Base de datos:

1) Definición: colección o conjunto de datos interrelacionados con un propósito específico vinculado a la resolución de un problema del mundo real.
2) Propiedades implícitas:
    a) Representa aspectos del mundo real (Universo de discurso).
    b) Colección coherente de datos con significados inherentes.
    c) Se diseña, construye y completa de datos con propósito específico.
    d) Se sustenta físicamente en archivos en dispositivos de almacenamiento secundarios.

Archivos:

1) Definición: coleccion de registro (datos), guardados en dispositivos secundarios de memoria, que abarcan entidades con un aspecto común y originados para algún propósito en particular.
2) Hardware:
    a) Las estructuras en general se definen y usan en RAM y el archivo persiste en memoria secundaria.
    b) Buffers provisorios sirven de memoria intermedia entre el archivo y el programa, estos residen en RAM y los maneja el Sistema Operativo.
    c) Capacidad: RAM < Secundaria
    d) Velocidad: RAM > Secundaria
3) Organización:
    a) Secuencia de bytes (archivos de texto), no se puede determinar fácilmente el comienzo y fin de cada dato.
    b) Registros y campos: pueden ser fijos o variables.
        i) Registro: conjunto de campos agrupados que definen un elemento.
        ii) Campo: unidad más pequeña lógicamente significativa.
4) Acceso:
    a) Secuencial Físico: se accede a los registros uno tras otro en el orden físico guardado.
    b) Secuencial Indizado (lógico): acceso a registros de acuerdo al orden establecido por otra estructura.
    c) Directo: acceso a registro sin acceder a sus predecesores.
    d) Los archivos se pueden clasificar según su tipo de acceso como Serie (Secuencial Físico), Secuencial (Secuencial Indizado) o Directo (Directo).
5) Viaje de un Byte:
    a) Involucrados:
        i) Administrador de archivos: conjunto de programas del SO que tratan aspectos relacionados con archivos y dispositivos de E/S.
        ii) Buffers de E/S: agilizan la E/S de datos.
        iii) Procesador de E/S: dispositivos utilizados para la transmisión desde o hacia almacenamiento externo. Independiente de la CPU.
        iv) Controlador de disco: encargado de controlar las operaciones del disco.
    b) Capas de protocolo de transmisión:
        i) Programa pide al SO escribir de variable a archivo.

ii) El SO transfiere el trabajo al administrador de archivos.
iii) El Adm. busca en su tabla de archivos y verifica las características.
iv) El Adm. obtiene de la FAT la ubicación física del sector del archivo donde se guardará el byte.
v) El Adm. se asegura que el sector este en buffer y lo graba allí.
vi) El Adm. da instrucciones al procesador de E/S, donde está el byte en RAM y en qué parte del disco se va a almacenar.
vii) El procesador de E/S encuentra el momento para transmitir el dato a disco, la CPU se libera.
viii) El procesador de E/S envía el dato al controlador de disco, con direcciones a ubicar.
ix) El controlador de disco prepara la escritura y transfiere el dato bit a bit al disco.

6) Claves:
    a) Definición: dato que permite identificar un registro y generar orden en el archivo por ese criterio.
    b) Primaria (unívocas): identifican a un elemento en particular en el archivo.
    c) Secundaria: reconoce un conjunto de elementos de igual valor.
    d) Forma canónica: forma estándar para una llave, que le da representación única, puede derivarse a partir de reglas bien definidas.
    e) Performance:
        i) Secuencial: Mejor 1; Peor N; Promedio N/2.
        ii) Directo: Mejor 1; Peor 1; Promedio 1.
        Nota: solo es preferible cuando se necesitan pocos registros específicos.
7) Tipos según # de cambios:
    a) Estáticos: pocos cambios, pueden actualizarse en procesamiento por lotes y no necesitan de estructuras adicionales.
    b) Volátiles: operaciones frecuentes (Agregar, Borrar y Actualizar), su organización debe facilitar cambios rápidos y necesita de estructuras adicionales para mejorar su tiempo de acceso.
8) Eliminación:
    a) Baja física: borrar efectivamente el registro recuperando el espacio físico. Se puede lograr:
        i) Generando un nuevo archivo: consiste en recorrer el archivo original copiando a un nuevo archivo la info que se desea mantener, guardandolo y borrar el archivo inicial.
        ii) Usando el mismo archivo: consiste en borrar el registro, reacomodar los demás y truncar el archivo.
    b) Baja lógica: borrar info de un archivo sin recuperar el espacio físico, se consigue poniendo una marca de borrado en el registro.
    c) Recuperación de espacio (bajas lógicas only)
        i) Recuperar espacio con bajas físicas.
        ii) Reasignación de espacio: utilizar lugares indicados como borrados para el ingreso de nuevos elementos del archivo.
            (1) Avanzar y poner registros en algún lugar libre (ineficiente).

