



*Universidad Nacional de la Patagonia San Juan Bosco*  
*Facultad de Ingeniería*

Cátedra: **Bases de datos I**

**Organización de la Información**

- Estructuras de almacenamiento
  - En el nivel conceptual de las BD no se necesita conocer los detalles físicos de la implementación del sistema: almacenamiento.
  - Físicamente, las BD casi siempre se almacenan en medios de acceso directo → discos.
  - Objetivo prioritario de desempeño en sistemas de BD → reducir al mínimo el número de accesos a disco.
  - **Estructura de almacenamiento:** Cualquier organización de los datos en el disco.
  - Un buen sistema deberá poder utilizar varias **estructuras alternativas**, cada una es apropiada para un tipo distinto de acceso a datos y habrá que llegar a un compromiso entre la **eficiencia** de los distintos tipos de operaciones de acceso a datos.
  - La elección final de la estructura de datos se hará teniendo en cuenta el uso esperado del sistema y las características físicas de la máquina.
  - Veremos aspectos de las **estructuras de datos** que permitirán un acceso rápido a los datos teniendo en cuenta el rendimiento, que depende de la **eficiencia de la estructura de datos** que se haya utilizado para representar la información de la BD y la **eficiencia del sistema de archivos** para manejar estas estructuras de datos. También va a depender de la **velocidad y tipo de los dispositivos de almacenamiento**.

- Acceso a las Bases de Datos
  - Como parte de la implementación física de una BD se encuentran varios tipos de archivos:
    - ✓ **Archivos de datos:** que almacenan la BD
    - ✓ **Archivos del diccionario de datos:** con información acerca de la estructura de la BD
    - ✓ **Archivos de índices:** que permiten acceso rápido a los datos (de la BD y del diccionario)
  - **Necesidades básicas** que motivan la existencia de archivos:
    - ✓ Almacenar **gran cantidad de información** que no cabe en la memoria principal
    - ✓ **Persistencia** de la información más allá de distintas ejecuciones
    - ✓ Acceso a información de forma **concurrente**
  - El **sistema operativo**, mediante la gestión del **Sistema de Archivos**, es el encargado de mantener los distintos archivos de la BD.
  - Cada archivo se divide en unidades de almacenamiento de longitud fija llamadas **bloques**, que son las **unidades de asignación de almacenamiento y transferencia de datos**.

- Uno de los **principales objetivos** del sistema de BD es minimizar el número de transferencias de bloques entre el disco y la memoria.
- Una forma de lograr esto es mantener en la memoria todos los bloques que sea posible. ¿Es posible mantener en la memoria principal todos los bloques? NO → Gestionar la asignación del espacio disponible en la memoria principal para almacenar los bloques.
- Un **buffer** es la parte de la memoria disponible para el almacenamiento de las copias de los bloques. En disco siempre hay una copia del bloque, pero puede ser que tengamos una versión más antigua que la que tenemos en el buffer.
- Los programas de un SGBD formulan solicitudes (llamadas) al gestor de memoria intermedia cuando necesitan un bloque de disco:
  - ✓ **Si el bloque se encuentra en el buffer**, se pasa al solicitante la dirección del bloque en memoria principal.
  - ✓ **Si no se encuentra**, el gestor debe asignar un espacio en la memoria para el bloque → descartar otro bloque, éste se vuelve a escribir en disco sólo si se modificó. A continuación, el gestor lee el bloque del disco y lo escribe en el buffer, pasando la dirección del bloque en memoria principal al solicitante.

- Organización de archivos

- Los archivos se almacenan **lógicamente como secuencias de registros**. Estos registros se corresponden con los bloques del disco.
- Los SGBD se apoyan en el sistema de archivos subyacente. Aunque los bloques son de tamaño fijo, los registros varían. En las BDR las tuplas de las distintas relaciones suelen ser de distintos tamaños.
- Existen archivos con **registros de longitud fija**, los cuales son más sencillos de implementar que los **registros de longitud variable**.

- ✓ **Registros de longitud fija**

```
Type depósito = record  
Nombre_sucursal: char (22);  
Número_cuenta: char(10);  
Saldo: float;
```

- Si se supone que cada carácter ocupa 1 byte y el float ocupa 8 bytes, el registro *depósito* tiene 40 bytes de longitud.

