Módulo 1. Datos

Introducción

En el módulo 1, conoceremos diferentes medios de almacenamiento, y detallaremos uno de los medios que más se usa: el disco rígido. Luego, veremos el concepto de archivo, y los diferentes tipos existentes, para más tarde conocer las operaciones y formas en las que se puede organizar un archivo. Ahondaremos en los archivos secuenciales y *hash*. Sobre estos últimos, veremos el encadenamiento y direccionamiento.

Bienvenidos a la empresa File Data: caso práctico

Video de inmersión

Unidad 1. Almacenamiento y archivos

Instalación de una base de datos

Tema 1. Tipos de almacenamiento

Las bases de datos, independientemente del modelo de datos que se haya usado para su diseño, se almacenan físicamente como archivos de registros; normalmente, se almacenan en discos rígidos. A lo largo del desarrollo de este apartado, hablaremos de las técnicas que permiten acceder a los datos en su espacio de almacenamiento de la manera más eficiente y eficaz posible. Además, presentaremos la organización de la base de datos en su espacio de almacenamiento.

La colección de datos, que constituye una base de datos, debe guardarse en algún medio de almacenamiento. De esta manera, el sistema gestor de bases de datos (SGBD) debe ser capaz de recuperar, actualizar y procesar los datos, siempre que sea necesario. Los medios de almacenamiento forman una jerarquía llamada jerarquía de almacenamiento o jerarquía de memoria, la cual contiene dos categorías principales:

- **Almacenamiento primario.** Incluye la memoria principal y la memoria caché. Son medios de almacenamiento muy rápidos, aunque con capacidad limitada.
- **Almacenamiento secundario:** incluye discos magnéticos, discos ópticos, cintas, etc. Son medios de almacenamiento de grandes capacidades, aunque con un acceso mucho más lento.

Si bien el nivel de almacenamiento primario se caracteriza por ser más caro y rápido que el secundario, podemos decir que, dentro de cada uno de estos niveles, hay subniveles; también, agregamos a estos criterios los conceptos de coste y velocidad. Así, en el nivel de almacenamiento primario, destacamos los siguientes subniveles:

- **Memoria caché.** Es la capa más cara de la jerarquía, y se trata de una memoria RAM estática que usa la CPU para aumentar la velocidad de ejecución de las máquinas. Es volátil.
- **Memoria DRAM (RAM dinámica) o memoria principal:** la usa la CPU para mantener los programas y los datos. Es más lenta que la RAM estática, aunque más barata. Es volátil.
- **Memoria *flash*:** memoria no volátil que se sitúa en el último subnivel dentro del nivel primario.

Por otra parte, en el nivel de almacenamiento secundario, tenemos los siguientes elementos:

- Discos de estado sólido (SSD).
- Discos rígidos: son los que más se usan en este nivel por su relación velocidad/precio.
- Discos que se basan en tecnología óptica con lectura/escritura láser: *compact disks* (CD), *digital versatile disks* (DVD), *blu-ray disks*.
- Las cintas: se encuentran en el nivel más barato.

En cualquier caso, tanto el nivel de almacenamiento primario como el secundario se usan para las bases de datos. La memoria principal, siempre que tenga capacidad suficiente, se puede usar para albergar datos (o parte de ellos), manteniendo siempre una copia de respaldo en memoria secundaria para evitar los problemas que podrían ocasionar la volatilidad de la primera. No obstante, en la gran mayoría de las ocasiones, el tamaño de la base de datos impide su almacenamiento en una memoria primaria, y se mantiene en almacenamiento secundario utilizando la memoria primaria para cargar índices que aumentan la velocidad de acceso a los

datos.

Dentro de la jerarquía del almacenamiento secundario, el principal soporte hasta el momento es el disco rígido; asimismo, los discos de estado sólido son protagonistas en la actualidad. En este curso, nos centraremos en ver el funcionamiento de los discos rígidos, el principal medio en las bases de datos.

Tema 2. Archivos, tipos de archivos y operaciones básicas

Estructura de un archivo y tipos

Un **archivo**, en base de datos, es una secuencia de registros. Estos registros pueden ser de longitud fija o variable. Cada **registro** es una colección de **campos** que se asocian de manera independiente con un puntero. Por lo general, un registro está asociado a una entidad, es decir, posee una serie de campos con valores relacionados entre sí. Los archivos pueden clasificarse según la forma en la que se manejen los conceptos mencionados anteriormente.

Figura 1: Organización de archivos según el tipo de registro

Fuente: elaboración propia.

Organización de un archivo

☰ Según el tipo de registro

- **Extendida:** los espacios libres en cada bloque se ocupan o se aprovechan. Para hacerlo, se almacena parte del registro en un bloque, y lo restante en otro. El enlace de ambos bloques se realiza a través de un puntero al final del bloque. De este modo, se indica cuál

es el bloque en el que continuará el registro. Se usa cuando el registro es mayor que el bloque.

- **No extendido:** los espacios libres no se utilizan. Deriva en un procesamiento de datos más simple, pero el espacio del disco no se utiliza de manera óptima.

☰ Según el ordenamiento de los registros

- **Archivos desordenados:** en este tipo de archivos, no existe un ordenamiento; los registros se insertan al final del archivo en la medida que se ingresan.
- **Archivos ordenados o secuenciales:** hay un orden al momento de la inserción de los registros. Se ordenan físicamente con algún campo clave. Por ejemplo, en el archivo de artículos, se ordena por nombre de artículo.

Operaciones básicas

Las operaciones básicas que podemos hacer con los archivos son las que se describen a continuación:

- **Insertar.** Esta operación permite poblar la base de datos. Las inserciones pueden ser individuales o masivas, según se ingresen uno o muchos registros al archivo. Por ejemplo, una inserción individual podría ocurrir en un sistema de facturación cuando damos de alta a un cliente. La inserción masiva puede suceder cuando se importan, masivamente, todos los artículos, mediante la importación de un archivo externo como un libro de Excel.
- **Extraer:** se busca un determinado dato/campo para ser presentado. Esta búsqueda se puede realizar con alguna condición para delimitar el universo de archivos que se deban recorrer. Este tipo de operación no modifica los datos de los archivos, ni los de la base de datos. Suele ser llamada **operación de lectura.** Una funcionalidad de este tipo de operaciones es traer o leer todos los artículos de un determinado rubro, por ejemplo, listar todos los artículos del rubro electrónico.
- **Actualizar:** se busca un determinado dato o universo de datos, con el fin de realizar una modificación sobre un campo o varios campos. Al igual que la inserción, esto se puede hacer de forma masiva, si queremos incrementar en un 5 % el precio de todos los artículos del rubro de electrónica o, de forma individual, si queremos incrementar el precio de un solo artículo.

- **Eliminar:** es la operación inversa a la inserción de datos. Cuenta con la dinámica de poder procesar de forma masiva o de manera individual. Es importante tener presente, en este tipo de operación —principalmente cuando se hace de manera masiva—, que la recuperación puede no ser posible. Deberemos estar muy seguros al momento de hacer la eliminación.

Actividad de repaso

¿Es el disco rígido el medio de almacenamiento más barato?

Verdadero.

Falso.

Justificación

Tema 3. Archivos secuenciales y ordenados

Los archivos secuenciales se caracterizan por tener un orden físico según los valores de uno de sus campos: el campo de ordenación. Si tiene, además, propiedades de identificador, es decir, si siempre tiene valor y no se repite, se conoce como clave de ordenación del archivo.

Estos tipos de archivos poseen ciertas ventajas:

a) La lectura de registros en el orden físico almacenado es extraordinariamente eficiente.

b) Los accesos a registros son en orden consecutivo y no se requiere cambiar de bloque en la mayoría de los casos, por lo que se optimiza el acceso al disco.

c) Las condiciones de búsqueda que afectan a valores de un campo clave de ordenación son muy eficientes cuando se emplea una estrategia de búsqueda binaria. Una búsqueda binaria accede, normalmente, a log2(n) bloques, encuentre o no el registro que busca entre n bloques.

Figura 2: Archivo secuencial y ordenado

	NOMBRE	DNI	CUMPLEAÑOS	EDAD	SALARIO	SEXO
Bloque 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
Bloque 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
Bloque 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
Bloque 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
Bloque 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
Bloque 6	Arnol, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
	⋮					
Bloque n-1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
Bloque n	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

Fuente: elaboración propia.