(2) Usar una lista encadenada invertida.

iii) Adaptaciones para que los métodos funcionen para registros de longitud variable

(1) En la lista no se puede usar el NRR (número relativo de registro) como enlace por lo que se debe utilizar un campo binario que indique explícitamente el enlace y convenientemente el tamaño.

(2) Como no se pueden colocar en cualquier espacio por cuestiones de tamaño existen formas de selección:

(a) Primer ajuste: se elige el primer espacio en que entre.

(b) Mejor ajuste: se elige el espacio más adecuado, es decir de menor tamaño, y se le asigna.

(c) Peor ajuste: se elige el espacio de mayor tamaño asignando sólo los bytes necesarios.

(3) Las selecciones causan fragmentación:

(a) Interna: cuando se asigna mayor espacio del necesario (primer y mejor ajuste).

(b) Externa: cuando el espacio entre dos registros es muy pequeño para ser reutilizado (peor ajuste).

9) Modificaciones:
   a) En longitud variable puede ocurrir que la modificación sea de mayor tamaño que el original por lo que se debe reasignar de posición. La solución más simple es dar de baja el registro e insertarlo (eliminación y alta).
   b) Otros problemas que pueden surgir:
      i) El orden al cambiar la clave de un registro.
      ii) Modificaciones si hay claves duplicadas.
10) Búsquedas:
   a) Cantidad de comparaciones (operaciones en memoria): se mejora con algoritmos.
   b) Cantidad de accesos (operaciones en disco)
   c) Buscar un registro:
      i) Rápido con el NRR (directo).
      ii) En secuencial se debe recorrer desde el principio, las claves y el orden mejoran la eficiencia.
   d) Búsqueda binaria
      i) Partir el archivo a la mitad, comparar clave e ir por el lado conveniente. Repetir hasta encontrarlo, o no.
      ii) Precondiciones: registros de longitud fija y archivo ordenado por clave.
      iii) Si N es la cantidad de registros la performance es O(log2(N))
      iv) Costos: mantener el archivo ordenado (clasificar):
         (1) Archivo a RAM: muy pesado.
         (2) Claves a RAM: mejor pero puede no alcanzar.
         (3) Claves > RAM: ordenar en disco, muy lento.
      v) Soluciones:
         (1) Partir el archivo, ordenar partes y hacer merge.

(2) No reordenar todo el archivo junto.
(3) Reorganizar con otras estructuras (Índices o Árboles).

11) Operaciones básicas en Pascal:
    a) Definición de archivo:
        Type archivo = file of tipo_de_dato;
        var archivo_logico: archivo;
    b) Correspondencia lógico-física:
        Assign(nombre_logico, nombre_fisico);
    c) Apertura y creación:
        rewrite(nombre_logico); //reescribe
        reset(nombre_logico); //reinicia
    d) Cierre de archivos:
        close(nombre_logico); //pone EOF al final
    e) Lectura:
        read(nombre_logico, var_dato);
    f) Escritura:
        write(nombre_logico, var_dato);
    g) Creación:
        definir archivo, asignar, abrirlo, escribirlo y cerrarlo
    h) Operaciones adicionales:
        i) EOF(nombre_logico);
        ii) FileSize(nombre_logico);
        iii) FilePos(nombre_logico);
        iv) seek(nombre_logico, posicion);