- Un enfoque sencillo es usar los primeros 40 bytes para el primer registro, los 40 siguientes para el segundo etc. Sin embargo hay dos problemas:
  1. Luego de borrar un registro de esta estructura, se debe llenar el espacio con otro registro o tener algún modo de marcar los registros borrados para que puedan pasarse por alto.
  2. A menos que los tamaños de los bloques sean múltiplos de 40 (bastante improbable) algunos registros guardarán una parte en un bloque y parte en otro. Hará falta dos accesos para leer o escribir esos registros.
- Al **borrar un registro** se podrían desplazar todos los registros a un registro anterior → necesita desplazar gran cantidad de registros, resultaría más sencillo desplazar el último registro al lugar vacío. Es mejor opción aún dejar esos espacios marcados para insertar un nuevo registro.
- Al comienzo del archivo se asigna cierto número de bytes como cabecera que contendrá gran cantidad de información sobre el archivo. Una de las cosas que se puede guardar es la **dirección del primer registro cuyo contenido se haya borrado** (punteros). Se utiliza este primer registro para guardar la dirección del segundo registro disponible y así sucesivamente, formando una **lista enlazada**.

Cabecera				↙
Registro 0	Centro	c-102	800	
Registro 1				↙
	Norte	c-309	1400	
				↙
	Centro	c_110	500	↙
Registro 6				

- Al insertar un registro nuevo se utiliza el registro indicado por la cabecera. Se cambia el puntero de la cabecera para que apunte al siguiente registro disponible. Si no hay registro disponible se añade al final del archivo.
- **El uso de punteros exige mucho cuidado**, si se borra un registro, se deben actualizar los punteros de la lista enlazada.
- **La inserción y borrado de archivos de longitud fija son sencillos de implementar**, dado que el espacio libre es el mismo que se necesita para insertar otro registro.

- **Registros de longitud variable:** Surgen por varios motivos → almacenar varios tipos de registros en un mismo archivo, tipos de registros que permiten longitud variable en algún atributo, etc.

```
Type lista_cuentas = record
  Nombre_sucursal: char (22);
  Información_cuenta: array [1..inf] of
    record
      Número_cuenta: char(10);
      Saldo: float;
    end;
end;
```

- 1. Representación en cadena de bytes:** Se adjunta un símbolo especial de fin de registro ( $\perp$ ) al final de cada registro. Cada registro se puede guardar como cadenas de bytes consecutivas. Varios **inconvenientes** → No es sencillo volver a utilizar el espacio dejado por un registro borrado, se desaprovechan fragmentos pequeños de almacenamiento en disco. Por lo general no queda espacio para el aumento de tamaño, hay que desplazar todo el registro.

Centro	C-102	2000	c-201	500	c-567	1000	$\perp$
Norte	c-490	600	c-789	2000	$\perp$		
Sur	c-101	300	c-923	1800	$\perp$		



**2. Representación de longitud fija:** Se usan registros de longitud fija para representar los registros variables. Hay dos técnicas:

- ✓ **Espacio reservado:** Existe una longitud máxima que nunca se supera, entonces se usan registros fijos de esa longitud. El espacio no usado se rellena con un símbolo especial (P. e. el fin de cadena). Es **útil** si la mayoría de los registros tienen una longitud cercana a la máxima, sino se desperdicia mucho espacio.

Centro	C-102	2000	c-201	500	c-567	1000
Norte	c-490	600	c-789	2000	⊥	⊥
Sur	c-101	300	c-923	1800	⊥	⊥

- ✓ **Punteros:** Consiste en representar el registro de longitud variable mediante una lista de registros de longitud fija enlazado por punteros.

Cabecera

Registro 0

Registro 1

Registro 6

Centro	c-102	800	
Sur	c-309	1000	
	c-309	1400	
	c-980	2000	
	c-540	350	
Norte	c-110	500	
	c-789	1500	

- Organización de registros

- Hasta ahora hemos visto como representar un registro en la estructura de archivos, ahora veremos como **organizar los registros en archivos**. Muchos sistemas de BDR guardan cada relación en un archivo diferente. Generalmente las tuplas de una relación pueden representarse como registros de longitud fija, pero cuando las BD son muy grandes el rendimiento no es tan bueno y hay que buscar estructuras de organización alternativas.

