GESTIÓN DE DATOS 2012

- Clase 1 -

Bibliografía

- Files & Databases: An Introduction *(Smith-Barnes)*
- Estructuras de Archivos *(Folk-Zoellick)*
- Bases de Datos Relacionales *(Giménez, Casamyor, Herranz)*
- Principles in DataBase Systems *(Navathe-Cieri)*
- Diseño conceptual de Bases de Datos *(Batini, Navate, Cieri)*
- Sistemas de Bases de Datos *(Date)*
- Fundamento de sistemas de BD *(Elmasri - Navathe)*
- Modern Database Management *(Hoffer, Prescott, McFadden)*
- Fundamentos de Bases de Datos *(Korth Silvershatz)*

Base de Datos

¿?

Base de datos: *Colección o conjunto de datos interrelacionados con un propósito específico vinculado a la resolución de un problema del mundo real.*

Sistema de Gestión de Bases de Datos

- Actualmente, cualquier sistema de software necesita interactuar con información almacenada en una B. D. y para ello requiere del soporte de un SGBD

Sistema de Gestión de Bases de Datos (SGBD): *Conjunto de programas necesarios para acceder y administrar una base de datos.*

Sistema de Gestión de Bases de Datos

- Posee dos tipos diferentes de lenguajes:
  - Lenguaje de Definición de Datos (LDD): Se utiliza para diseñar la estructura de la BD, describir los datos y sus relaciones, la semántica asociada y las restricciones de consistencia.
  - Lenguaje de Manipulación de Datos (LMD): Se utiliza para agregar, recuperar, modificar o borrar datos.

Sistema de Gestión de Bases de Datos

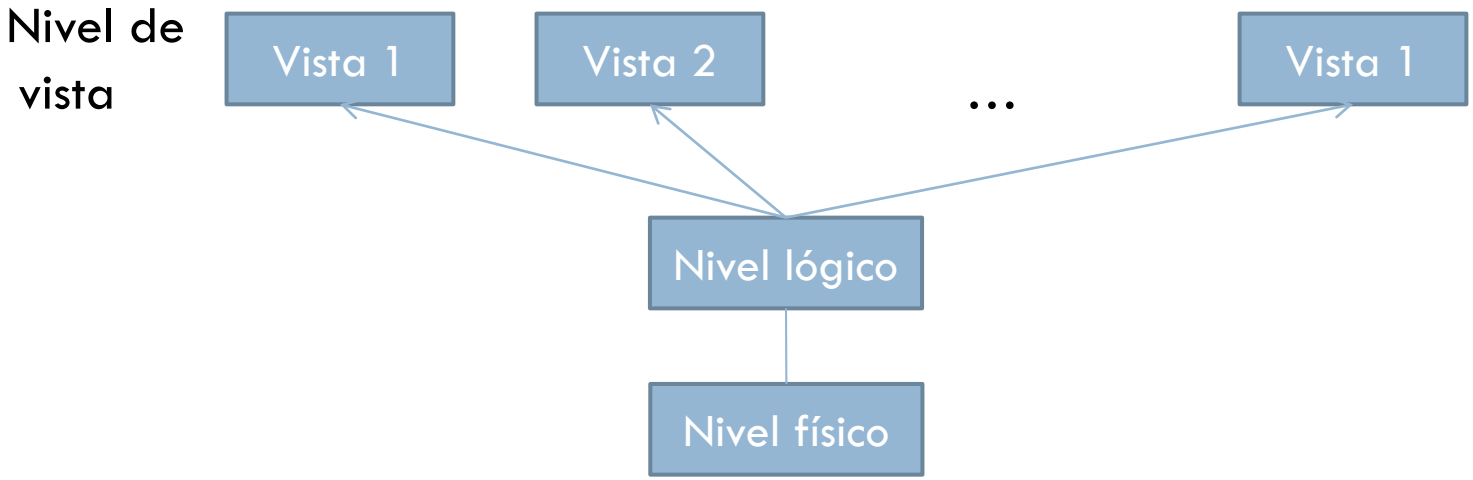
- Los objetivos más relevantes de un SGBD son:
  - Controlar la concurrencia: Varios usuarios pueden acceder a la misma información en el mismo momento. Si el acceso es para consulta, no hay inconvenientes. Si es para actualizar, se puede llegar a un estado de inconsistencia que, con la supervisión del SGBD se puede evitar.
  - Tener control centralizado: Tanto de los datos como de los programas que aceden a los datos.
  - Facilitar el acceso a los datos: Dado que provee un lenguaje de consulta rápida de información.

