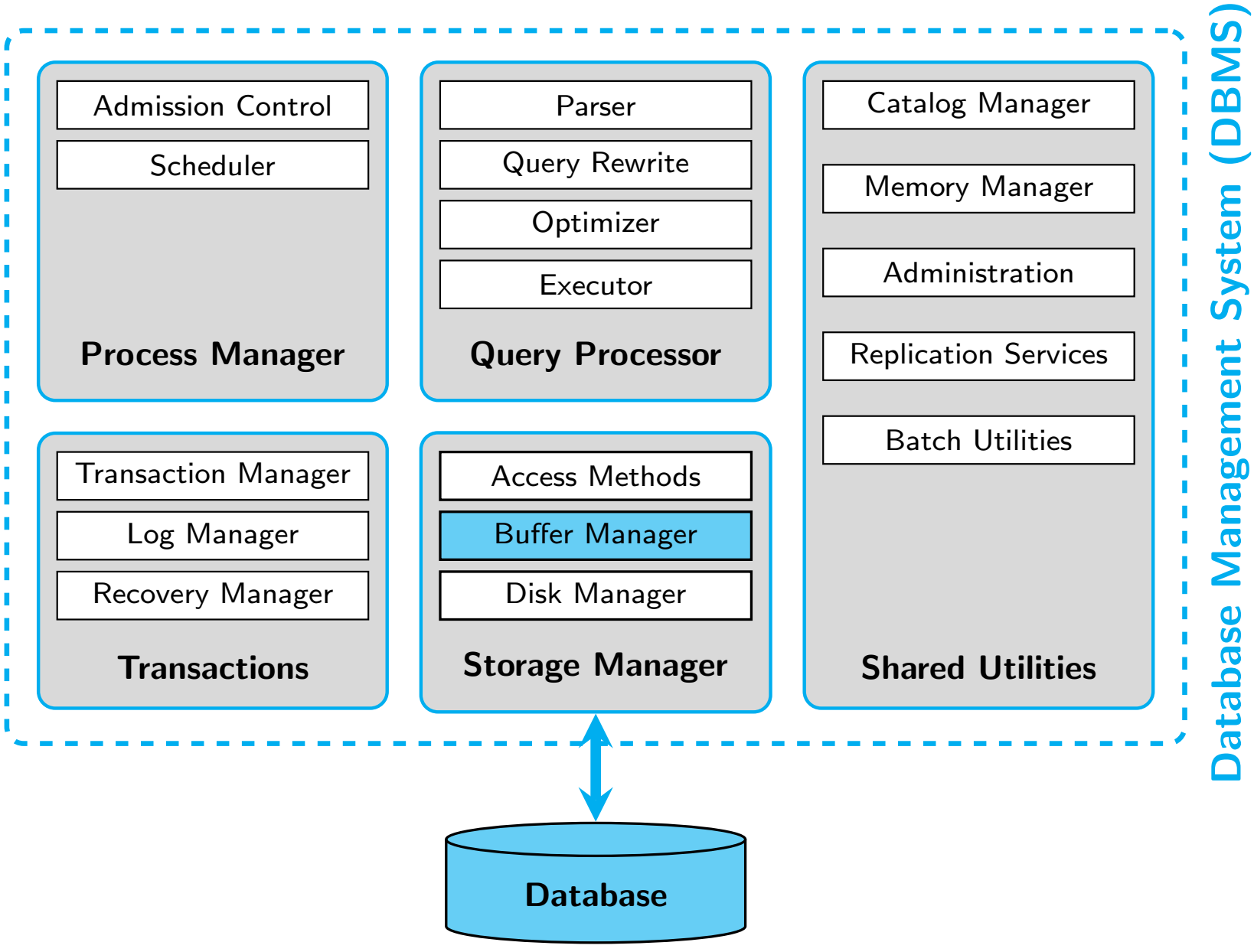
Organización de los datos

Clase 03

IIC 3413

Prof. Cristian Riveros

Organización de los datos



Considere el siguiente esquema relacional:

Players(pId, pName, pBirthdate, pDescription)
Matches(mId, mStadium, mDate)
Players_Matches(pId, mId, pGoals)

¿cómo almacenamos las tuplas de cada relación en disco?

¿cómo almacenamos las tuplas de cada relación en disco?

Posibilidades de diseño:

- Un archivo del SO para todas las relaciones (ej. SQLite).
- Un archivo del SO por cada relación.
- Varios archivos del SO por cada relación.

¿alguna desventaja?

Solución: almacenar cada relación
en un conjunto de **páginas** (bloques) de datos.

Almacenamiento basado en páginas

Una **página** corresponde a un **bloque** en disco.

Cada relación corresponde a un set de páginas que contienen un subconjunto de las tuplas.

Ventajas:

- Manejo granular del contenido en disco.
- Optimización del acceso al disco.
- Facilidad para el manejo de transacciónes.

Dado que almacenaremos nuestras relaciones en páginas,...

- ¿cómo almacenaremos las tuplas?
- ¿y las relaciones?

Heap files

Dado que el espacio en RAM es **muchisimo menor** que el espacio en disco:

- ¿cómo manejaremos la transferencia de datos entre el disco y la RAM?

Buffer manager

Outline

Heap files

Buffer Manager

Archivos vs. Memoria

Alternativas

Outline

Heapfiles: varias aristas del problema

1. ¿cómo representamos una tupla?
2. ¿cómo almacenamos varias tuplas en una página?
3. ¿cómo almacenamos una relación en varias páginas?

Representación de una tupla

Suponga que queremos guardar una tupla de la relación:

Relación:

Players(pId, pName, pBirthdate, pDescription)

Tuplas: (ejemplo)

(7, Carlos Caszely, 5/7/1950, Jugador conocido por perder ...)