- ✓ **Organización de archivos en montículo**
  - ✓ **Organización en archivos secuenciales**
  - ✓ **Organización asociativa (hash) de archivos**
  - ✓ **Organización de archivos en agrupaciones**

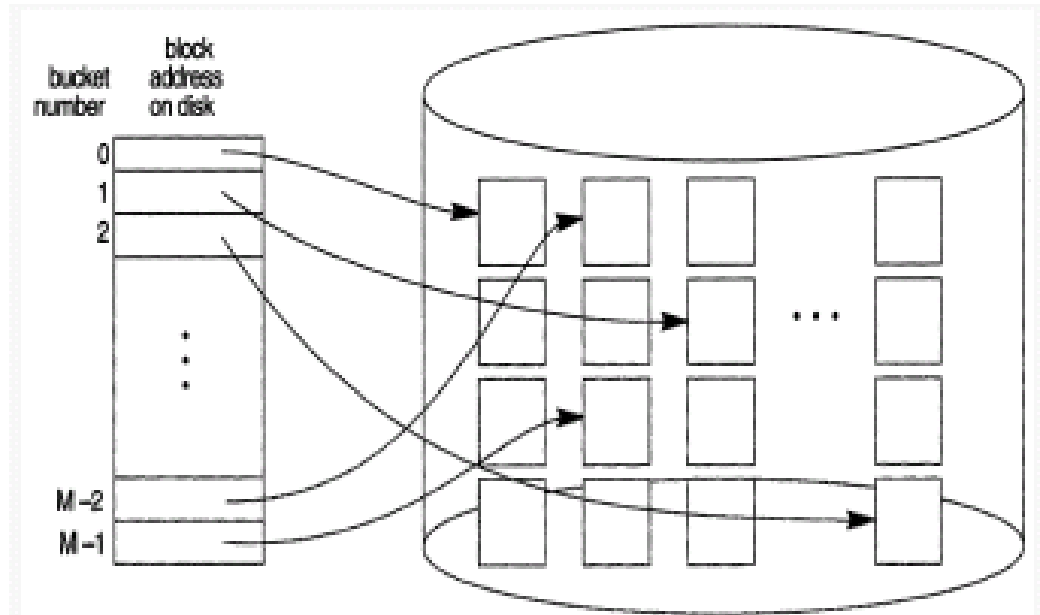
- ✓ **Organización de archivos en montículo:** Se puede colocar cualquier registro en cualquier parte del archivo en el que haya suficiente espacio. Generalmente hay un archivo por cada relación.
- ✓ **Organización en archivos secuenciales:** Los registros se guardan en un orden secuencial basado en el valor de la clave de búsqueda de cada registro. Para permitir la recuperación rápida, **los registros se vinculan mediante punteros**. El puntero de cada registro apunta al siguiente registro según el orden indicado por la clave de búsqueda (P. e. por nombre de sucursal: Centro, Norte, Sur). Para minimizar el número de accesos a disco, **los registros se guardan físicamente en el orden indicado**, o en un orden tan cercano a éste como sea posible. Pero es muy difícil mantener ese orden físico secuencial cuando se insertan y borran registros. Si hay que guardar un registro y no hay lugar se puede insertar en un bloque de desbordamiento.

Belgrano	c-102	800
Lamas	c-309	1000
Martinez	c-309	1400
Perez	c-480	2000
Perez	c-540	350
Sanchez	c-110	500
Sanchez	c-789	1500

--	--	--

- ✓ **Hash de archivos:** Se calcula una **función de asociación** (hash) de algún atributo de cada registro. Está relacionada con las estructuras para la creación de índices. Uno de los campos de la relación es la llave de hash del archivo. Un registro cuya llave de hash tiene el valor  $K$  es almacenada en el *bucket*  $i$ , donde  $i=h(K)$ , y  $h$  es la función de hashing. La función nos da la dirección donde debe guardarse el registro. La búsqueda en llave de hash es muy eficiente.



- ✓ **Organización de archivos en agrupaciones:** Se pueden guardar registros de varias relaciones diferentes en el mismo archivo. La ventaja se puede ver cuando se realizan consultas sobre una o mas relaciones. Muchos SGBD guardan cada relación en un archivo diferente, es factible en BD personales no en BD grandes.

- Estructuras de almacenamiento de BD Orientada a Objetos
  - Las técnicas que vimos pueden servir para guardar los objetos de las BDOO. Sin embargo, se necesitan **características adicionales**.
  - **Correspondencia de los objetos con los archivos:** tiene un gran parecido con las correspondencias de las tuplas con los archivos de los sistemas relacionales.
  - **Implementación de los IDOs:** vimos que los objetos se identifican por medio de los IDOs, por lo que los sistemas de almacenamiento de objetos necesitan un mecanismo para encontrar un objeto dado su IDO.
    - ✓ Si los **IDOs son lógicos**, es decir, no especifican la ubicación del objeto, el sistema de almacenamiento debe tener un índice que asocie los IDOs con la ubicación real.
    - ✓ Si los **IDOs son físicos**, es decir, tienen la dirección del objeto, se puede encontrar el objeto directamente. Los IDOs físicos suelen tener tres partes:
      1. Un identificador de archivo.
      2. Un identificador de página dentro del archivo.
      3. Un desplazamiento dentro de la página.