Aun así, existen algunas desventajas. Analicemos cada una de las operaciones sobre archivos.

La inserción es una operación costosa, ya que se debe conservar el orden físico de los registros:

a) encontrar la posición para el nuevo registro en el fichero, según el campo de ordenación;
b) abrir espacio en el bloque correspondiente, desplazando el resto de los registros.

Esta última fase de la operación es especialmente dura, debido a que implica tener que leer y volver a escribir una media de n/2 bloques para un fichero de n bloques.

La búsqueda por un campo de ordenación —ya sabemos— es extremadamente eficiente; sin embargo, la búsqueda por cualquier otro tipo de campo se convierte en una tarea de búsqueda lineal.

El borrado implica primero buscar el registro, eliminarlo y mover los registros posteriores. Esto es una operación costosa. La única ventaja que existe es cuando hay búsqueda del registro a borrar por el campo de ordenación.

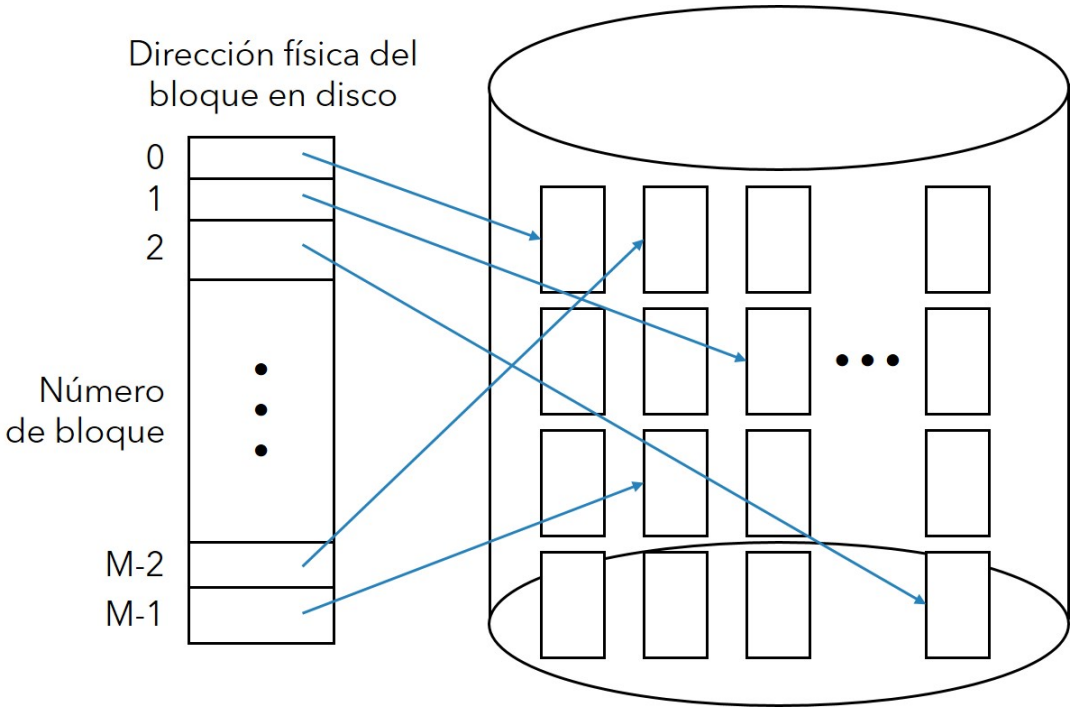
La actualización implica la búsqueda del registro, que solo es eficiente cuando la restricción contiene el campo de ordenación. Si lo que se ha modificado es el valor del campo de ordenación, entonces, se convierte en una operación de borrado más inserción del registro en su nueva ubicación, con la complejidad que ello conlleva.

La lectura secuencial del fichero ordenado es muy eficiente, siempre que se siga el campo de ordenación. En cualquier otro caso, es una búsqueda lineal.

Tema 4. Archivos *hash*. Función *hash*. Resolución de conflictos

Los archivos de direccionamiento calculado, también conocidos como dispersos o *hashing*, proporcionan un acceso muy rápido a los registros, bajo ciertas condiciones de búsqueda. La condición de búsqueda se realiza sobre un único campo del registro, que se conoce como **campo de direccionamiento calculado o clave *hash***. El funcionamiento de este tipo organización se basa en establecer una **función *hash*** que, una vez aplicada sobre el valor del campo de direccionamiento de un registro, produciría la dirección del bloque de disco en el que se almacena el registro buscado. Una vez que se obtiene el bloque, se copia a memoria principal, y se realiza la búsqueda del registro dentro del bloque.

Figura 3: Almacenamiento disperso en disco

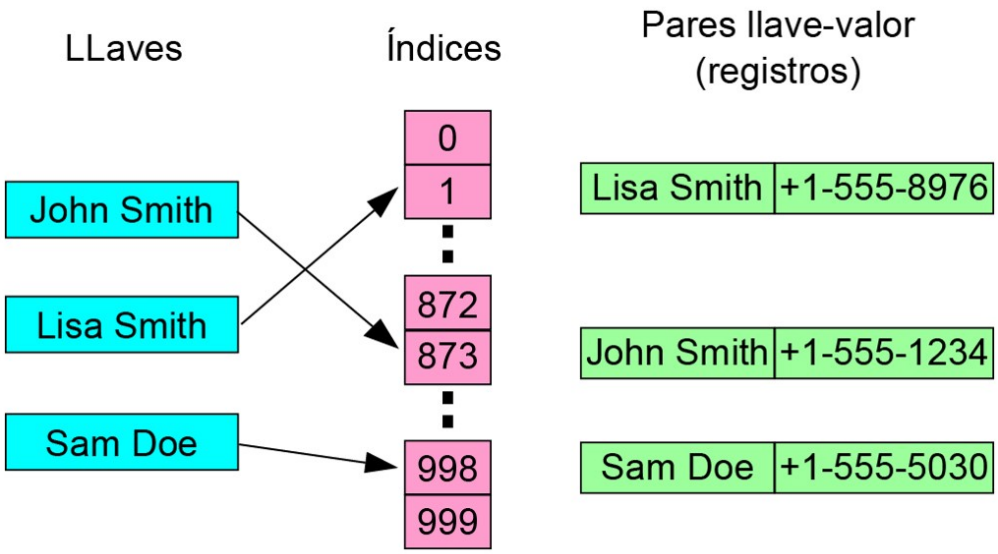


Fuente: elaboración propia.

Tabla *hash*

Una tabla *hash* es una estructura de datos, que se conoce, además, como tabla fragmentada. Esta estructura se basa en una operación en la que se asocian claves con valores. Esto permite lograr accesos muy rápidos al momento de buscar la información que se precisa, por ejemplo, la búsqueda de un artículo por su nombre a partir de una clave generada. Lo que se hace con esta estructura de tablas es transformar una clave a través de una función, **función de <**, en un número ***hash***, que permite localizar la posición de la tabla que posee el valor buscado.

Figura 4: Ejemplo de tabla *hash*



Fuente: [imagen sin título sobre tabla hash], s. f., https://bit.ly/3yynrbO.

Asimismo, si contamos con una buena función que genere la mínima cantidad posible de colisiones, obtendremos un excelente rendimiento de las tablas *hash*. Aclaremos: una colisión sucede cuando, para diferentes valores de claves, obtenemos el mismo valor de índice de tabla.

Para ese *hash* calculado, tenemos una lista de valores. En consecuencia, es necesario buscar el valor dentro de esta lista de forma secuencial. Cuando una función *hash* genera muchos valores iguales, las búsquedas son más lentas. Por este motivo, es importante encontrar una función *hash* que minimice las colisiones. No obstante, hallar esta función hash tiene un costo muy alto. Por esta razón, los esfuerzos se encausan en buscar alternativas para resolver el problema de las colisiones.

> Por la forma en la que se almacenan los datos, las **tablas *hash*** suelen ser más lentas para acceder, que los índices de árboles binarios autobalanceados.

Se recomienda usar la misma función *hash* para cualquier tamaño de *arrays*. Calcular la posición en el *array* consiste en dos pasos:

1. El primero es el *hash* genérico en el que, por medio del llenado de un entero natural de máquina, se calcula el valor.
2. El resultado del valor se reduce a un índice válido en el vector al realizar este segundo paso. Su módulo se encontrará en concordancia con el tamaño del *array*.