Algoritmia clásica sobre archivos: (definiciones y consejos)

1) Agregar datos a archivo existente (Algoritmia básica): abrir archivo, posicionar, leer dato, escribir y cerrarlo.
2) Actualización Maestro-Detalle:
    a) Maestro: archivo que resume información sobre un dominio específico.
    b) Detalle: archivo que contiene novedades o movimientos realizados sobre la información almacenada en el maestro.
    c) El algoritmo depende mucho de las precondiciones del ejercicio.
    d) Con un detalle sin repetición se puede hacer el while(not eof(archivo))
    e) Con un detalle con repetición hacer el siguiente procedure y preguntar en los while por lo que sea <> a valor_alto:

```
const valor_alto='999999'
procedure leer (var archivo: detalle; var dato: v_prod);
    begin
        if (not eof(archivo))
            then read (archivo, dato)
            else dato.cod := valor_alto;
    end;
```

    f) Con N detalles hacer algoritmo que analice el dato mínimo de alguno de los archivos, para hacer esto voy a necesitar variables del principio de cada archivo y hacerlas avanzar a medida que las utilice.

3) Corte de control:
    a) Los problemas consisten en la generación de reportes. Usaremos de ejemplo Provincia -> Ciudad -> Sucursal
    b) Casi siempre hay que totalizar por cada región por ende declarar variables totalizadoras y variables para el corte de control.
    c) Bucles esenciales en la práctica:
        i) while (reg.provincia <> valor_alto)
        ii) while (prov = reg.provincia)
            while (prov = reg.provincia) and (ciudad = reg.Ciudad)
                while (prov = reg.provincia) and (ciudad = reg.Ciudad) and (sucursal = reg.sucursal)
4) Merge:
    a) Resumir varios detalles en un solo maestro.
    b) Hacer el procedure 'leer' para el while (min.cod <> valor_alto).
    c) Si hay repetición de información hacer un while (codprod = min.codigo).
    d) Hacer un procedure de mínimo.
    e) Básicamente, una mezcla de todo lo anterior.

Índices:

1) Definiciones:
    a) Herramienta para encontrar registros en un archivo, consistente de campo clave y campo de referencias.
    b) Estructura de datos usada para decrementar el tiempo de acceso a un archivo.
    c) Tabla que opera con un procedimiento que acepta valores de entrada y provee valores de salida.
2) Característica fundamental: permite imponer orden sin re-acomodar físicamente, teniendo un archivo de datos y otro de índices.
3) El índice primario de la tablas se crea a partir de la clave primaria.
4) Operaciones básicas:
    a) Crear archivos, índice y de datos.
    b) Cargar el índice en memoria (al ser chico entra).
    c) Trabajar con el índice en memoria (busquedas binarias).
    d) Reescritura del archivo, hacer cambios mediante índices.
5) Cambios:
    a) Agregar registros: copiar la dirección del registro agregado, meterla en tabla de índices y ordenarla.
    b) Eliminar: quitar la dirección del registro eliminado de la tabla.
    c) Actualizar:
        i) Si no cambia de dirección no hay dificultades.
        ii) Si cambia de dirección el registro, se debe eliminar y agregar.
6) Ventajas (generales):
    a) Se pueden almacenar en memoria principal.
    b) Permite la búsqueda binaria.
    c) Mantenimiento menos costoso.
7) Desventajas (generales):

a) Puede no caber en RAM.
b) Reescritura del archivo de índices.
c) Mantener la persistencia de datos.

8) Índices secundarios: permiten relacionar una clave secundaria con uno o más claves primarias, permitiendo hacer búsquedas más naturales.
    a) Problemas: mucho espacio, muchas ocurrencias y muchos reacomodamientos.
    b) Soluciones:
        i) Usar un arreglo: clave + vector de punteros con ocurrencias. El problema es elegir el tamaño ya que es fijo.
        ii) Usar una lista de punteros con ocurrencias, mejor aún si es una lista invertida para evitar las reorganizaciones completas.