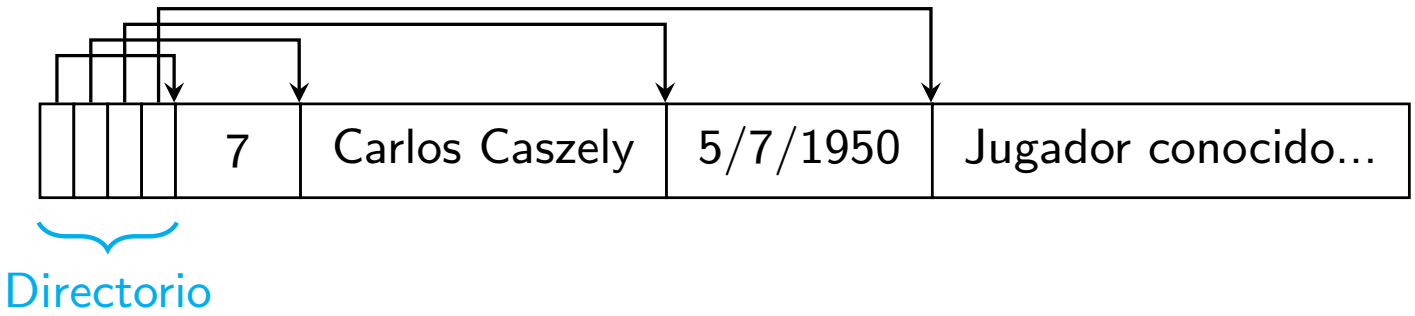
Primer approach:

#	7	$	Carlos Caszely	$	5/7/1950	$	Jugador conocido...	#

¿alguna desventaja?

Representación de una tupla

Segundo approach:

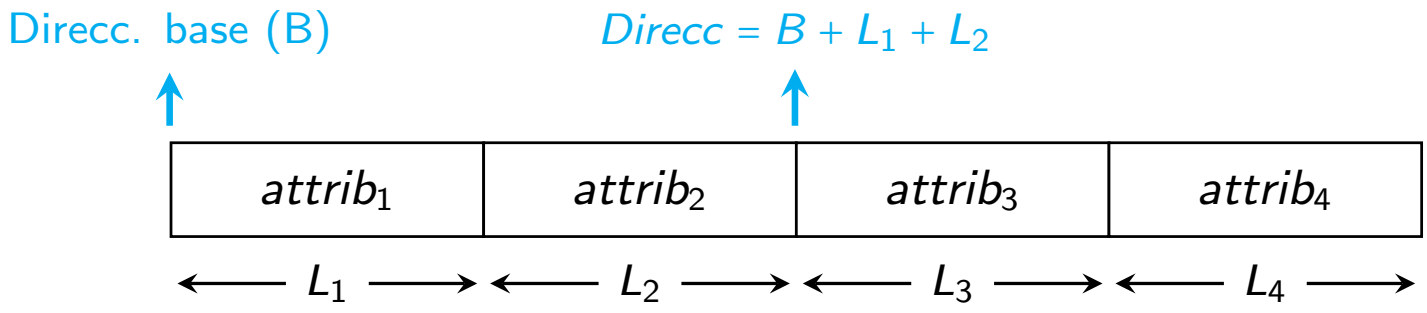


- ¿qué ocurre con los valores null?

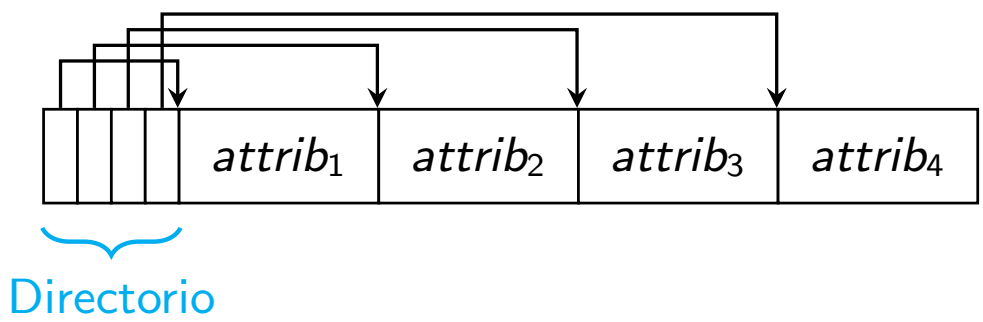
¿desventajas de este approach?

Representación de una tupla (diseño final)

Atributos de tamaño fijo:



Atributos de tamaño variable:



Heapfiles: varias aristas del problema

1. ¿cómo representamos una tupla? ✓
2. ¿cómo almacenamos varias tuplas en una página?
3. ¿cómo almacenamos una relación en varias páginas?

Record ID

- Identificador único de cada tupla.
- Necesario para mantener unicidad de cada tupla.

¿cúal es una buena elección de RID?

Usualmente, RID = $(PageID, NumSlot)$.

Tipo de almacenamiento de tuplas

1 página = 1 bloque en disco = tamaño fijo ($\approx$ 8 KB)

Tamaño de tuplas:

- Tamaño fijo.
- Tamaño variable.