Como siempre, el objetivo es buscar el menor número de colisiones. El tamaño del vector de una tabla *hash*, generalmente, es un número primo. Esto evita que los *hashes* de grandes enteros tengan divisores comunes con el tamaño de la misma tabla.

¿Cuál es el problema más común en las funciones *hash*?

La aglomeración de valores. Esto se da cuando las claves más usadas caen todas muy cerca, y es probable que disminuya el rendimiento en una tabla de manera significativa. Utilizar una función *hash*, que devuelva un valor constante, es una manera de depurar el manejo de las colisiones. Por ejemplo, 1, y que cause colisión en cada inserción.

Resolución de colisiones

Hasta ahora, hemos mencionado, en muchas oportunidades, las colisiones, pero no hemos descrito a ciencia cierta cuándo se originan. Esto sucede cuando dos números de la función *hash* apuntan al mismo índice. Como no es posible almacenar dos registros en el mismo lugar, se debe encontrar una ubicación para el nuevo registro.

El ejemplo típico de colisión es cuando la clave se genera por medio de la fecha de nacimiento. La función *hash* correspondiente generará números aleatorios, pero, por más que nuestro vector tenga cerca de un millón de posiciones, hay una posibilidad del 95 % de que al menos se origine una colisión.

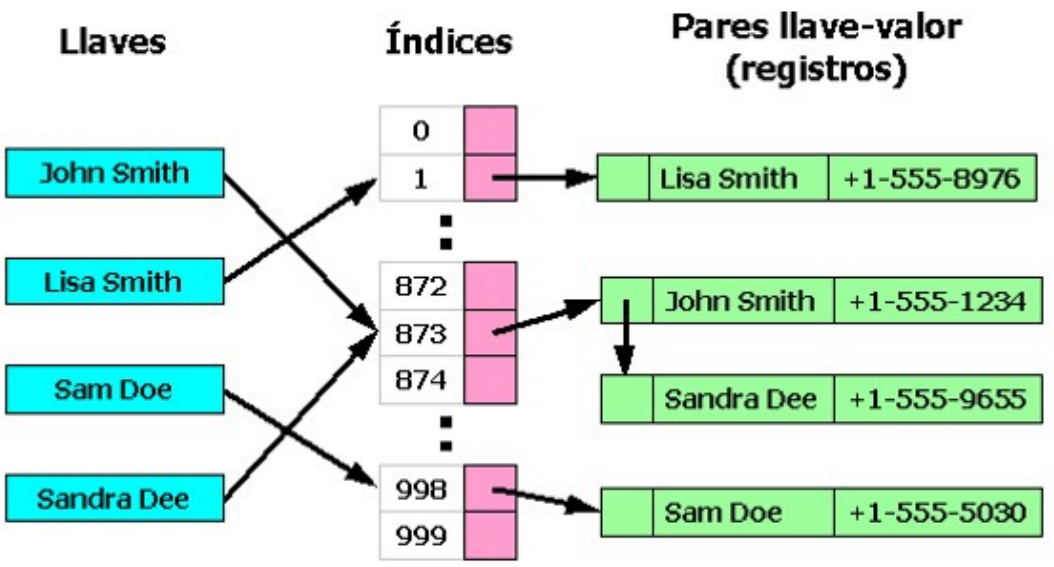
Por suerte, existen varias técnicas que brindan soluciones. Las más conocidas son las siguientes:

- encadenamiento;
- direccionamiento abierto.

Encadenamiento

El encadenamiento se da cuando una casilla del *array* hace referencia a una lista de registros que colisionan en esa casilla.

Figura 5: Ejemplo de encadenamiento



Fuente: [imagen sin título sobre encadenamiento], s. f., https://bit.ly/3LkqjvH.

Las principales ventajas del encadenamiento sobre el direccionamiento directo son las siguientes:

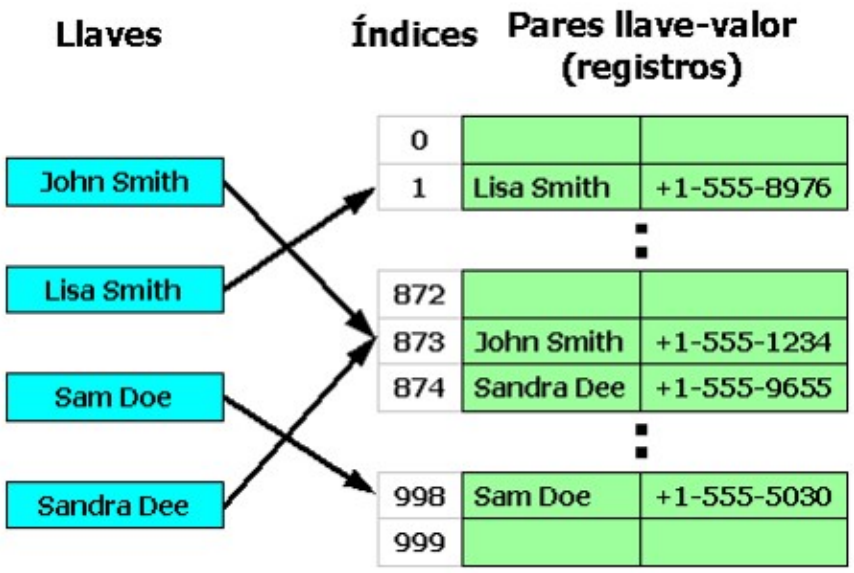
- El borrado es simple.
- El crecimiento de la tabla se puede posponer durante más tiempo, aun cuando todas las casillas están ocupadas.

Puede que las tablas *hash* encadenadas no requieran crecimiento nunca, debido a que la degradación del rendimiento es lineal, en la medida en la que se va llenando la tabla. Otra técnica de encadenamiento es usar árboles autobalanceados.

Direccionamiento abierto

En las tablas *hash* de direccionamiento abierto, podemos almacenar los registros en el *array*, directamente. Las colisiones se resolverán por medio de un sondeo del *array*, hasta encontrar el registro, o hasta llegar a una casilla vacía.

Figura 6: Ejemplo de direccionamiento abierto



Fuente: [imagen sin título sobre direccionamiento abierto], s. f., https://bit.ly/3Md4Ohp.

Sondeos más usados

- **Sondeo lineal:** el intervalo entre cada intento es constante. Favorece el rendimiento del caché, pero es más sensible a la aglomeración.
- **Sondeo cuadrático:** el intervalo entre los intentos aumenta linealmente. Se sitúa en medio en cuanto a rendimiento en caché y aglomeración.
- **Doble «*hasheo*»:** el intervalo entre intentos es constante para cada registro, pero es calculado por otra función *hash*. Disminuye el rendimiento en el caché, pero elimina el problema de aglomeración.

Funciones *hash* más usadas

De acuerdo con Gómez (2017):

1. ***Hash* de división:** dado un diccionario D, se fija un número M >= |D| (M mayor o igual al tamaño del diccionario) y que sea primo, no cercano a potencia de 2 o de 10. Siendo K la clave a buscar y h(K) la función hash, se tiene h(K) = K %M (resto de la división k/m).
2. ***Hash* de multiplicación:** si, por alguna razón, se necesita una tabla *hash* con tantos elementos o punteros como una potencia de 2 o de 10, será mejor usar una función *hash* de multiplicación, independientemente del tamaño de la tabla. Se escoge un tamaño de tabla M >= |D| (M mayor o igual al tamaño del diccionario) y un cierto

número irracional φ (normalmente, se usa 1+5^ (1/2) /2 o 1-5^ (1/2) /2). De este modo, se define h(K) = suelo (M*parte fraccionaria (K *φ)). (https://bit.ly/3LkqjvH).

Tabla 1: *Hash* perfecto vs. *hash* dinámico

Perfecto	Dinámico
La función de *hash* no duplica los valores de las claves o genera valores únicos.	Las estructuras de las tablas *hash* son del tipo árbol, y permiten almacenar un gran número de informaciones. Su mayor eficiencia la presentan en las operaciones de inserción, eliminación y búsqueda.
No se presentan colisiones.	La función *hash* que genere valores únicos no existe. No pueden crecer, ya que son estructuras estáticas, debido a que necesitan un tamaño fijo.

Fuente: elaboración propia.