Sistema de Gestión de Bases de Datos

- Los objetivos más relevantes de un SGBD son:

  - Proveer seguridad para imponer restricciones de acceso: Se deben definir explícitamente quiénes son los usuarios autorizados a acceder a la BD.
  - Mantener la integridad de los datos: Esto implica que los datos respeten las condiciones establecidas al definir la estructura de la Base de Datos (por ej. no permitir un precio negativo) y que, ante una falla del sistema, se posea la capacidad de restauración.

Niveles de visión de los datos



- Nivel de vista: Corresponde al nivel más alto de abstracción. En este nivel se describe parcialmente la BD, solo la parte que se desea mostrar.
- Nivel lógico: Se describe la BD completa, indicando qué datos se almacenarán y las relaciones existentes entre esos datos.
- Nivel físico: Nivel más bajo de abstracción, en el cual se describe cómo se almacenan realmente los datos.

Almacenamiento de bases de datos

- Las bases de datos, generalmente, almacenan grandes cantidades de datos que deben *persistir* por períodos largos, y lo hacen en almacenamiento secundario (disco). Durante dichos períodos, los datos se acceden y procesan una y otra vez. Esto contrasta con la noción de estructuras de datos transitorias que persisten sólo por un tiempo limitado a la ejecución de un programa
- Los datos almacenados en el disco están organizados en archivos.

Archivos

Archivo: *Colección de registros que abarcan entidades con un aspecto común y originadas para algún propósito particular.*

- Hay varias organizaciones primarias de archivos que determinan la forma en que los registros se colocan físicamente en el disco, y por lo tanto, cómo se puede acceder a ellos.
  - Archivos no ordenados
  - Archivos ordenados
  - Archivos dispersos (hashing)

Archivos no ordenados

- Los registros se colocan en el archivo en el orden en que se ingresan.
- La inserción de un nuevo registro es muy eficiente: el último bloque de disco del archivo se copia en el búfer, se añade el nuevo registro y se reescribe el bloque de disco. La dirección del último bloque se guarda en la cabecera del archivo, y los registros nuevos se añaden al final del archivo.
- Buscar un registro con una condición de búsqueda requiere una búsqueda lineal, bloque por bloque (procedimiento costoso).
- Para eliminar un registro, primero se debe primero el bloque donde se encuentra, copiarlo en un búfer, eliminarlo del búfer y finalmente reescribir el búfer en el disco. Esto deja espacio desocupado en el bloque. También se puede utilizar una marca de eliminación. Ambas requieren reorganización periódica.

Archivos ordenados (secuenciales)

- Los registros se ordenan basándose en los valores de uno de sus campos, llamado campo de ordenación.
- Para la búsqueda de un registro, por el campo que está ordenado, se puede utilizar búsqueda binaria.
- Para insertar un nuevo registro, hay que encontrar su posición correcta en el archivo y luego abrir espacio para colocarlo en esa posición. Es una operación costosa.
- Para la eliminación de registros, se usan marcas de eliminación y reorganización periódica.
- Se utilizan pocas veces, a menos que se incluya un camino de acceso adicional, llamado índice primario; dando lugar así a los archivos secuenciales indexados.