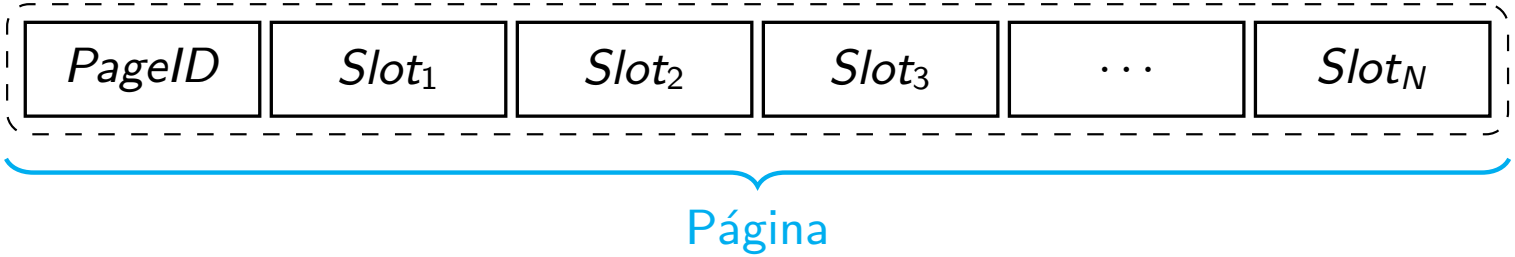
Atomicidad de una tupla:

- Cada tupla en una página (spanned).
- Una tupla en multiples páginas (unspanned).

Tipo de tuplas en una misma página:

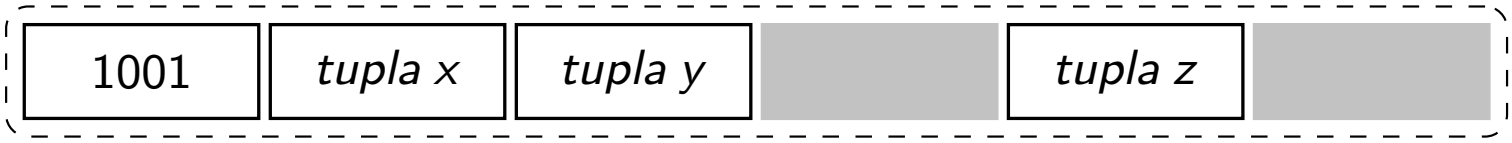
- Solo una relación (homogeneous).
- Multiples relaciones (non-homogeneous).

Formato página: tuplas de tamaño fijo



Record ID (RID) para una tupla en el $slot_n$ viene dado por: $(PageID, n)$.

Ejemplo página:

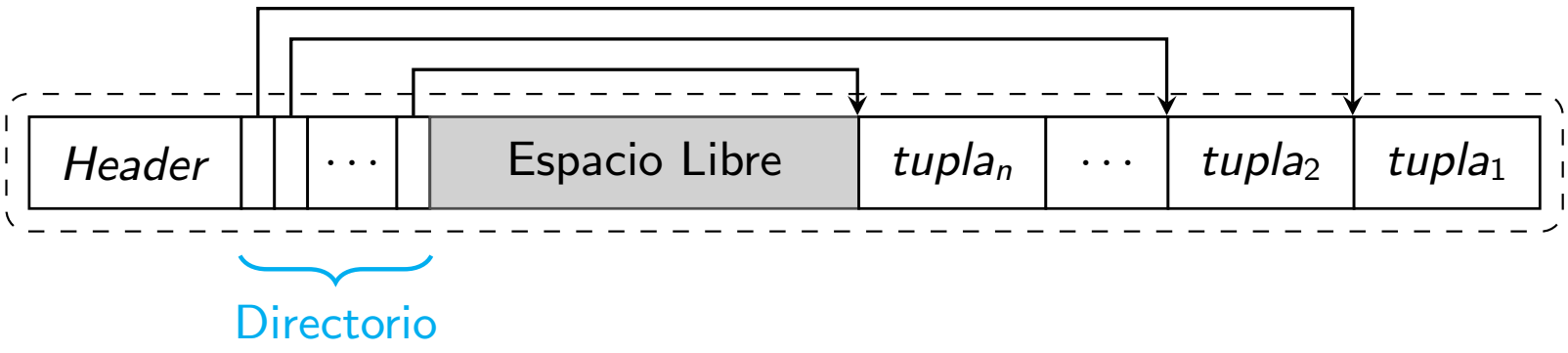


¿cómo insertamos/eliminamos una tupla en este formato?

Formato página: tuplas de tamaño variable

¿cómo almacenamos varias tuplas de tamaño variable en una página?

Formato página: tuplas de tamaño variable

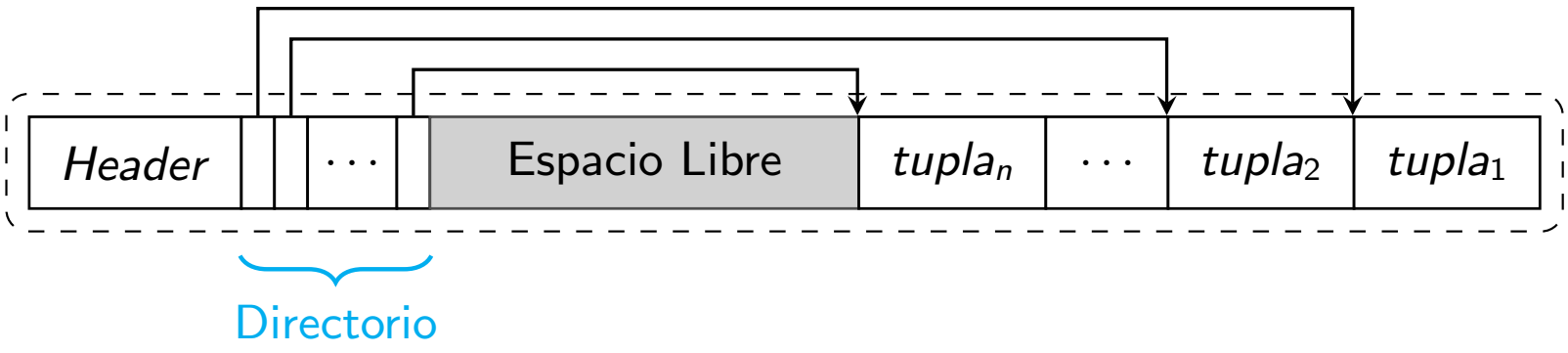


Header contiene el *PageID* + número de tuplas + tamaño directorio, etc.