Hash: aplicaciones

Existen múltiples aplicaciones que se relacionan con las tablas hash. Una de ellas es la del uso de la firma digital. Cada vez son más las empresas y entidades que se dedican, exclusivamente, a la seguridad en el software. Global Sign es una autoridad certificada por Web Trust que provee servicios de identidad. Fundada en Bélgica en 1996, la compañía ofrece una amplia gama de soluciones de servicios de identidad. Veamos cómo explican en su blog la aplicación del hash en el concepto de firma digital:

¿Cómo funcionan las firmas digitales?

Fuente: Global Sign. (2016) ¿Cómo funcionan las firmas digitales? Recuperado de https://www.globalsign.com/es/blog/como-funcionan-las-firmas-digitales/

Unidad 2. Índices y árboles

Tema 1. Archivos de índices

En el apartado anterior, vimos que hay diferentes formas de organización para almacenar la información. Analizamos archivos secuenciales y de direccionamiento calculado (dispersa o *hashing*). Sin embargo, estas estructuras pueden no ser suficientes para las necesidades de una base de datos. En muchos casos, es necesario hacer uso de unas estructuras de acceso

auxiliares llamadas **índices**. Los índices se emplean para aumentar la *performance* de acceso a registros, bajo ciertas condiciones de búsqueda. Dichas estructuras proporcionan caminos alternativos para acceder a la información deseada, sin que se vean afectados los datos puros dentro del disco.

Los campos de un registro que se utilizan para construir un índice se denominan **campos de indexación**. Cualquier campo del registro puede ser un campo de indexación y un mismo archivo puede incluir múltiples índices (es decir, más de un campo de indexación por registro).

Detallemos, entonces, el funcionamiento de un archivo de índice: cuando se produce una condición de búsqueda que afecta a un campo de indexación, en lugar de acceder directamente al archivo de datos para iniciar la búsqueda (según la organización propia del fichero de datos: ordenada o direccionamiento calculado), se accede previamente a un archivo (llamado índice o archivo de índice) en el que se encuentran los valores del campo de indexación correspondiente —ordenados según la organización propia del índice, junto con un puntero que señala físicamente al bloque de disco que alberga a ese registro en el archivo. El acceso al registro ya es una tarea trivial, según hemos visto.

El uso de índices permite llevar a cabo las siguientes acciones:

- Realizar búsquedas eficientes en archivos de datos organizados en montículo.
- Acceder, de forma eficiente, mediante condiciones de búsqueda que afectan a cualquier campo en un archivo ordenado (recordemos que a esta organización solo se accedía de forma eficiente cuando la búsqueda se realizaba sobre el campo de ordenación, y este era único para un determinado fichero).
- Acceder, de forma eficiente, mediante condiciones de búsqueda que afectan a cualquier campo en un archivo con direccionamiento calculado (este accede de forma eficiente cuando la búsqueda se refiere al campo de direccionamiento calculado, pero las búsquedas por otros campos se tienen que realizar secuencialmente).

Para poder aplicar los archivos de índice, se debe usar un archivo de tipo secuencial o realizarlo mediante el empleo de estructuras de datos en árbol (árboles binarios), que optimizan las búsquedas.

A partir del concepto de índice, podemos tener diferentes posibilidades al crear este tipo de archivos. Veamos, entonces, la clasificación de los diferentes índices que existen:

- Densos. Una entrada por cada valor de la clave de indexación.
- No densos o escasos: solo entradas para algunos valores de la clave de indexación.
- Tipos de índices ordenados de un nivel:

    - primario o principal;
    - agrupado;
    - secundario.

- Índices multinivel:

    - índices multinivel;
    - índices multinivel dinámicos que utilizan árboles B y B+.

Tema 2. Índices primarios y secundarios

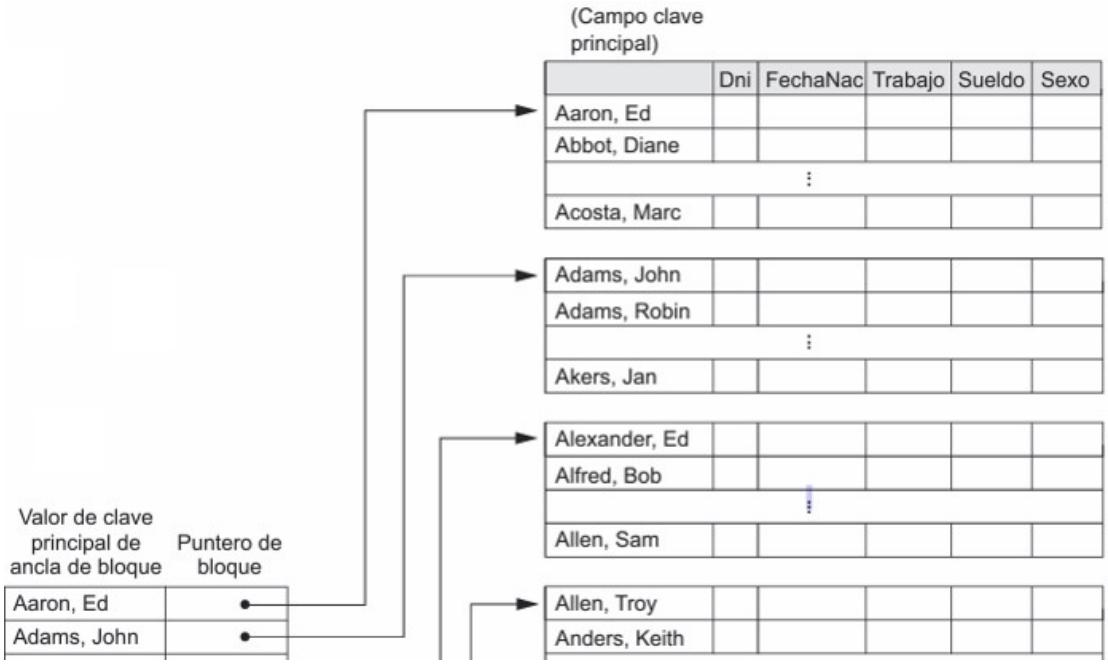
Índice primario

En estos índices, es necesario que el archivo de datos esté ordenado (físicamente) por campo clave, y que no contenga valores repetidos. El archivo de índice tiene dos columnas: la primera contiene el valor del campo clave del primer registro de cada bloque, y la segunda un puntero al bloque de datos «puro».

Los índices primarios son no densos, es decir, no se cargan todas las claves, ocupan menos espacio que el archivo de datos y poseen registros de longitud fija.

Se utiliza la búsqueda binaria porque el archivo es ordenado, y luego la lectura del bloque con el dato de interés. La inserción de datos es un problema, debido a que puede cambiar registros anclas o usar desbordamiento. Para el borrado, se utiliza un marcador en lugar de eliminar el dato físicamente.

Figura 7: Índice primario



Fuente: [Imagen sin título sobre índice]. (s.f.).