Árboles:

1) Es una estructura de datos que permite localizar en forma más rápida la información de un archivo, permiten intrínsecamente la búsqueda binaria.
2) Árboles binarios: estructura donde cada nodo tiene dos sucesores, izquierda y derecha. La búsqueda es de O(log2(N+1)).
    a) Implementación en Pascal:

```
type registro_arbol_binario = record
        elemento_de_dato: tipo_de_dato;
        hijo_izq, hijo_der_ integer;
end;
indice_binario = file of registro_arbol_binario;
```

    b) Balanceados: cuando la altura de la trayectoria más corta a una hoja no difiere de la altura de la trayectoria más grande.
    c) AVL: árbol binario balanceado en altura (BA(1)) en el que las inserciones y eliminaciones se efectúan con un mínimo de accesos. Su búsqueda es de O(1.44 log2(N+2)).
    Para cada nodo existe un límite en la diferencia que se permite entre las alturas de cualquiera de los subárboles del nodo.
    d) Paginados: divide al árbol en páginas para minimizar accesos a memoria secundaria y que logre entrar en la memoria primaria. El problema al implementarlo es cómo elegir la raíz de cada página, cómo mantenerlo balanceado, y su construcción descendente.
3) Arboles multicaminos: cada nodo tiene K punteros y K-1 claves, disminuye la profundidad.
    a) Balanceados: se construyen de forma ascendente. Existe:
        i) Árbol B de orden M:
            (1) Propiedades:
                (a) Ningún nodo tiene más de M hijos.
                (b) Cada nodo (menos la raíz y el terminal) tienen como mínimo [M/2] hijos.
                (c) La raíz tiene como mínimo 2 hijos o ninguno.
                (d) Todos lo nodos terminales están a igual nivel.

(e) Los nodos no terminales con K hijos tienen K-1 registros.
(f) Los nodos terminales tiene como mínimo [M/2]-1 registros y como máximo M-1 registros.

(2) Búsqueda:
(a) Peor caso: h lecturas (h altura del árbol).
(b) Mejor caso: 1 lectura.

(3) Estimación de h: h <= [1 + log[m/2]((n+1)/2)].
Se consigue a partir de un axioma que dice que árbol balanceado de orden M, si el número de elementos es N hay N+1 punteros nulos en nodos terminales.

(4) Inserción: registros se insertan en nodos terminales.
(a) Casos:
(i) Sin Overflow: se inserta y se re-acomodan los elementos en el nodo.
(ii) Overflow: Se divide el nodo, se reparte los elementos y se promociona uno a nivel superior. El overflow se puede propagar y generar una nueva raíz.
(b) Performance:
(i) Mejor caso (sin overflow): h lecturas y 1 escritura.
(ii) Peor caso (overflow hasta la raíz): h lecturas y 2h+1 escrituras (2 por nivel más la raíz).

(5) Eliminación: siempre eliminar de nodos terminales, si el registro no está allí debe llevarse a nodo terminal.
(a) Casos:
(i) Se borra elementos del nodo y no hay underflow, solo reacomodar.
(ii) Se produce underflow (# elem < [M/2]-1).
(b) Soluciones para underflow:
(i) Re-distribuir: trasladar llaves de un nodo a su adyacente hermano. Se puede en caso de que el otro tenga suficientes elementos.
(ii) Concatenación: si el adyacente hermano está al mínimo, se concatenan disminuyendo el # de nodos, en algunos casos hasta la h del árbol
(c) Performance:
(i) Mejor (borrar de nodo terminal): h lecturas y 1 escritura.
(ii) Peor (concatenación que decrementa en 1 el nivel del árbol): 2h-1 lecturas y h+1 escrituras.

ii) Árbol B*:
(1) Propiedades:
(a) Ningún nodo tiene más de M hijos.