Record ID (RID) = (*PageID*, *DirectoryID*).

¿cómo insertamos/eliminamos una tupla en este formato?

Formato página: tuplas de tamaño variable



¿Ventajas de este formato?

- Manejo de tuplas de tamaño variable.
- Eliminación de tuplas.
- Mover tuplas dentro de una página (indireccionamiento),
- o dentro de distintas páginas (forwarding address).

Formato página: tuplas de tamaño variable

Suposición del formato anterior:

Tuplas de tamaño variable
pero **no mayor** *al tamaño de una página.*

¿cómo almacenamos tuplas/atributos de tamaño mayor a una página?

`char(n)` vs. `varchar(n)` vs. `clob(n)`

BLOB = Binary Large OBject
CLOB = Character Large OBject

- Soportado por la mayoría de DBMS modernos.
- Incluyen: imagenes, videos, textos, etc.

¿cuál es la diferencia entre `char(n)`, `varchar(n)`, y `clob(n)`?

Implementación estandar para `clob`:

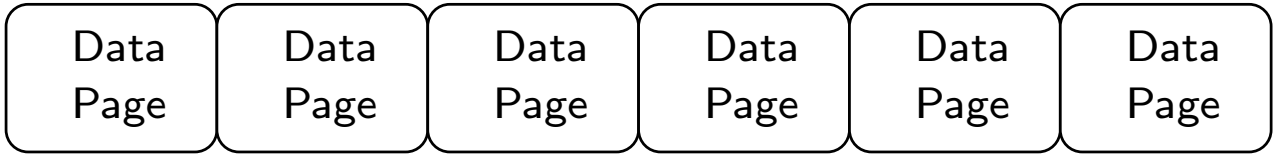
- Almacenar información del atributo en una o varias páginas.
- Guardar puntero a página en el atributo BLOB/CLOB.

Heapfiles: varias aristas del problema

1. ¿cómo representamos una tupla? ✓
2. ¿cómo almacenamos varias tuplas en una página? ✓
3. ¿cómo almacenamos una relación en varias páginas?

Heap file: Implementación 1

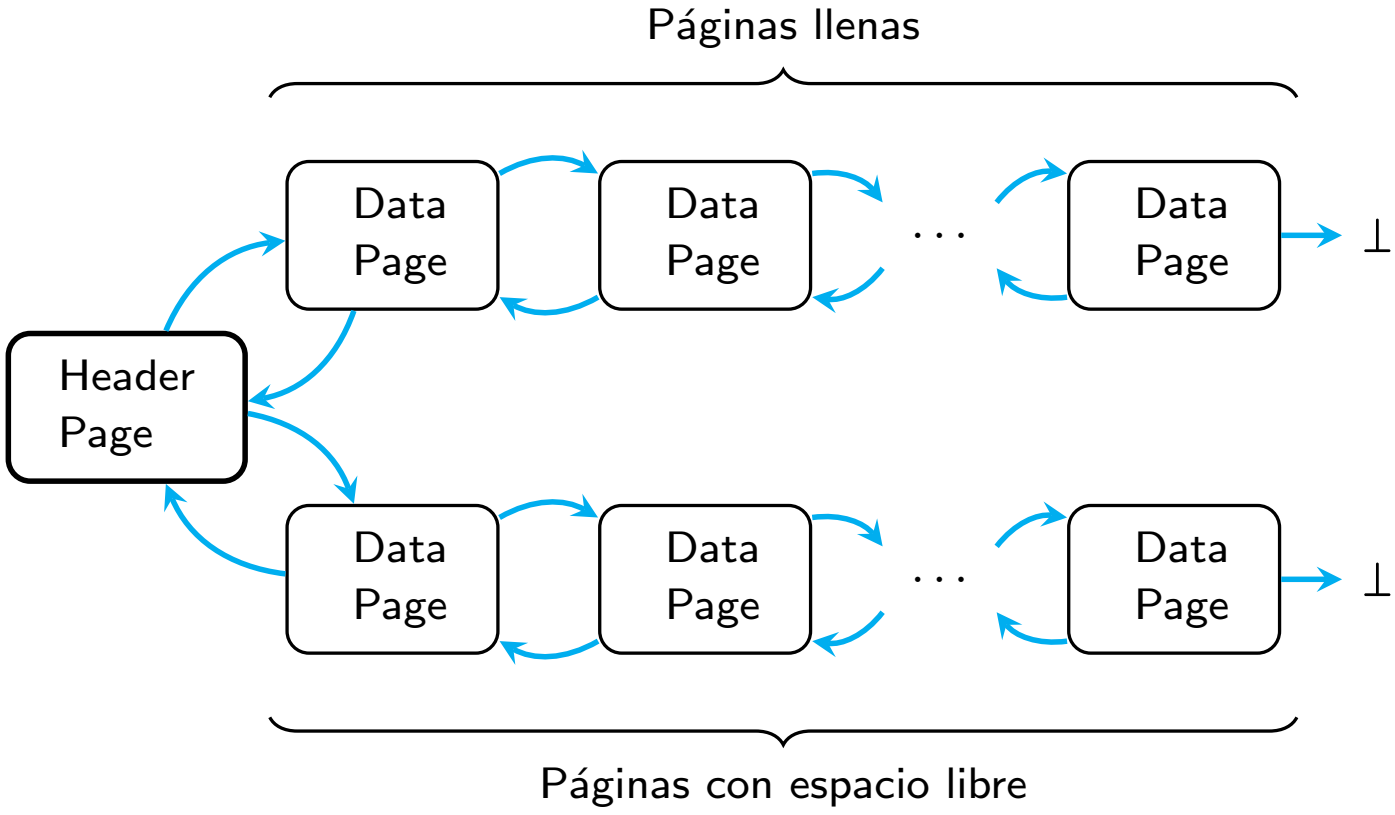
Páginas consecutivas:



¿Problemas?:

- Fragmentación de espacio en las páginas.
- Es necesario buscar en todas las páginas para encontrar espacio.
- Requiere un espacio contiguo en disco.