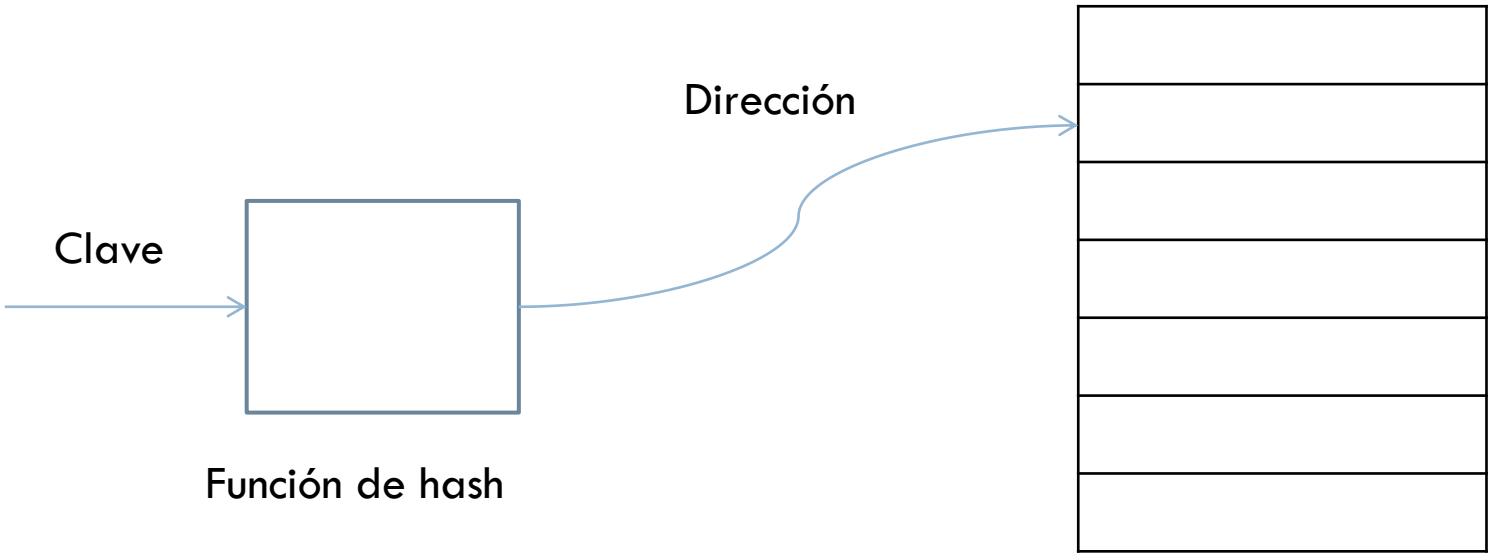
Archivos dispersos (hashing)

- Para insertar un registro con clave de búsqueda *k*, se calcula una función h(*k*) cuyo resultado proporciona la dirección del bloque para ese registro.
- Para realizar una búsqueda con el valor *k*, basta con calcular h(*k*) y luego buscar el bloque con esa dirección.
- El borrado es igual de sencillo. Si el valor de la clave a buscar es *k*, se calcula h(*k*), después se busca el bloque correspondiente a esa dirección y se borra.

Hashing (Dispersión)

- **Tipos de dispersión**
  - Hashing con espacio de direccionamiento estático: El espacio disponible para dispersar los registros de un archivo de datos está fijado previamente.
  - Hashing con espacio de direccionamiento dinámico: El espacio disponible para dispersar los registros de un archivo de datos aumenta o disminuye en función de las necesidades de espacio que en cada momento tiene el archivo.

Hashing (Dispersión)



Hashing (Dispersión)

Función de hash: Función que transforma un valor, que representa una clave primaria de un registro, en otro valor dentro de un determinado rango, que se utiliza como dirección física de acceso para insertar un registro en un archivo de datos.

- Características:
  - ✓ Debe distribuir las claves almacenadas uniformemente entre todas las direcciones, de modo que cada una de ellas tenga el mismo número de registros.
  - ✓ No hay relación aparente entre clave y dirección.
  - ✓ Dos claves distintas pueden transformarse en iguales direcciones (colisiones).

Hashing (Dispersión)

Colisión:

- Situación en la que un registro es asignado a una dirección ya ocupada (no tiene suficiente espacio para ser almacenado).
- A las claves que por dispersión se convierten en la misma dirección ➔ **sinónimos**
- La solución sería elegir un algoritmo de dispersión sin colisiones (perfectos, imposible de conseguir).

Hashing (Dispersión)

- **Soluciones para las colisiones**
  - **Esparcir registros:** Buscar métodos que distribuyan los registros de la forma más aleatoria posible entre las direcciones disponibles.
    - Utilizar 2 letras no es bueno.
  - **Usar memoria adicional:** Distribuir pocos registros en muchas direcciones (ej: 75 registros en 1000 direcciones):
    - Disminuye el overflow (colisiones).
    - Desperdicia espacio.