(b) Cada nodo (menos la raíz y el terminal) tienen como mínimo [(2M-1)/3] hijos.
(c) La raíz tiene como mínimo 2 hijos o ninguno.
(d) Todos lo nodos terminales están a igual nivel.
(e) Los nodos no terminales con K hijos tienen K-1 registros.
(f) Los nodos terminales tiene como mínimo [(2M-1)/3]-1 registros y como máximo M-1 registros.

(2) Inserción: derecha; izquierda o derecha; izquierda y derecha.

iii) Árbol B+: conjunto de grupo de registros ordenados por clave en forma secuencial más conjunto de índices para rápido acceso:
(1) Propiedades:
(a) Ningún nodo tiene más de M hijos.
(b) Cada nodo (menos la raíz y el terminal) tienen como mínimo [M/2] hijos.
(c) La raíz tiene como mínimo 2 hijos o ninguno.
(d) Todos lo nodos terminales están a igual nivel.
(e) Los nodos no terminales con K hijos tienen K-1 registros.
(f) Los nodos terminales tiene como mínimo [M/2]-1 registros y como máximo M-1 registros.
(g) Los nodos terminales representan un conjunto de datos y son linkeados entre ellos.
(h) Los nodos no terminales no tienen datos sino punteros a los datos.
(2) Conjunto de secuencia (otra estructura que da origen a esta): conjunto de registros que mantienen un orden físico por llave mientras que se agregan o quitan datos, si podemos mantenerlo, podemos indizar.
(3) Separadores: derivados de las llaves de los registros que limitan un bloque en un conjunto de secuencia.
(4) Árbol B+ de prefijos simples: conjunto índice constituido por separadores cortos.

4) Conclusiones:
a) Ubicación de datos:
i) Árbol B: nodos (cualquiera).
ii) Árbol B+: nodos terminales.
b) Tiempo de búsqueda: igual en ambos árboles.
c) Procesamiento secuencial:
i) Árbol B: lento (complejo).
ii) Árbol B+: rápido (con punteros)-
d) Inserción y eliminación: el árbol B+ requiere más tiempo por las conexiones.

Hashing:

1) Técnica de almacenamiento y recuperación que usa una función de hash para mapear registros en direcciones de almacenamiento.
2) Atributos:
    a) No requiere almacenamiento adicional (índice).
    b) Facilita la inserción y eliminación rápida de registros.
    c) Encuentra registros con pocos accesos al disco.
3) Costo:
    a) No se puede usar registros de longitud variable.
    b) No puede haber orden físico de datos.
    c) No permite llaves duplicadas.
4) Determinar dirección:
    a) La clave se transforma en un número aleatorio.
    b) El número aleatorio en dirección de memoria.
    c) El registro se intenta colocar en la dirección asignada.
    d) Si la dirección está ocupada ocurre una colisión y puede ocurrir overflow si no hay espacio.
5) Parámetros
    a) Función de hash: transforma la clave en dirección para el registro.
        i) Si dos llaves tienen igual dirección ocurre:
            (1) Colisión: situación donde el registro es asignado a una dirección ya utilizada.
            (2) Overflow: situación donde el registro es asignado a una dirección utilizada y en la misma no hay espacio.
        ii) Soluciones:
            (1) Esparcir registros: que la función sea más aleatoria.
            (2) Usar un mejor algoritmo para la función.
            (3) Usar más memoria para que haya más direcciones.
            (4) Que haya más de un registro por dirección.
    b) Tamaño de la cubeta: más de un registro por dirección, al tener mayor tamaño hay menor overflow pero mayor fragmentación.
    c) Densidad de empaquetamiento: proporción de espacio del archivo asignado que en realidad almacena registros. A menor densidad hay menos colisiones pero más espacio desperdiciado. DE = nro de registros/capacidad de archivo.
    Se puede estimar el overflow con la función de Poisson, que nos da la probabilidad que un nodo tenga I elementos.
    Siendo R la cantidad de registros del archivo, N la cantidad de cubetas y C la capacidad del nodo tenemos que: P(I) = (((R/N)^I) * e^-(R/N))/(I!)
    d) Métodos de tratamiento de desbordes (overflow):
        i) Saturación progresiva: almacenar registros en direcciones más próximas siguientes para no impedir la búsqueda de registros, a los borrados se los marca con #.
        ii) Saturación progresiva encadenada: igual al anterior pero se enlazan los nodos que realizaron overflow.
        iii) Dispersión doble: almacena registros en zonas no relacionadas aplicando una segunda función de hash para sumar un