- Además los IDOs físicos pueden contener un identificador único, que distingue el IDO de un objeto de los IDOs de otros objetos que se hayan guardado anteriormente en esa ubicación y se hayan borrado o trasladado a otra parte.

Archivo	Página	Desplazamiento	Identificador único
---------	--------	----------------	---------------------

Identificador físico del objeto

Identificador único	Datos....
---------------------	-----------

Objeto

Identificador único	Datos	Ubicación
51	Datos....	6.32.45608
6.32.45608	51	IDO Adecuado
6.32.45608	50.	IDO Incorrecto

- El identificador único se guarda con el objeto y debe coincidir con los identificadores del IDO y del objeto correspondiente. Si el identificador único de un IDO físico no coincide con el del objeto al que apunta el IDO, el sistema detecta un **puntero colgante**.

- Objetos de gran tamaño

- Los objetos pueden ser muy grandes, p. e. los objetos multimedia pueden ocupar varios megabytes, o secuencias de video del orden de los gigabytes. Por lo tanto **suelen dividirse** en varios objetos más pequeños.
- Los objetos de gran tamaño se denominan **BLOBs** (binary large objects: objetos binarios de gran tamaño).
- La mayor parte de las **bases relacionales limitan el tamaño de los registros** (tuplas) para que no sean mayores que el tamaño de la página.
- Los objetos de gran tamaño suelen guardarse en un archivo o conjunto de archivos especiales → Se presenta un **problema** en la gestión de objetos de gran tamaño con la asignación de las **páginas de memoria intermedia**. Los objetos grandes pueden necesitar ser guardados en una secuencia contigua de bytes cuando se llevan a memoria, y si el objeto es mayor que una página, se deben asignar páginas contiguas de memoria para guardarlo.

- Indexación - Conceptos básicos




- Un índice para un archivo del sistema **funciona como el índice de un libro**. Si se desea leer un tema en particular se consulta el índice del libro para ver en que página comienza ese tema y así acceder directamente a la página correspondiente. Ahora, si no se utiliza el índice se deberá recorrer las páginas desde el principio del libro, para ver donde comienza el tema buscado (búsqueda secuencial).
- Con las consultas a las tablas de la BD sucede lo mismo. Si la tabla sobre la que se realiza una consulta no posee ningún índice, el SGBD se ve obligado, por cada consulta que se realice sobre la tabla, a recorrer todos los datos secuencialmente, si la tabla posee un índice se minimizan las operaciones de lectura sobre la misma.
- Un **índice es un archivo auxiliar** que hace más eficiente la búsqueda de un registro en un archivo de datos.
- Los registros de un índice son de la forma **<valor del campo, puntero al registro>** y se encuentran ordenados por el valor del campo. Un índice usualmente ocupa considerablemente menos bloques en disco porque sus registros son mas pequeños.



- Técnicas de indexación y asociación
  - Existen varias técnicas, ninguna de ellas es la mejor, cada una es la más apropiada para una aplicación específica de BD. Cada técnica debe ser valorada según los siguientes criterios:
  - ✓ **Tipos de acceso:** Podrían incluir la búsqueda de registros con un valor concreto en un atributo, o buscar los registros cuyo atributos contengan valores en un rango especificado.
  - ✓ **Tiempo de acceso:** Tiempo que se tarda en buscar un determinado elemento de datos, o conjunto, usando la técnica en cuestión.
  - ✓ **Tiempo de inserción:** Tiempo empleado en insertar un nuevo elemento de datos. Este valor incluye el tiempo empleado en buscar el lugar apropiado donde insertar el nuevo elemento de datos, así como el tiempo empleado en actualizar la estructura del índice.
  - ✓ **Tiempo borrado:** Tiempo empleado en borrar un elemento de datos. Este valor incluye el tiempo empleado en buscar el elemento a borrar, así como el tiempo empleado en actualizar la estructura del índice.
  - ✓ **Espacio adicional requerido:** Ocupado por la estructura del índice. Normalmente la cantidad necesaria de espacio adicional suele ser moderada, es razonable sacrificar el espacio para alcanzar un rendimiento mejor.