Heap file: Implementación 2

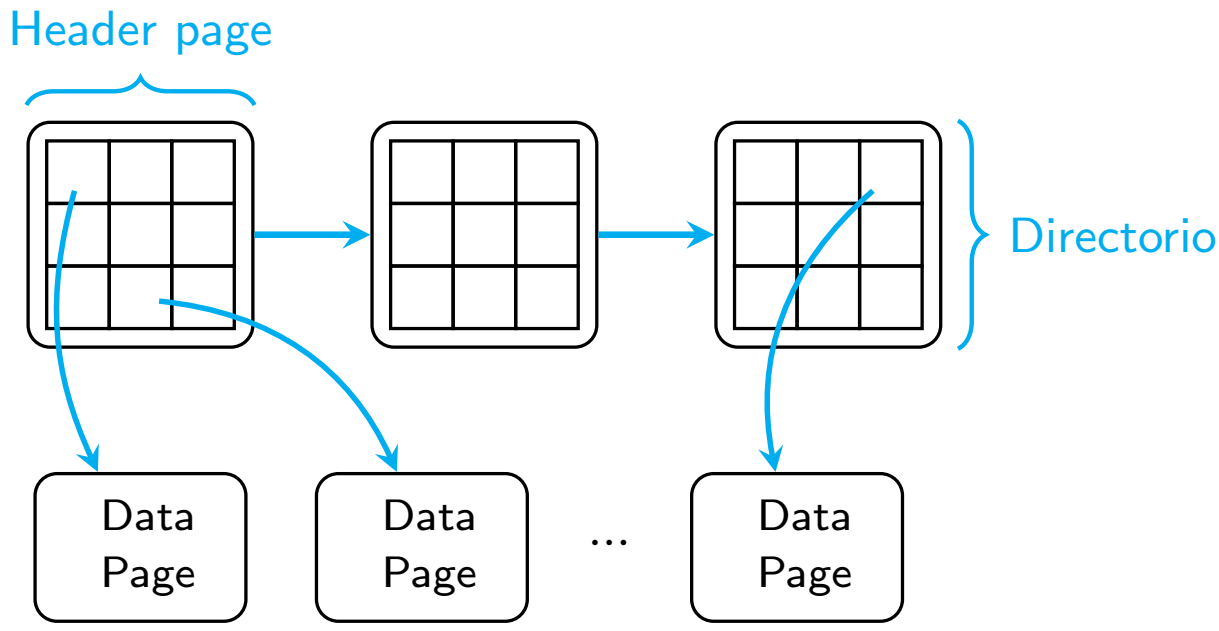


¿Problemas?:

- Todavía es necesario buscar en todas las páginas vacías por espacio para una tupla.

Heap file: Implementación 3

Directorio de páginas:



Ventajas:

- Directorio contiene meta-información sobre las páginas (espacio libre).
- Mayor eficiencia en la busqueda de espacio libre.

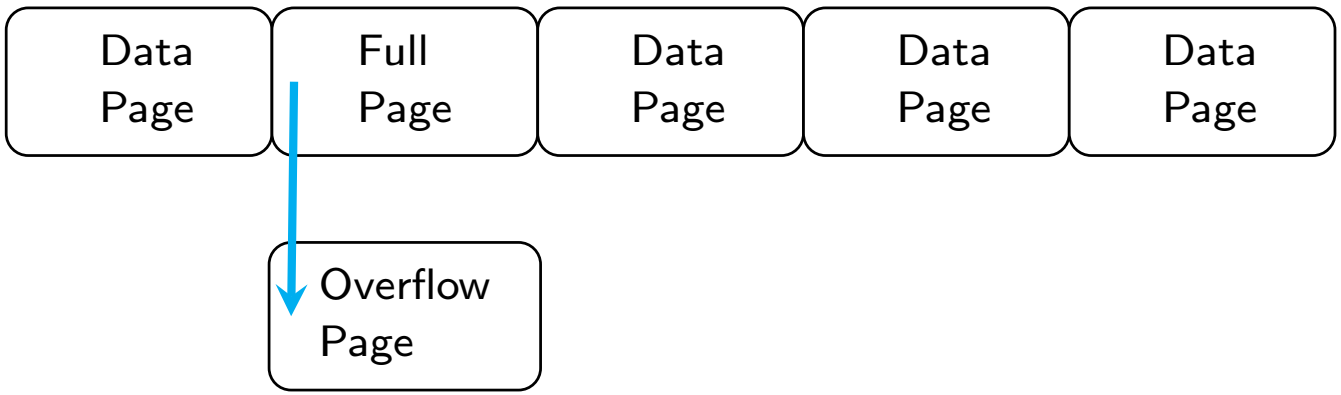
Heap file: modificaciones

Heapfiles no ordenados:

- Insertar, modificar o borrar tuplas es sencillo.