Hashing (Dispersión)

- **Soluciones para las colisiones**
  - **Colocar más de un registro por dirección:**
    - Direcciones con N claves.
    - Mejoras notables.
    - Ej: Bloque de 512 bytes y el registro a almacenar es de 80 bytes → se pueden almacenar hasta 6 registros por cada dirección de archivo (cada dirección tolera hasta 5 sinónimos)
    - Direcciones que pueden almacenar varios registros en esta forma → *cubetas*

Hashing (Dispersión)

- **Densidad de empaquetamiento**

  Es la relación entre el espacio disponible para el archivo de datos y la cantidad de registros que integran dicho archivo.

$$DE = \frac{\text{cantidad de registros que componen un archivo}}{\text{cant. de registros por nodo} * \text{cant. de direcciones}}$$

  - Por ejemplo, si se debieran esparcir 30 registros entre 10 direcciones con capacidad de 5 registros por cada dirección,

$$DE = 30 / 5 * 10 = 0{,}6 \text{ ó } 60\%$$

Hashing (Dispersión)

- Cuanto mayor sea la densidad de empaquetamiento, mayor será la posibilidad de colisiones, dado que se dispone de menos espacio para esparcir registro.

- Por otra parte, cuando la densidad de empaquetamiento se mantiene baja, se desperdicia espacio en disco, dado que se utiliza menor espacio que el reservado, generando fragmentación.

Hashing (Dispersión)

- **Aún utilizando todas las soluciones presentadas, ocurrirán colisiones** → Se debe incorporar algún método para tratar con los registros que no pueden entrar en su dirección base.

- **Tratamiento de colisiones**
  - Saturación progresiva
  - Saturación progresiva encadenada
  - Saturación progresiva encadenada (en áreas separadas)
  - Dispersión doble

Hashing (Dispersión)

- **Saturación Progresiva**
  - Cuando se completa una dirección de memoria se busca en las siguientes direcciones en secuencia, hasta encontrar una vacía.
  - La gran ventaja es su simplicidad.

  - **Búsqueda:**
    - Comienza en la dirección base y continúa buscando en localidades sucesivas hasta encontrar la clave buscada (puede haber circularidad).
    - Si se encuentra una dirección vacía-> se puede suponer que la clave buscada no está en el archivo.

Hashing (Dispersión)

- **Saturación Progresiva**
  - **Eliminación:**
    - No debe permitirse que el espacio liberado por la eliminación obstaculice las búsquedas posteriores.
    - Al mismo tiempo, debe ser posible utilizar el espacio liberado para adiciones posteriores.
    - La búsqueda finaliza al encontrar una dirección vacía, por eso no es conveniente dejar direcciones vacías, que terminen la búsqueda por saturación en forma inapropiada.
    - Se marca el espacio liberado (por ej. con ####):
      - El espacio liberado no rompe la secuencia de búsquedas.
      - El espacio liberado está disponible y puede ser usado en adiciones posteriores.

Hashing (Dispersión)

- **Saturación progresiva encadenada**
  - Es otra técnica para evitar los problemas causados por la acumulación de registros.
  - Funciona igual que la Saturación Progresiva, excepto que las claves sinónimos se enlazan por apuntadores.
  - Cada dirección base contiene un número que indica el lugar del siguiente registro con la misma dirección base. El siguiente registro contiene a la vez un puntero al siguiente registro con la misma dirección base y así sucesivamente.

Hashing (Dispersión)

- **Saturación progresiva encadenada**
  - **Ventaja:** Sólo se necesita acceder a los registros con claves que son sinónimos.
    - Mejora el N° de accesos promedio
  - **Desventaja:** Debe agregarse un campo de enlace a cada registro → requiere mayor espacio de almacenamiento.

Hashing (Dispersión)

- **Saturación progresiva encadenada**
  - Problemas: Acceder a una dirección base ocupada con un registro que no es de ese lugar.
  - Ej. Supongamos que Beta tiene dirección base 22, pero se inserta en 23, entonces cuál es el siguiente de Delta, el 23 o el 25 ?

Dir. Base	Dir. Real	Clave	Encadenado
20	20	Alfa	22
21	21	Epsilon	23
20	22	Delta	25
21	23	Beta	-1
24	24	Gamma	-1
20	25	Pi	-1

Hashing (Dispersión)

- **Saturación progresiva encadenada**
  - **Solución:** Cargar el archivo en 2 pasos

    1) Sólo cargar los registros con direcciones Base. Los registros (duplicados) que no son base se guardarán en un archivo separado → se garantiza que ninguna dir. base estará ocupada por registros en saturación.

    2) Cargar los repetidos en direcciones libres.

  - Aunque esta solución no garantiza que las eliminaciones y/o inserciones posteriores no tendrán problemas.