- Claves de búsqueda:

- Los **atributos o conjuntos de atributos usados para buscar** en un archivo se llaman **claves de búsqueda**. Esta definición de clave de búsqueda no tiene nada que ver con clave primaria, clave candidata y superclave. Pueden existir varias claves de búsqueda. Usando nuestro concepto de clave de búsqueda, vemos que si hay varios índices en un archivo, existirán varias claves de búsqueda.
- Un índice consta de registros (denominados entradas de índice) de la forma:

Clave de búsqueda	Puntero
123	
156	
235	

- Un registro índice o entrada del índice consiste en un valor de la clave de búsqueda y punteros a uno o más registros con ese valor de la clave de búsqueda
- Existen dos clases básicas de índices:
- ✓ **Índices ordenados**: las claves de búsqueda se almacenan de forma ordenada.
- ✓ **Índices asociativos**: las claves de búsqueda están distribuidas uniformemente en registros, el valor asignado a cada registro está determinado por una función de asociación (hash function).

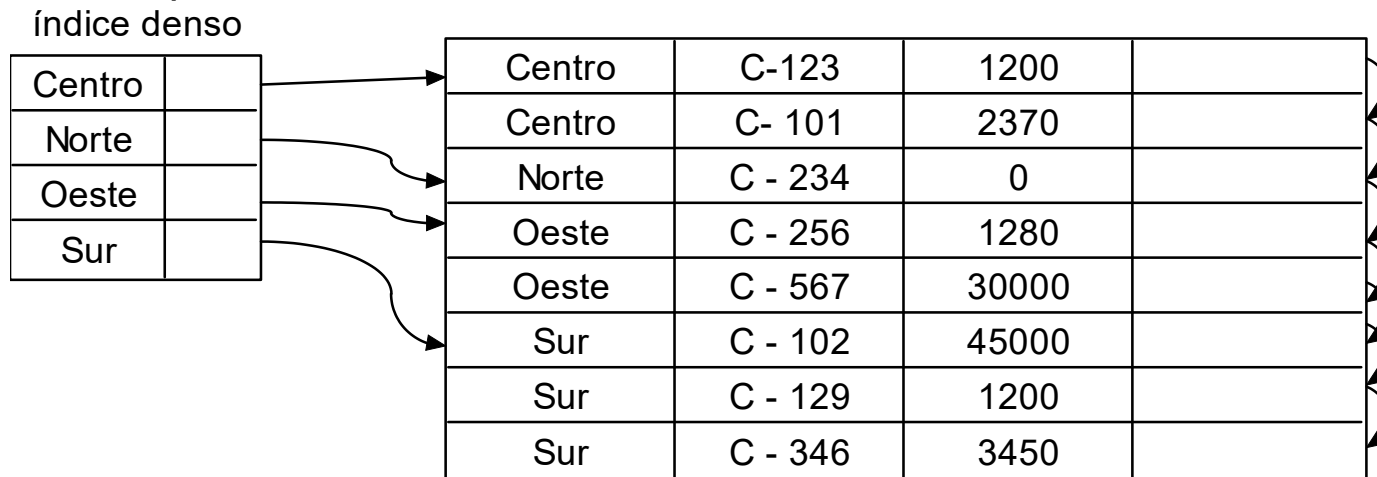
- Índices Ordenados: Índice primario y secundario
  - **Índice primario:** en un archivo con registros ordenados secuencialmente, el índice cuya clave de búsqueda tiene el mismo orden secuencial del archivo, se llama índice primario (índices con agrupación: clustering indexes). Algunas veces el término índice primario se emplea para hacer **alusión a un índice según la clave primaria** ya que la clave de búsqueda de un índice primario es normalmente la clave primaria.
  - Los archivos ordenados secuencialmente con índice primario según una clave de búsqueda se llaman **archivos secuenciales indexados**.
  - Siguiendo el orden de un índice primario, una búsqueda secuencial es eficiente porque los archivos están guardados físicamente de la misma manera que el índice.
  - Ejemplo de un archivo secuencial indexado:

Centro		Centro	C-123	1200	
Oeste		Centro	C- 101	2370	
Sur		Oeste	C - 234	0	
		Oeste	C - 256	1280	
		Oeste	C - 567	30000	
		Sur	C - 102	45000	
		Sur	C - 129	1200	
		Sur	C - 346	3450	

- Los índices cuyas claves de búsqueda especifican un orden diferente del orden secuencial del archivo se llaman **índices secundarios** (índices sin agrupación: non clustering indices). Un índice secundario sobre una clave candidata es como un índice primario, excepto en que los registros apuntados por los sucesivos valores del índice no están almacenados secuencialmente.
- Un **índice secundario** provee un medio secundario de acceso a un archivo para el que ya existe un acceso primario, puede definirse sobre un campo que sea o no clave candidata.