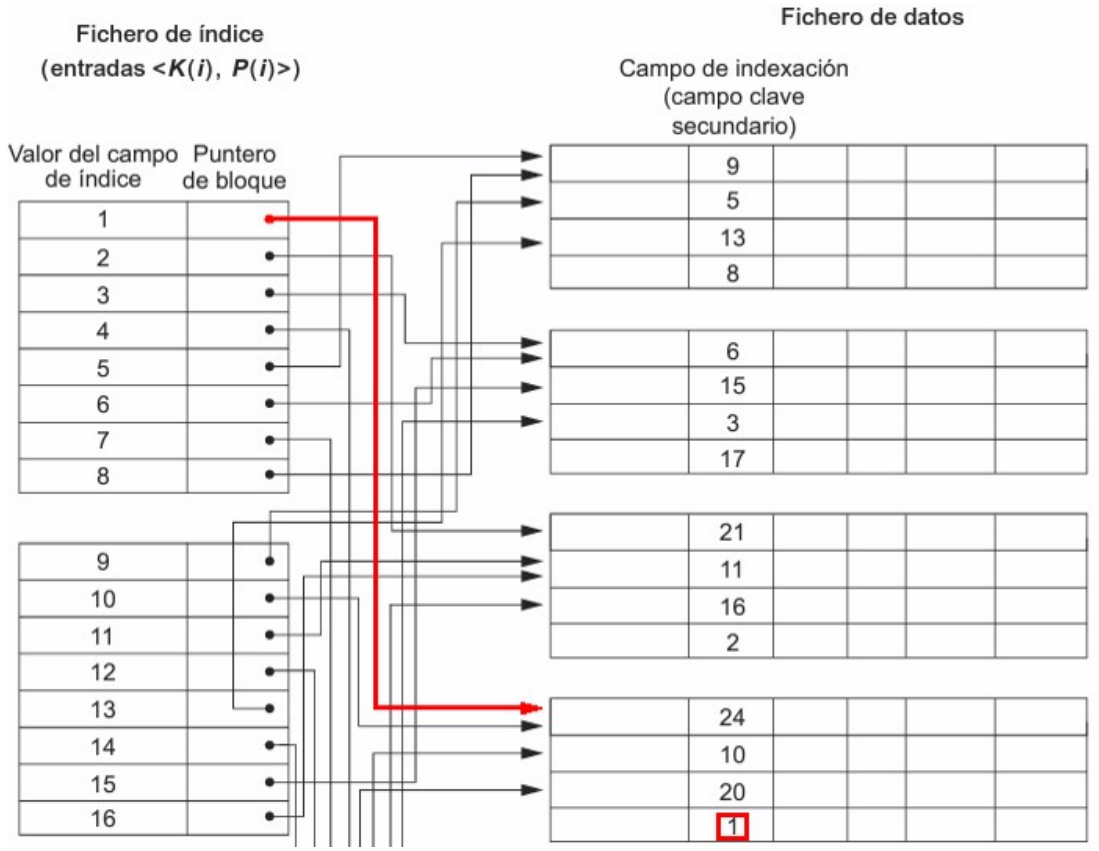
Índice secundario

Los índices secundarios se usan cuando un archivo ya posee un índice. Se generan por la necesidad de búsquedas sobre campos que pueden o no ser primarios, pero es necesario que no existan valores duplicados (únicos). Pueden existir muchos índices secundarios por archivo de datos, y proporcionan orden «lógico» a los registros almacenados.

El archivo índice consta de dos columnas: la primera es el campo de indexación, y la segunda un puntero al bloque o al registro. Si el índice es por clave candidata, hay una entrada en el índice por cada registro en el archivo de datos (índice denso), pero sigue siendo más pequeño que el archivo de datos. Es lento para la búsqueda y crece proporcionalmente. Si el índice es por campo no clave, los valores del campo de indexación se repiten. Ante esto, se puede generar lo siguiente:

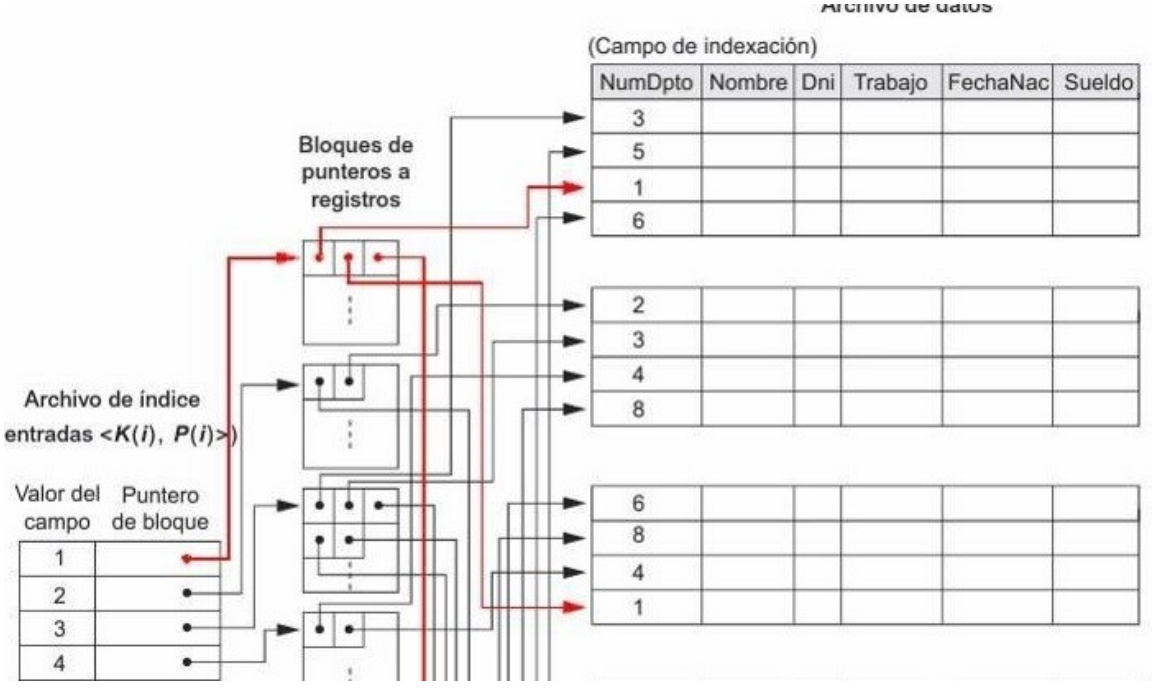
- un índice denso con un registro por cada valor, incluso repetido;
- un índice con registro de longitud variable con un campo repetitivo de punteros a bloque;
- un índice no denso, es decir, cada registro almacena puntero a bloque de punteros, y estos apuntan a registros (usado).

Figura 8: Índice secundario, puntero a registro



Fuente: [Imagen sin título sobre índice]. (s.f.).

Figura 9: Índice secundario, puntero a bloque de punteros



Fuente: [Imagen sin título sobre índice]. (s.f.).

Actividad de repaso

¿Cuál de las siguientes es una utilidad de los archivos de índices?

Ocupar espacio extra.

Realizar accesos eficientes al disco.

Hacer eficiente el uso de la memoria RAM.