Hashing (Dispersión)

- **Saturación progresiva con encadenamiento en áreas separadas**
  - Al conjunto de direcciones base se le llama área principal de datos.
  - Al conjunto de direcciones en saturación se le llama área de saturación.
  - Cuando se agrega un registro nuevo si hay lugar en dirección base se almacena allí, sino se mueve al archivo de saturación (en un área separada) donde se agrega a la lista enlazada que comienza en la dirección base.

Hashing (Dispersión)

- **Dispersión doble**
  - Cuando sucede una colisión se aplica una segunda función de dispersión a la clave, para producir un N°, el cual se suma a la dirección original tantas veces como sea necesario hasta encontrar una dirección vacía (con espacio).
  - Se evita acumulamiento.
  - Los registros no quedan “locales”, tienden a esparcirse en el archivo.

Hashing (Dispersión)

- Si el archivo se completa o las cadenas de saturación se vuelven tan largas que la respuesta resulta inadmisible, ¿qué hacemos?
- Se tiene que incrementar el espacio de direcciones y, usando una nueva función de dispersión, REDISPERSAR el archivo completo.
- El costo de redispersar es muy alto. El tiempo utilizado es importante, y mientras se realiza la operatoria no es posible que el usuario final de la información acceda al archivo. Por esto, cuando se piensa en la creación de un archivo, es importante:
  - Analizar la tasa de crecimiento posible del archivo.
  - Determinar la cantidad de direcciones que optimice el uso de espacio vs. la periodicidad de la redispersión.
- Alternativa: Trabajar con archivos que administren el espacio de direccionamiento de manera dinámica.

Hashing (Dispersión)

- **Hashing con espacio de direccionamiento dinámico**: Dispersa las claves en función de las direcciones disponibles en cada momento, y la cantidad de direcciones puede crecer, a priori sin límites, en función de las necesidades de cada archivo particular.
- ***Hash extensible***: Alternativa de implementación para hashing con espacio de direccionamiento dinámico.
  - Consiste en comenzar a trabajar con un único nodo para almacenar registros e ir aumentando la cantidad de direcciones disponibles a medida que los nodos se completan.
  - No se utiliza el concepto de densidad de empaquetamiento ya que el espacio utilizado aumenta o disminuye en función de la cantidad de registros que dispone el archivo en cada momento.

Hashing (Dispersión)

- **Hash extensible**
  - Como las direcciones no están definidas a priori, la función de hash retorna un string de bits. La cantidad de bits que retorna determina la cantidad máxima de direcciones a las que puede acceder el método.
  - Por ejemplo, con una función de hash que retorna 32 bits es posible direccionar 2^{32} direcciones. Si además, en cada dirección se pueden almacenar varios registros, la cantidad de claves a dispersar es importante.

Hashing (Dispersión)

- **Hash extensible**
  - Necesita, para su implementación, de una estructura auxiliar (una tabla) que se administra en memoria principal y contiene la dirección física de cada nodo.
  - Comienza con un solo nodo en disco y una tabla que solamente contiene la dirección a ese único nodo.

Hashing (Dispersión)

- **Hash extensible**
  - Se utilizan sólo los bits necesarios de acuerdo con cada instancia del archivo.
  - Los bits tomados forman la dirección del nodo que se utilizará.
  - Si se intenta insertar en un nodo lleno, deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nuevo; para ello se toma 1 bit más.
  - La tabla tendrá tantas entradas (direcciones de nodos) como 2^i, siendo i el número de bits actuales para el sistema.

Hashing (Dispersión)

- **Hash extensible**
  - El proceso de búsqueda asegura encontrar cada registro en un solo acceso. Se calcula la secuencia de bits para la clave, se toman tantos bits como indique el valor asociado a la tabla, y la dirección del nodo contenida en la celda respectiva deberá contener el registro buscado. En caso de no estar, el elemento no forma parte del archivo de datos.
  - Desventaja: Requiere mayor procesamiento cuando una inserción genera overflow.

Hashing (Dispersión)

- **Ventajas y desventajas del hash extensible**
  - La ventaja principal del hash extensible es que el rendimiento no se degrada a medida que el archivo aumenta de tamaño.
  - No es necesario reservar bloques para un futuro aumento de tamaño.
  - El espacio adicional que requiere la tabla de direcciones es pequeño.
  - Un inconveniente del hash extensible es que la búsqueda implica un nivel adicional de referencia.