Heapfiles ordenados:

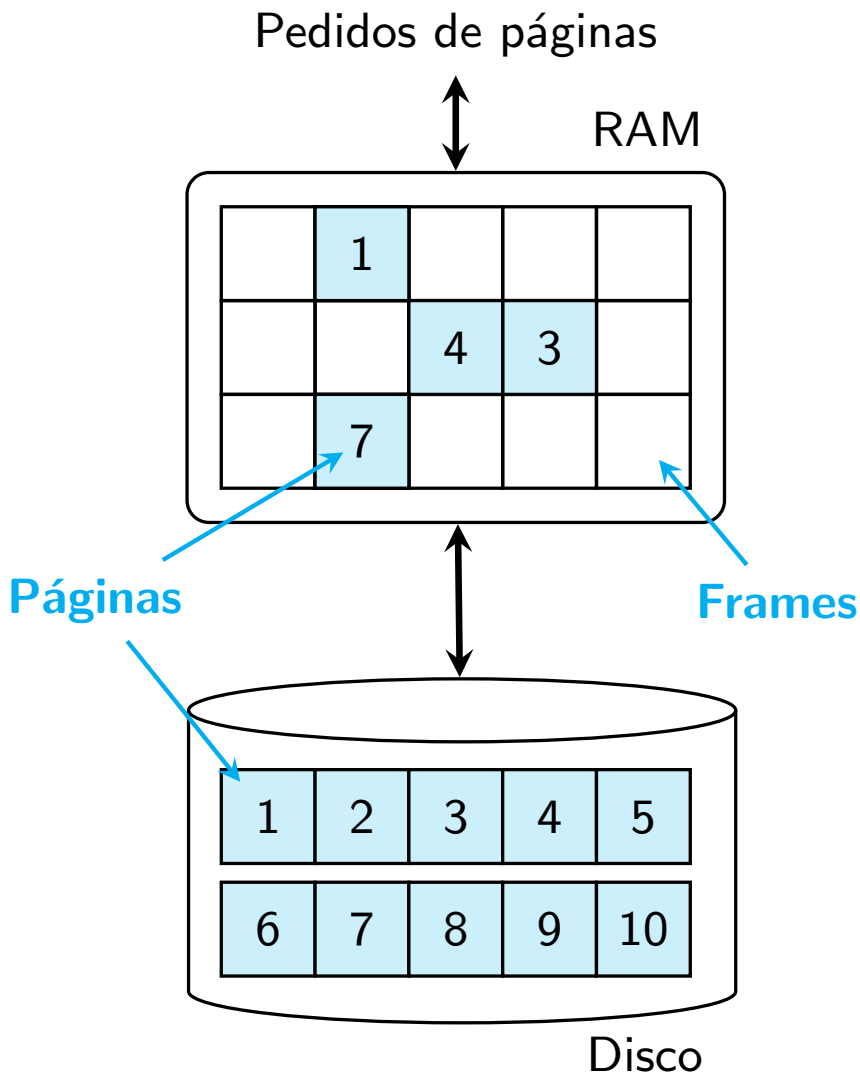
- Para insertar o modificar tuplas: usar **overflow pages**.



Outline

Buffer manager

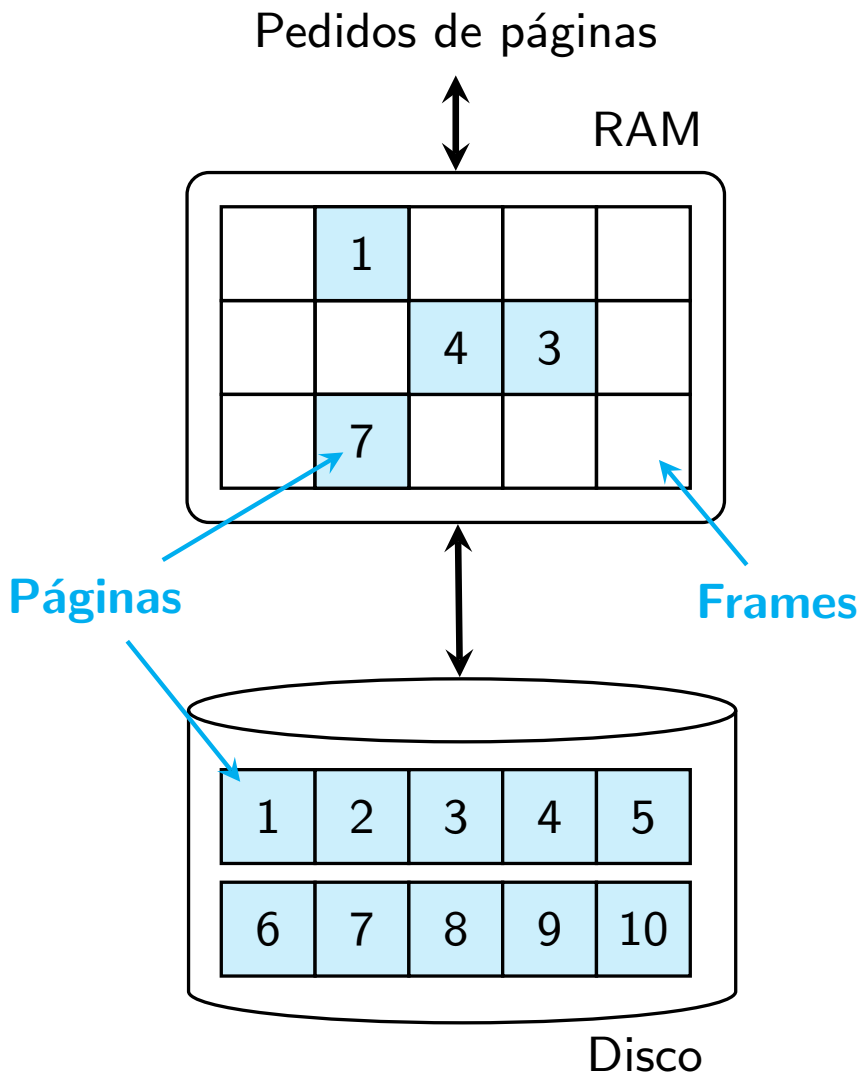
- Mediador entre el disco y la memoria principal.
- Cuenta con una cantidad restringida de memoria RAM.
- Páginas son traidas a memoria a pedido.
- Encargado de decir que páginas eliminar cuando el buffer esta lleno.