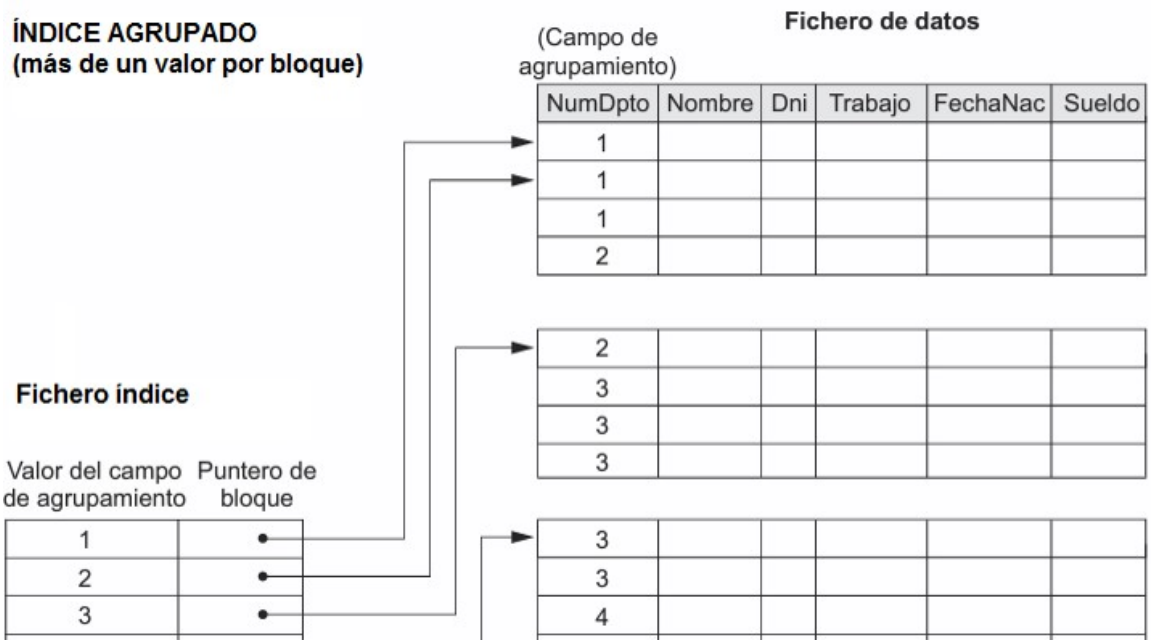
Justificación

Tema 3. Índices agrupados y multinivel

Índice agrupado

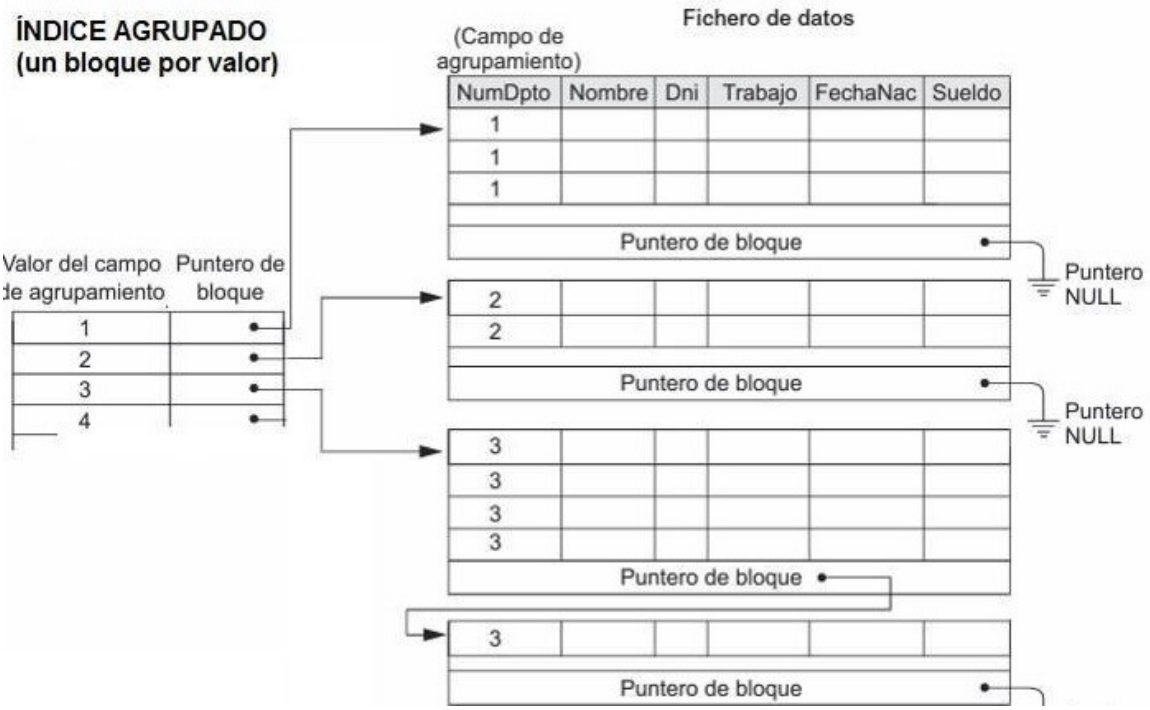
- Precondición: archivo de datos ordenado (físico) por campo no clave.
- El fichero índice tiene dos columnas:
    - contiene un registro por cada valor distinto del campo agrupado;
    - puntero al primer bloque que contiene un registro con ese valor.
- Es un índice no denso.
- Ocupa menos espacio que el archivo de datos; registro de longitud fija.
- Tiene más registros del archivo índice en cada bloque.
- Permite una búsqueda binaria en el archivo índice y lectura del bloque.
- Actualización:
    - inserción y borrado son un problema. Se sugiere reservar un bloque o contiguos por cada valor distinto del campo de indexación.

Figura 10: Índice agrupado, más de un valor por bloque



Fuente: [Imagen sin título sobre índice]. (s.f.).

Figura 11: Índice agrupado, bloque por valor



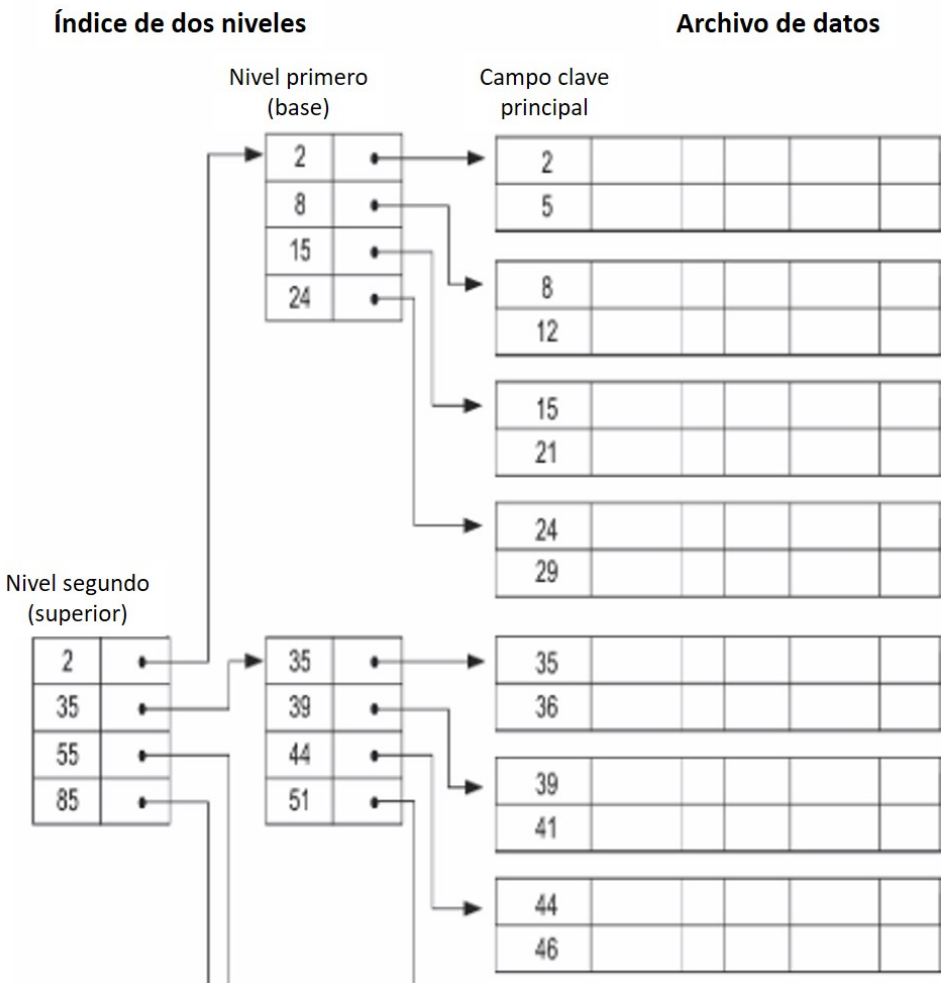
Fuente: [Imagen sin título sobre índice]. (s.f.).

Índice multinivel

- Se aplica después de generar el índice principal, agrupado o secundario.
- Ese índice de primer nivel es sobre campo clave, valores no repetidos y entradas de longitud fija.

- Genera el segundo nivel principal, y toma el primer nivel como fichero ordenado con valor distinto en clave.
- El segundo nivel posee una entrada por bloque.
- Genera el tercer nivel, si el segundo nivel ocupa más de un bloque.
- Lo anterior ocurre sucesivamente hasta que las entradas de un mismo nivel se almacenan en un bloque.

Figura 12: Índices multiniveles



Fuente: [Imagen sin título sobre índice]. (s.f.).

Tema 4. Árboles AVL, B y B+

¿Sabés lo que es un árbol? A continuación, te contamos de qué se trata:

Estructuras de datos

Fuente: Makigas.es [Makigas: tutoriales de programación]. (2016). Estructuras de datos – 11. Introducción a los árboles [YouTube]. Recuperado de https://www.youtube.com/watch?v=Qexq1k8LB6k.

¡Tomate unos minutos para ver video con atención!