desplazamiento a la dirección original reiteradas veces hasta encontrar un lugar libre.
        iv) Área de desbordes por separado: los registros en overflow van a otro archivo donde se enlazan entre sí.
6) Problemas con hashing con espacio de direccionamiento estático:
    a) Como hay direcciones fijas, cuando el archivo se llena la única solución es hacer un archivo más grande y re-dispersar, lo que lleva a muchos cambios.
    b) Saturación excesiva.
7) Hashing con espacio de direccionamiento dinámico:
    a) Reorganizar tablas sin mover muchos registros
    b) Técnicas que asumen bloques físicos, que pueden utilizarse o liberarse. Entre ellas encontramos: hash virtual, hash dinámico y hashing extensible.
    c) Hashing extensible: adapta el resultado de la función de hash de acuerdo al número de registros que tenga el archivo y de las cubetas necesitadas para el almacenamiento.
        i) La función genera secuencias de bits (generalmente 32).
        ii) Forma de trabajo:
            (1) Se utilizan sólo los bits necesarios de acuerdo a la instancia del archivo.
            (2) Los bits tomados forman la dirección del nodo a utilizar.
            (3) Si hay overflow, hacer un nuevo nodo, reubicar registros y tomar un bit más para direccionamiento.
            (4) La tabla auxiliar tiene tantas entradas como 2^i donde i es el número de bits actuales para el sistema.

Conclusiones:

1) Accesos y organización:
    a) Acceso a un registro por clave primaria:
        i) Ninguna: muy ineficiente (lento).
        ii) Secuencial: poco eficiente (lento).
        iii) Secuencial indizada: muy eficiente (buena).
        iv) Hash: el más eficiente (rápido).
    b) Acceso a todos los registros por clave primaria:
        i) Ninguna: muy ineficiente (lento).
        ii) Secuencial: no eficiente (rápido).
        iii) Secuencial indizada: el más eficiente (rápido).
        iv) Hash: muy ineficiente (lento).
2) La elección de la organización va a depender de los requerimientos del usuario. Examinar:
    a) Características del usuario, cantidad de registros y tamaño de cada uno.
    b) Requerimientos:
        i) Tipos de operaciones y cantidad de accesos.
        ii) Características del hardware.
        iii) Parámetros.
        iv) Tiempo de desarrollo y mantenimiento.
        v) Uso promedio: # de registro usados / # de registros.

Hiper Leo Resumen:

La búsqueda secuencial es muy deficiente y la mayor cantidad de operaciones son consultas, por lo que tenemos que centrar la mejora en esto. Además mantener el orden físico es muy costoso debido a los tamaños de los archivos y la cantidad de accesos a discos, que es mucho inferior a la velocidad de RAM.

La primera gran mejora son los índices como tabla, permitiendo orden no físico de los registros, y luego su implementación con árboles, hasta lograr los arboles balanceados, que se construyen de abajo para arriba, donde primero encontramos a los B y B* (que son parecidos), y a los B+ que permiten, además de acceso rápido, un buen recorrido secuencial.

En caso de necesitar eficiencia de búsqueda EXTREMA, sacrificando el recorrido secuencial, tenemos las técnicas de Hashing (dispersión para hispanohablantes), que las hay estáticas, que causan problemas por falta de espacio al ser fijas, y dinámicas, que aseguran siempre un acceso con la contra de tener mucho procesamiento de overflow.

Bueno, cualquiera de estas técnicas es siempre mejor que no tener ninguna forma de organización.