Buffer manager

Cada **frame** tiene una variable:

- **#pin** = cantidad de procesos que estan usando esta página.
- **Dirty** = si el contenido es necesario guardarlo en memoria.



Interfaz para acceder el buffer manager

Función `pin( pageno )`

1. Solicita la página `pageno` al buffer manager.
2. Si la página `pageno` esta en memoria, se incrementa su #-pin en 1.
3. Si la página `pageno` no esta en memoria:
   - Se selecciona un frame vació X.
   - Se trae la página `pageno` a memoria y se carga en X.
   - Se actualiza $X.\#pin := 1$ y $X.dirty := false$.
4. Buffer manager retonar una referencia al frame que contiene `pageno`.

¿Problemas?

- ¿qué ocurre si no hay un frame vacío?
- ¿qué ocurre si todos los frames tienen #pin mayor que 1?

Interfaz para acceder el buffer manager

Función `unpin( pageno, dirty )`

1. Solicita la liberación de la página `pageno` (almacenada en el frame X).
2. Se decrementa el $X.\#pin$ en uno.
3. Se actualiza $X.dirty := true$ si la página fue modificada.

¿Problemas?

- ¿cuándo son guardados los datos de la página en disco?
- ¿qué ocurre si un proceso nunca hace unpin de su página?

Requisitos de la interfaz `pin` y `unpin`

1. Cada proceso (de una BD) debe mantener las funciones `pin` y `unpin` **correctamente anidadas**.

   $d \leftarrow$ `pin`(p);
   $\vdots$
   (proceso lee y usa los datos en la dirrección de memoria d)
   $\vdots$
   `unpin`(p, false);

2. Cada proceso (de una BD) debe liberar una página lo antes posible.

Escritura concurrente de una página

Suponga lo siguiente:

- La misma página p es solicitada por más de un proceso ($\#pin > 0$).
- Mas de un proceso modifica el mismo dato de p.

¿qué hacemos?

... estos conflictos son manejados por el administrador de transacciones.

Políticas de reemplazo (replacement policies)

La efectividad del buffer manager depende directamente de la política de reemplazo usada.

Diferentes tipos de póliticas o (heurísticas).

- FIFO.
- Least Recently Used (LRU).
- Clock.
- Random.

Recordar: estas son heuristicas y pueden fallar.

Buffer manager en la práctica

Varias técnicas adicionales:

- Prefetching.
- Page fixing / hating.
- Buffer particionado.

Base de Datos vs. Sistemas Operativos

¿cuáles son las ventajas de usar un buffer manager propio?

- Mayor conocimiento del patrón de acceso a los datos.
- Política de reemplazo propia.
- Hacer pinned y unpinned de las páginas.
- Forzar el almacenamiento de páginas.
- Acceso “fino” por parte del administrador de transacciones.

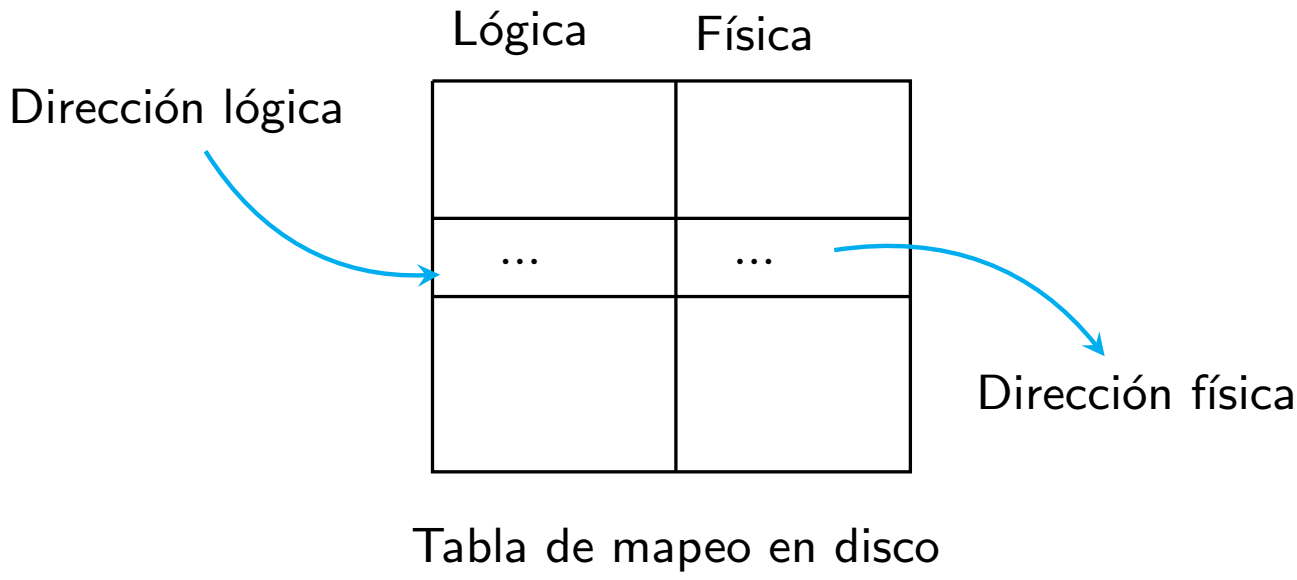
Outline

Direcciones de páginas y tuplas

1. Direcciones **físicas**, compuestas por:
   - Número de disco.
   - Número de cilindro.
   - Número de track dentro del cilindro.
   - Número de bloque dentro del track.
   - (Para tuplas) Posición de la tupla en el bloque.

Direcciones de páginas y tuplas

2. Direcciones **lógicas**: string arbitrario.



¿Ventajas de mantener direcciones lógicas?