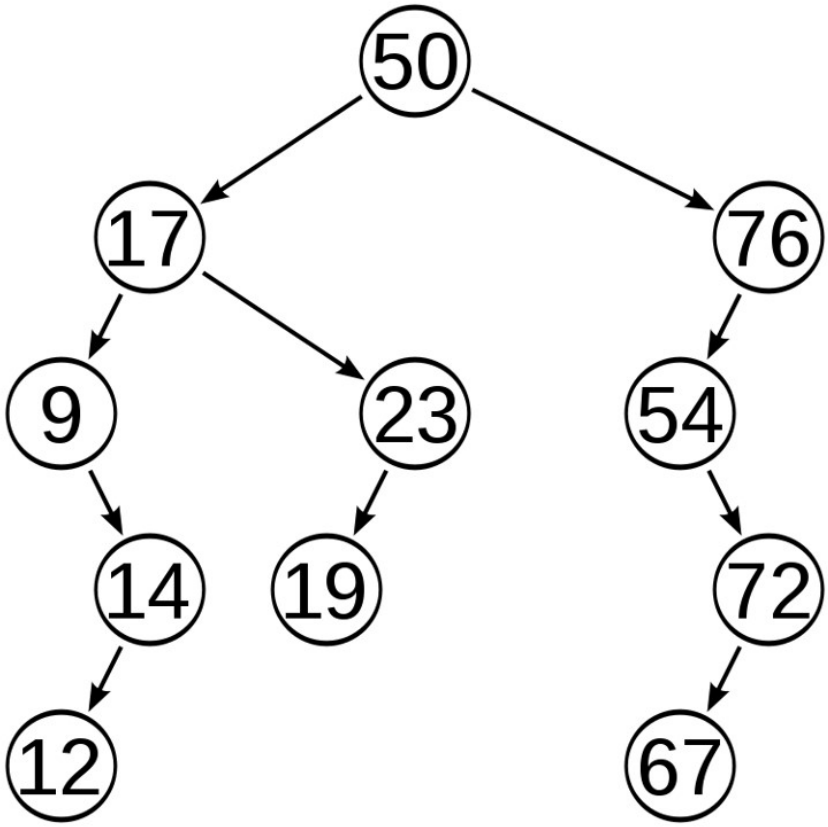
Árboles AVL

Siguiendo a Ponce (2010):

> Los llamados árboles AVL no son perfectamente equilibrados, pero son lo suficiente como para ofrecer un comportamiento bastante bueno al ser usados. La rama derecha, por lo general, no difiere en altura en más de una unidad de la rama izquierda o viceversa.
>
> El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles. Si, al realizar una operación de inserción o borrado, se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos para volver a equilibrarlo (https://bit.ly/3FCffc1).

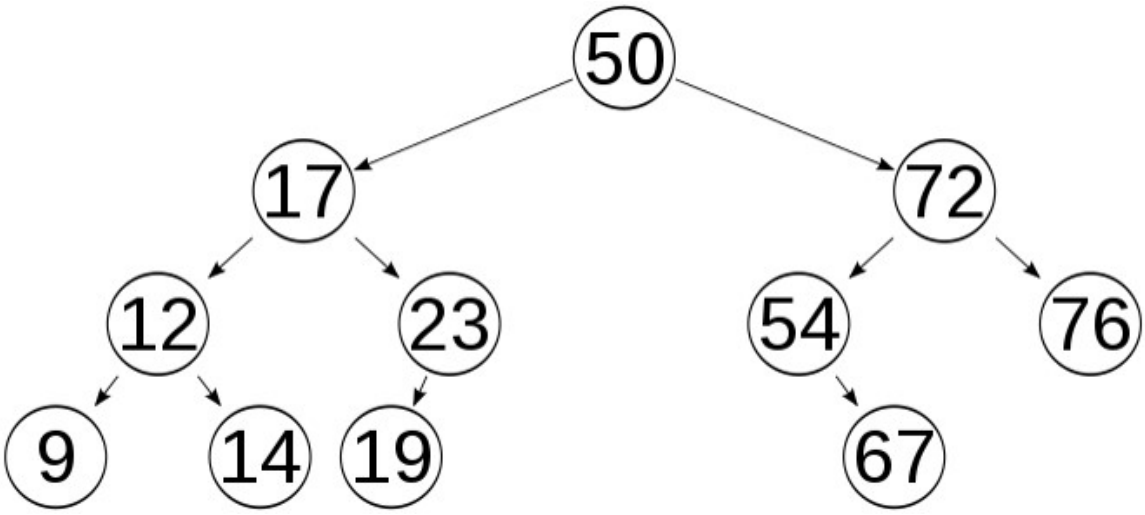
El árbol **AVL fue** creado por los matemáticos rusos Adelson-Velskii y Landis, y es un tipo especial de árbol binario.

Figura 13: Árbol AVL no equilibrado



Fuente: [imagen sin título sobre árbol AVL no equilibrado], 2013, https://bit.ly/39ydxqA.

Figura 14: Árbol AVL equilibrado



Fuente: [imagen sin título sobre árbol AVL equilibrado], 2013, https://bit.ly/3l2IpHB.

Árboles B y B+

Los índices multinivel dinámicos, que utilizan árboles B y B+, son casos especiales de estructuras dinámicas de datos. Permiten implementar índices multinivel, con la posibilidad de que un índice se expanda o contraiga. Cada nodo contiene los siguientes elementos:

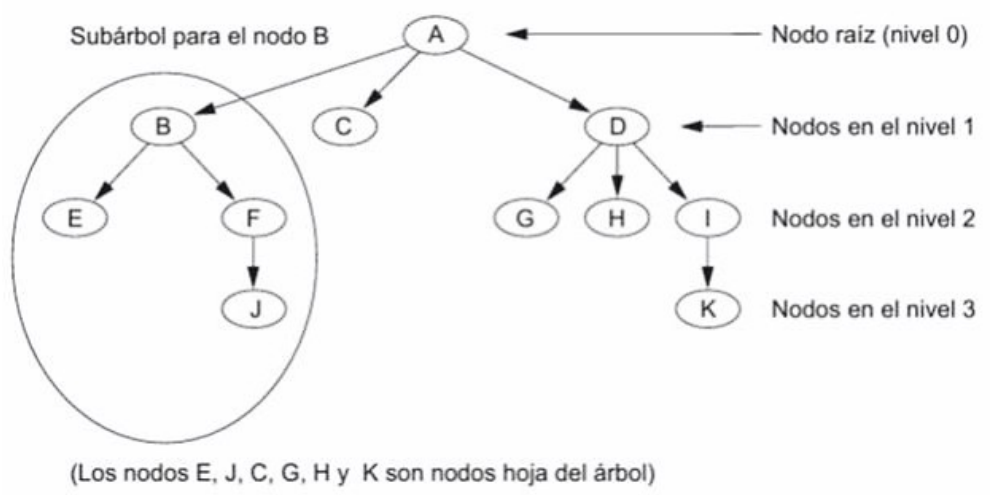
1. un puntero por cada nodo hijo que tiene;
2. valor del campo de indexación.

De acuerdo con Rodríguez-Tastets (2017):

> Las dos limitaciones que tienen los árboles de búsqueda, siendo K un valor de búsqueda de conjunto ordenado, son:
>
> - dentro de cada nodo: K1<K2<…<Kq-1;
> - para todos los valores de X del subárbol al que apunta Pi, tenemos que Ki-1<X<Ki. (http://bit.ly/38NrxwK).

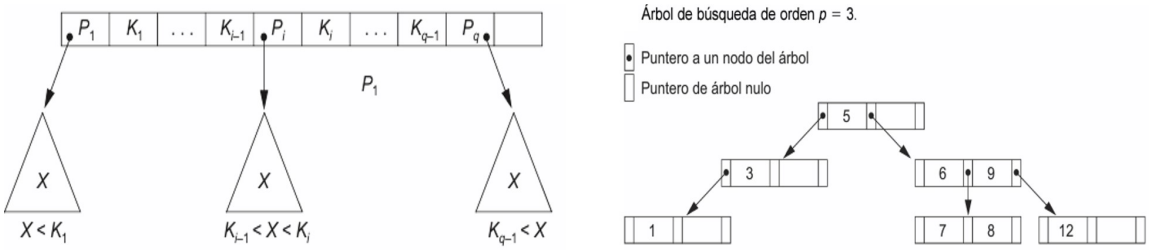
El puntero señala a un registro o a un bloque. Los algoritmos no aseguran equilibrio ni una velocidad de búsqueda no uniforme.

Figura 15: Estructura de datos de árbol desequilibrado



Fuente: [Imagen sin título sobre índice]. (s.f.).

Figura 16: Restricción de búsqueda 1 y 2



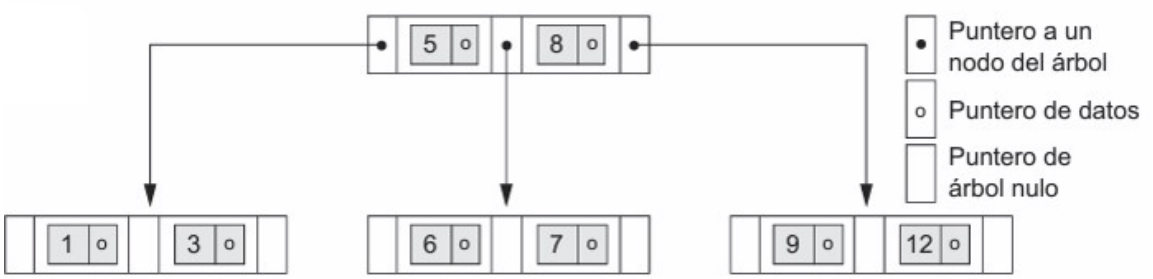
Fuente: [Imagen sin título sobre índice]. (s.f.).