- **Índices ordenados: densos y dispersos**

- **Índice Denso:** Aparece un registro índice por cada valor de la clave de búsqueda en el archivo.



- **Índice Disperso:** Sólo se crea un registro índice para algunos de los valores de la clave de búsqueda

índice disperso



- Para localizar un registro en un **índice disperso**, se busca la entrada del índice con el valor más grande que sea menor o igual que el valor que se está buscando. Se empieza por el registro apuntado por esa entrada de índice y se continúa hasta encontrar el registro deseado.
- Para localizar un registro en un **índice denso**, se busca la entrada del índice con el valor igual al valor que se está buscando. Se empieza por el registro apuntado por esa entrada de índice y se continúa hasta encontrar el registro deseado.
- Generalmente es **más rápido** localizar un registro si se usa un **índice denso** en vez de un índice disperso. Sin embargo los **índices dispersos** tienen algunas **ventajas** sobre los índices densos, como el utilizar un **espacio más reducido** y un **mantenimiento adicional menor** para las inserciones y borrados.

- Creación de índices en SQL:

- Usamos la instrucción **CREATE** seguida por **INDEX**:

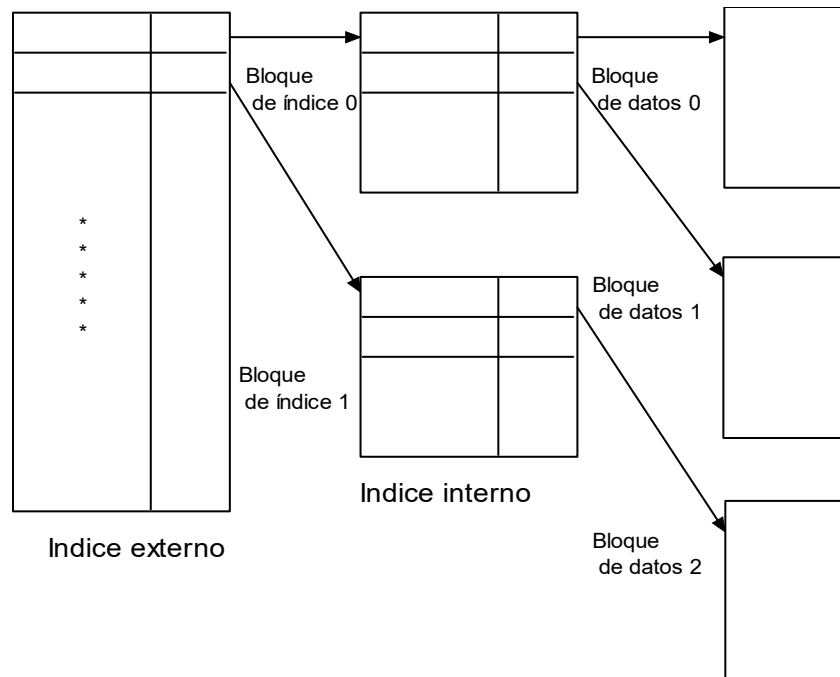
```
CREATE [UNIQUE] INDEX Nombre_Indice ON Nombre_Tabla  
[using nombre_acceso] (Nombre_Col_1,..., Nombre_Col_n)
```

- **UNIQUE**: provoca que el sistema compruebe si existen valores duplicados en la tabla cuando se crea el índice (si ya existen datos) y cada vez que se añaden datos → Los intentos de insertar o actualizar datos duplicados generarán un error.
- **Nombre\_Indice**: Nombre del índice que se debe crear.
- **Nombre\_Tabla**: Tabla para la que se quiere crear el índice.
- **nombre\_acceso**: Nombre del método de acceso que se utilizará para el índice. El método de acceso por defecto es BTREE.
- **Nombre\_Col\_1**: Nombre de una columna de la tabla.
- Para borrar un índice usamos la instrucción **DROP** con la siguiente sintaxis:

```
DROP INDEX Nombre_Indice
```

- Índices multinivel:

- Si un índice es lo bastante pequeño como para guardarlo completamente en la memoria principal, el tiempo de búsqueda para encontrar una entrada será breve.
- Sin embargo si el índice es tan grande que se debe guardar en disco, buscar una entrada implicará leer varios bloques de disco → el tiempo de búsqueda para encontrar la entrada puede ser muy costoso.
- Para resolver este problema, **se trata el índice como si fuese un archivo secuencial** (índice interno) **y se construye un índice disperso** (índice externo) sobre el índice primario.