- Nivel de **indireccionamiento**.
- Ahorro de espacio.

Problema de direccionamiento

- Tenemos una red de estructuras en memoria de la forma:

```
struct   node {
         int data;
         struct node *next;
};
```

- Deseamos almacenar esta red de estructuras en disco.

¿cómo almacenamos la dirección de los punteros en memoria?

Ejemplos:

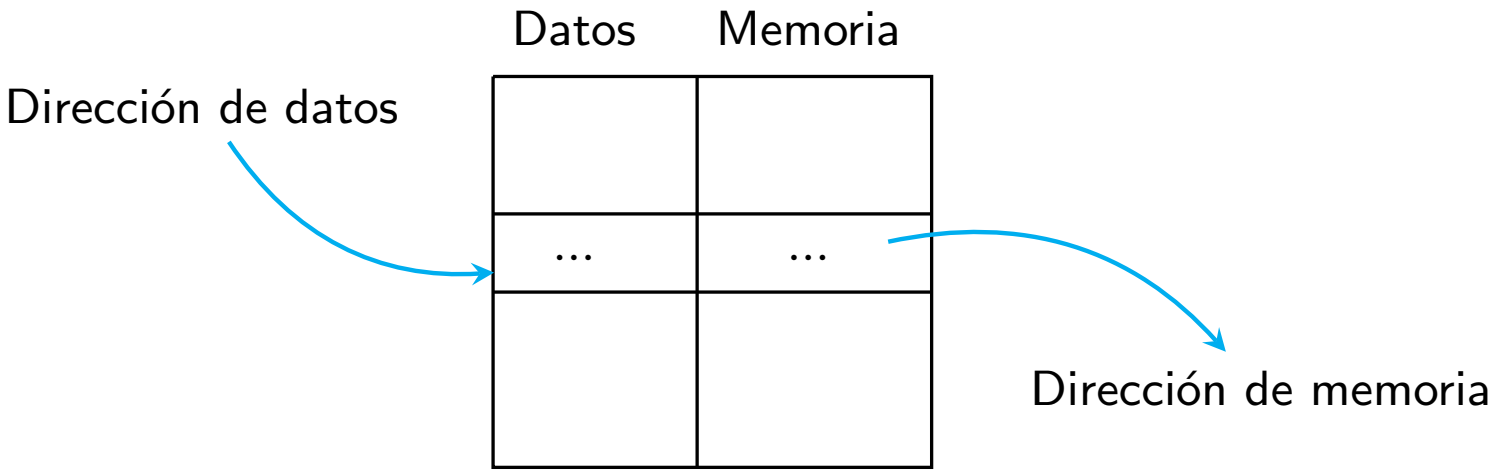
- BD orientada a objetos.
- Índices.

Pointer swizzling

Problema de convertir punteros de memoria en "punteros" en disco.

Solución:

- Mantener dos tipos de direcciones: dirección de **datos** y de **memoria**.
- Mantener una tabla para convertir dirección de datos en dirección de memoria y viceversa.



¿cuál es la diferencia con las direcciones anteriores?

Pointer swizzling

¿cuándo convertimos los punteros de datos en punteros de memoria?

Estrategias para determinar cuando cambiar o hacer “swizzling” del puntero:

- No swizzling.
- Swizzling automatico.
- Swizzling on-demand.
- Swizzling programado.

Pointer swizzling

Posible problema:

- Página A referencia con un puntero en memoria a B.
- B es eliminado del buffer.
- C es traido al buffer y puesto en la dirección de B.

¿Solución?

- Hacer **pinned** del frame de B para así no permitir su eliminación.

Outline

Alternativas de almacenamiento

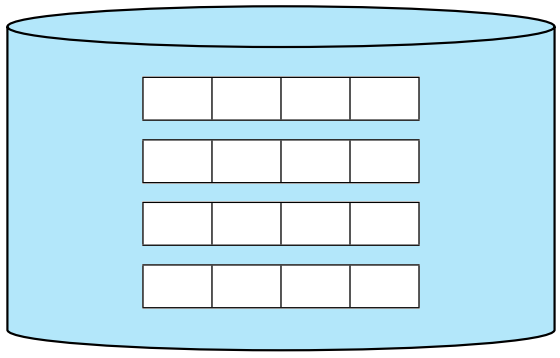
Hasta el momento hemos asumido que la información se guarda en filas.

¿qué sucedería si almacenamos la información por columnas?

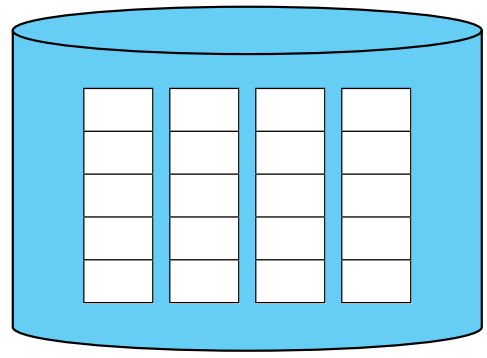
BD orientadas a columnas

Column-oriented DBMS

Basado en filas



Basado en columnas



BD orientadas a columnas

Ventajas de base de datos orientadas a **columnas**:

- Información puede ser compactada.
- Mayor eficiencia en operaciones basadas en columnas.

Ventajas de base de datos orientadas a **filas**:

- Mayor eficiencia en escritura.
- Mayor eficiencia en acceso por tuplas completas.

BD orientadas a columnas

Tema de interes actual en investigación.

- Leer "C-Store: A column oriented DBMS" por Stonebreaker et all.

Existen sistemas comerciales y académicos en el mercado:

- Vertica (C-store) – inicialmente desarrollado en MIT.
- MonetDB – desarrollado por CWI Amsterdam.