Árboles B

Según lo establece la Escuela Universitaria de Informática (2017):

> Los algoritmos de los árboles B son complejos; cada valor del campo de búsqueda aparece con su puntero, por única vez. Hay otras restricciones adicionales que aseguran el equilibrio:
>
> 1. cada nodo interno es de la forma <P1,,P2,,…,Pq-1,,Pq > donde q≤p, cada Pi es un puntero de árbol (a otro nodo del árbol) y cada Pi es un puntero a un registro cuyo valor de clave sea Ki;
> 2. en cada nodo K1 < K2 < … <Kq-1;
> 3. para todo valor X en el subárbol apuntado por Pi, Ki-1 <X < Ki;
> 4. cada nodo tiene como máximo p punteros de árbol y p-1 pares;
> 5. cada nodo interno (excepto el raíz) tiene como mínimo p/2 punteros;
> 6. un nodo con q punteros de árbol tiene q-1 valores de campo clave;
> 7. todos los nodos hoja (punteros a árbol nulos) están al mismo nivel (https://bit.ly/3lrS62C)

Figura 17: Árboles B balanceados



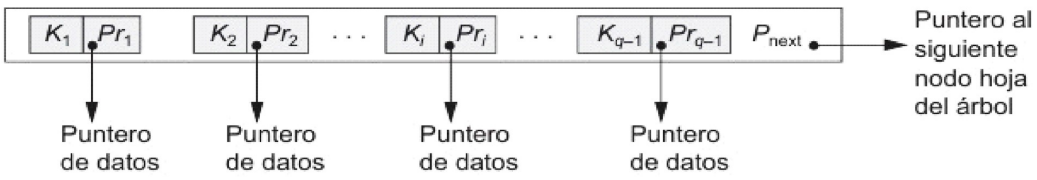
Fuente: [Imagen sin título sobre índice]. (s.f.).

Los árboles B+ son una variación de los árboles B. Los punteros a datos solo están en los nodos hoja, con lo cual se logran menos niveles y mayor capacidad. Los punteros de los nodos hoja pueden ser los siguientes:

- de registro, si es campo clave;
- de bloque, que contiene el registro con indirección, si no es campo clave.

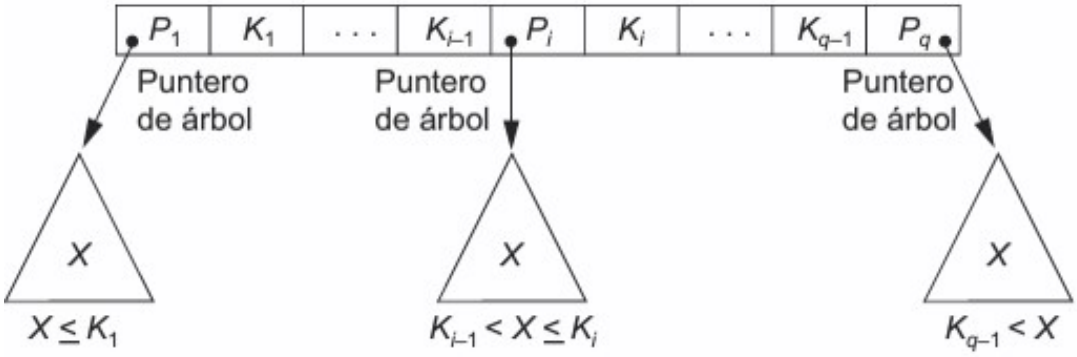
Los punteros en nodos internos son punteros de árbol a bloque. Los nodos hoja se encadenan para obtener acceso ordenado a registros.

Figura 18: Bloque con puntero al árbol



Fuente: [Imagen sin título sobre índice]. (s.f.).

Figura 19: Puntero al nodo hoja



Fuente: [Imagen sin título sobre índice]. (s.f.).

Video de habilidades

Video: Manejo de colisiones > Desde el minuto 13:06 hasta el final

Ubicación: después de Direccionamiento abierto (página 12)

Texto explicativo: En el video podemos apreciar como es la lógica de asignación de una clave en una tabla *hash*, ver qué sucede cuando ocurre una colisión y como se soluciona.

También entenderemos porque la búsqueda se realiza de manera más rápida.

Las tablas hash asocian valores con punteros

- Verdadero.
- Falso.

Justificación

¿Cuáles de las siguientes son operaciones básicas de las tablas hash?

- Modificación.
- Inserción.
- Sustitución.
- Eliminación.
- Concatenación.

Justificación

¿Cuáles de los siguientes son elementos que componen la tabla hash?

- Llaves o Claves.
- Operadores.

Clúster.

Registro.

Grupos de bloques.

Justificación

Los registros almacenados en una casilla del array guardan el índice

Verdadero.

Falso.

Justificación

¿Qué ocurre si a dos llaves o claves distintas se les asigna el mismo índice?

Una colisión.

Una inserción.

Una modificación.

Una eliminación.

Una actualización.

Justificación

Cierre

Aprendimos qué es un índice, los diferentes tipos de índices (primarios, secundarios, agrupados y multinivel), archivos de índices y, por último, las estructuras de árbol, como son los árboles AVL, B y B+.

Para saber más sobre los diferentes tipos de árboles y su uso, revisá los siguientes enlaces:

- Fuente: **Estructurasite**, (2016a). Árboles. Estructura de datos I. Recuperado de https://estructurasite.wordpress.com/arbol/.
- Fuente: **Estructurasite**, (2016b). Árbol. Teoría general de árboles. Recuperado de https://estructurasite.wordpress.com/arbol/.

Glosario

Referencias

[Imagen sin título sobre árbol AVL equilibrado], (2013). Recuperado de https://estructuras-lineal.blogspot.com/.

[Imagen sin título sobre árbol AVL no equilibrado], (2013). Recuperado de https://estructuras-lineal.blogspot.com/.

[Imagen sin título sobre direccionamiento abierto], (s. f.). Recuperado de https://commons.wikimedia.org/wiki/File:Tabla_hash2.png.

[Imagen sin título sobre encadenamiento], (s. f.). Recuperado de https://sites.google.com/a/espe.edu.ec/programacion-ii/home/tablas-hash.

[Imagen sin título sobre tabla hash], (s. f.). Recuperado de https://upload.wikimedia.org/wikipedia/commons/2/2f/Hash_table-es.svg.

Escuela Universitaria de Informática, (2017). Organización física de las bases de datos. Recuperado de http://web.archive.org/web/20100107094502/http://users.dsic.upv.es/~jcarlos/docente/bda/tema3_4p.pdf

Estructurasite, (2016a). Árboles. Estructura de datos I. Recuperado de https://estructurasite.wordpress.com/arbol/.

Estructurasite, (2016b). Árbol. Teoría general de árboles. Recuperado de https://estructurasite.wordpress.com/arbol/.

Global Sign. (2016) ¿Cómo funcionan las firmas digitales? Recuperado de

https://www.globalsign.com/es/blog/como-funcionan-las-firmas-digitales/.

Gómez, E. (2017). Estructuras de datos en Java. Recuperado de https://sites.google.com/a/espe.edu.ec/programacion-ii/home/tablas-hash.

Makigas.es [Makigas: tutoriales de programación]. (2016). *Estructuras de datos – 11. Introducción a los árboles* [YouTube]. Recuperado de https://www.youtube.com/watch?v=Qexq1k8LB6k.

Mauricio Avilés [Mauricio Avilés]. (2013). ED - Tablas Hash [YouTube]. Recuperado de https://www.youtube.com/watch?v=rCWVYKI8h6M&t=786s.

Ponce, L. (2010). Árbol AVL. Estudios. Ingeniería en Sistemas y Ciencias de la Computación. Recuperado de http://ponce-sistemas.blogspot.com/2010/08/arbol-avl.html.

Rodríguez-Tastets, M. A. (2017). Modelo físico. Recuperado de https://docplayer.es/57562461-Modelo-fisico-m-andrea-rodriguez-tastets-ii-semestre-universidad-de-concepcion-chile-andrea.html.