- Para localizar un registro se usa en primer lugar una **búsqueda sobre el índice más externo**, buscando el registro con el mayor valor de la clave de búsqueda que sea menor o igual al valor deseado. **El puntero apunta a un bloque en el índice más interno**. Se examina este bloque hasta encontrar el registro con el mayor valor de la clave que sea menor o igual al valor deseado. El puntero de este registro apunta al bloque del archivo que contiene el registro buscado.
- Usando los dos niveles de indexación y con el índice más externo en memoria principal, tenemos que leer un único bloque índice. Si el archivo es extremadamente grande, incluso el índice exterior podría crecer demasiado para caber en la memoria principal. En este caso se podría crear otro nivel más de indexación → **Se podría repetir este proceso tantas veces como fuese necesario**. Los índices con dos o más niveles se llaman **índices multinivel**. La búsqueda de registros usando un índice multinivel necesita claramente menos operaciones de E/S. Cada nivel de índice se podría corresponder con una unidad del almacenamiento físico. Así, podríamos tener índices a nivel de pista, cilindro o disco.
- Los índices multinivel están estrechamente relacionados con la **estructura de árbol** que veremos a continuación.

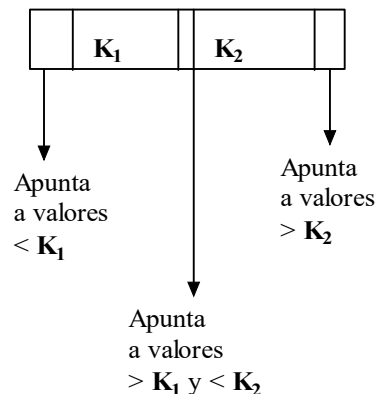


- Estructura de árbol B+ (1972)

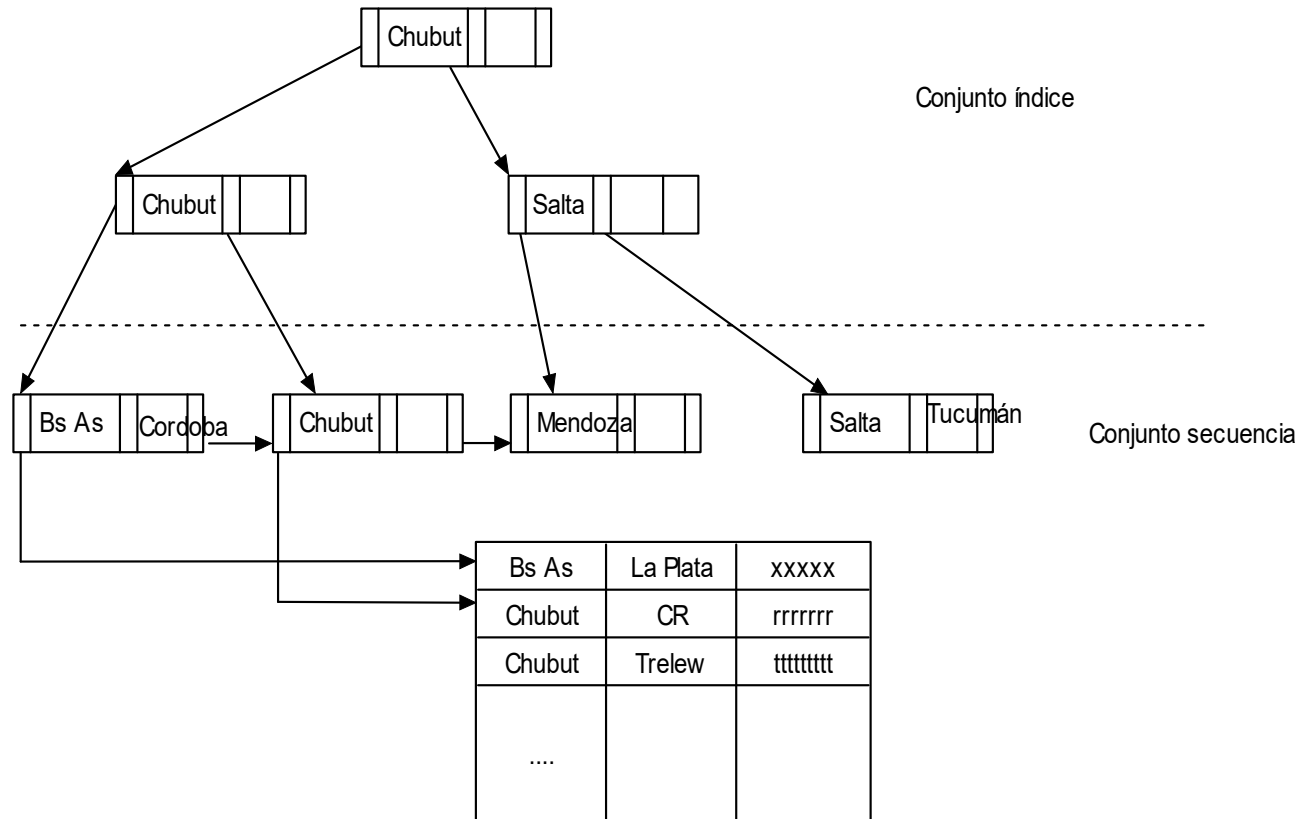
- **Un índice de árbol B+ es un índice multinivel.** En la figura se muestra un nodo típico de un árbol B+. Puede contener hasta  $n-1$  claves de búsqueda ( $K_i$ ) y  $n$  punteros ( $P_i$ ). Los valores de la clave de búsqueda de un nodo se mantienen ordenados, si  $i < j$ ,  $K_i < K_j$ .



