguales.



> *Es una combinación de fragmentación horizontal y vertical, donde los datos se dividen tanto en filas como en columnas.*

<u>Ejemplo</u>: fragmentación mixta *HV* de la tabla empleados.

Tabla: Empleados

ID	Nombre	Departamento	Puesto	Salario
1	John Smith	Marketing	Gerente	$5000
2	Jane Doe	Ventas	Ejecutivo	$4000
3	Mike Brown	Finanzas	Analista	$3500
4	Lisa Green	Ventas	Representante	$3000
5	Alex Lee	Marketing	Asistente	$2500

Para realizar una fragmentación mixta de la tabla de empleados, hay que aplicar una fragmentación horizontal primero y seguidamente una fragmentación vertical a los fragmentos horizontales obtenidos.

Fragmentación horizontal → se dividirá la tabla en tres fragmentos horizontales basados en el atributo Departamento.

Fragmento 1 - Marketing:

ID	Nombre	Departamento	Puesto	Salario
1	John Smith	Marketing	Gerente	$5000
5	Alex Lee	Marketing	Asistente	$2500

Fragmento 2 - Ventas:

ID	Nombre	Departamento	Puesto	Salario
2	Jane Doe	Ventas	Ejecutivo	$4000
4	Lisa Green	Ventas	Representante	$3000

Fragmento 3 - Finanzas:

ID	Nombre	Departamento	Puesto	Salario
3	Mike Brown	Finanzas	Analista	$3500

Si se desea realizar una fragmentación vertical a los tres fragmentos horizontales obtenidos anteriormente, teniendo en cuenta todas las filas, se puede dividir cada fragmento en dos fragmentos verticales basados en los atributos: Nombre/Puesto y Nombre/Salario.

Fragmento 1 - Marketing:

ID	Nombre	Puesto
1	John Smith	Gerente
5	Alex Lee	Asistente

ID	Nombre	Salario
1	John Smith	$5000
5	Alex Lee	$2500

Fragmento 2 - Ventas:

ID	Nombre	Puesto
2	Jane Doe	Ejecutivo
4	Lisa Green	Representante

ID	Nombre	Salario
2	Jane Doe	$4000
4	Lisa Green	$3000

Fragmento 3 - Finanzas:

ID	Nombre	Puesto
3	Mike Brown	Analista

ID	Nombre	Salario
3	Mike Brown	$3500

Es importante tener en cuenta que la fragmentación mixta puede variar según las necesidades y consultas

específicas del sistema. El ejemplo proporcionado muestra una posible forma de realizar la fragmentación vertical, pero la elección de los atributos y cómo se dividen depende de los requisitos y objetivos particulares del sistema.

- ✓ **Fragmentación funcional**: los datos se fragmentan según alguna función o criterio específico. Por ejemplo, los datos pueden dividirse en función de la ubicación geográfica o la categoría de los usuarios.

Ejemplo: fragmentación funcional de la tabla productos.

Tabla: Productos

ID	Nombre	Categoría	Precio	Stock
1	Producto A	Categoría1	$10	100
2	Producto B	Categoría2	$20	150
3	Producto C	Categoría1	$15	80
4	Producto D	Categoría3	$25	120
5	Producto E	Categoría2	$18	90

Se quiere fragmentar esta tabla en un entorno distribuido considerando las funciones más frecuentes en el sistema:

- Fragmento 1 - Consulta de precios: este fragmento contendrá solo los atributos ID, Nombre, Categoría y Precio de los productos.
- Fragmento 2 - Consulta de stock: este fragmento contendrá solo los atributos ID, Nombre, Categoría y Stock de los productos.

Al fragmentar la tabla de esta manera, se puede mejorar el rendimiento al realizar consultas específicas sin tener que acceder a todos los atributos de la tabla original. Por ejemplo, si se necesita realizar una consulta de precios, solo se necesita acceder al Fragmento 1, que contiene los datos necesarios para esta función. Del mismo modo, si se necesita consultar el stock, se puede acceder directamente al Fragmento 2.

Es importante tener en cuenta que la fragmentación funcional debe diseñarse cuidadosamente para reflejar las consultas más frecuentes y asegurarse de que los fragmentos resultantes contengan la información necesaria para satisfacer esas consultas. Además, es posible que sea necesario implementar mecanismos de coordinación y sincronización entre los fragmentos para garantizar la consistencia de los datos en el sistema distribuido.

La fragmentación funcional de la tabla "Productos" quedaría de la siguiente forma:

Fragmento 1 - Consulta de precios:

ID	Nombre	Categoría	Precio
1	Producto A	Categoría1	$10
2	Producto B	Categoría2	$20
3	Producto C	Categoría1	$15
4	Producto D	Categoría3	$25
5	Producto E	Categoría2	$18

Fragmento 2 - Consulta de stock:

ID	Nombre	Categoría	Stock
1	Producto A	Categoría1	100
2	Producto B	Categoría2	150
3	Producto C	Categoría1	80
4	Producto D	Categoría3	120
5	Producto E	Categoría2	90

TAREA

12. ¿Qué permiten las *BD distribuidas*? Enumerar sus ventajas e inconvenientes.

13. Considerar la siguiente relación, ¿qué fragmentos se obtendrían al realizar una fragmentación horizontal por *ESCUELA*?

Jno	NOMBRE	NOTA	ESCUELA
J1	LUIS YANEZ	8	CIME
J2	ERIKA QUIROZ	8	CIME
J3	DANIEL MURILLO	9	EISIC
J4	MARIA JOSE MENDEZ	10	EISIC
J5	ADONIS PABON	9	EISIC

14. Considerar las siguientes relaciones, ¿qué fragmentos principales y derivados se obtendrían al realizar una fragmentación horizontal por *ESCUELA*?

ID	MATERIA	ESCUELA	CRÉDITOS	NIVEL
1	Análisis Matemático	EISIC	4	1
2	Sistemas Operativos	EISIC	6	3
3	Programación II	CIME	6	2
4	Tecnología Eléctrica	EISIC	4	2
5	Técnicas de Aprendizaje	CIME	4	1
6	Dibujo Mecánico	CIME	6	3

Jno	NOMBRE	MATERIA	NOTA
J1	LUIS YANEZ	Sistemas Operativos	8
J2	ERIKA QUIROZ	Dibujo Mecánico	8
J3	DANIEL MURILLO	Técnicas de Aprendizaje	9
J4	MARIA JOSE MENDEZ	Análisis Matemático	10
J5	ADONIS PABON	Programación II	9

15. Considerar la relación del ejercicio 13. Realizar una fragmentación vertical por *NOTA* y por *NOMBRE/ESCUELA*.

16. Realizar una fragmentación mixta *HV*. La fragmentación horizontal se debe realizar por *ESCUELA* y la vertical por *ESCUELA/NOTA_INGRESO/NOMBRE* y *NOMBRE/BECA*.

DNI	Escuela	Nombre	Nota ingreso	Beca
87633483	EUI	Concha Queta	5.6	No
99855743	EUI	Josechu Letón	7.2	Si
33887293	EUIT	Oscar Romato	6.1	Si
05399075	EUI	Bill Gates	5.0	No
44343234	EUIT	Pepe Pótamo	8.0	No
44543324	EUI	Maite Clado	7.5	Si
66553234	EUIT	Ernesto Mate	6.6	No