- Consideremos la estructura de los **nodos hoja**. Para  $i=1, 2, \dots, n-1$  el puntero  $P_i$  apunta a un registro del archivo con el valor de clave de búsqueda  $K_i$ , o bien a otro nodo hoja con un puntero que apunta a un registro del archivo con valor de la clave de búsqueda  $K_i$ . Una propiedad importante es que un **árbol B+ debe estar equilibrado**, es decir la distancia desde la raíz a cada nodo hoja debe ser la misma, este requisito asegura buen rendimiento para búsquedas, inserciones y borrados → **B** es de **balanced**, balanceado.
- La composición de una hoja es la siguiente:



- El árbol B+ consta de dos partes:
- ✓ **El conjunto índice:** que proporciona un acceso directo y rápido al conjunto secuencia. Puede tener varios niveles.
- ✓ **El conjunto secuencia:** consiste en un índice denso de un solo nivel con punteros que apuntan a los datos.



- El conjunto índice es el árbol B+, en el ejemplo es de profundidad (i) 2 y n=3.

- **Consulta con árboles B+:** Supongamos que se desea encontrar todos los registros cuyo valor de clave de búsqueda sea Córdoba. Primero se examina el nodo raíz para buscar el menor valor de clave de búsqueda mayor que Córdoba. En nuestro caso es Chubut. Seguimos con el puntero hasta otro nodo, si la clave de búsqueda es mayor a Córdoba, seguimos por el puntero hasta otro nodo, así llegamos (en nuestro caso) al nodo del conjunto secuencia, donde encontramos la clave de búsqueda. El puntero apunta al registro correspondiente en el archivo.
- **Actualizaciones en árboles B+:** El borrado y la inserción son más complicados que las búsquedas, ya que puede ser necesario **dividir** un nodo que sea demasiado grande, como resultado de una inserción o **combinar** nodos si un nodo se volviera demasiado pequeño al borrar un valor. Además se debe asegurar que el árbol quede en **equilibrio** después de cualquier operación.
- ✓ **Inserción:** Usando la misma técnica que para buscar, se busca el nodo hoja donde tendría que aparecer el valor de la clave de búsqueda. Si el valor ya aparece en el nodo hoja, se inserta un nuevo registro (tupla) en el archivo. Si el valor de la clave de búsqueda no aparece, se inserta en el nodo de manera que las claves de búsqueda queden ordenadas. Luego insertamos el registro en el archivo.
- ✓ **Borrado:** Usando la misma técnica que para buscar, se busca el registro a borrar y se elimina del archivo. Si no queda ningún registro con el valor de la clave de búsqueda se borra éste del nodo hoja.

- Ventajas y Desventajas de los Índices B+
  - **Ventajas**
    - ✓ Se reorganiza automáticamente por sí mismo con pequeños cambios locales, a pesar de las inserciones y los borrados.
    - ✓ No es necesario reorganizar todo el archivo para mantener el rendimiento.
  - **Inconvenientes**
    - ✓ Inserciones extras
    - ✓ Sobrecarga de borrados
    - ✓ Gasto de espacio en disco mayor
  - Las ventajas de los árboles B+ superan a los inconvenientes, por lo que se emplean ampliamente → Es el estándar para la organización de archivos.
  - Los índices de árbol B+ son una alternativa a los archivos secuenciales indexados.