Ficheros Y Bases de Datos Tm1 - 8

Ficheros y Bases de Datos (Universidad Carlos III de Madrid)

Resumen de Ficheros y Bases de datos

CONTENIDO
TM1: Introducción a las BB.DD y SGBD. Archivo TM2: Estática del Modelo Relacional Archivo TM3: Dinámica del Modelo Relacional Archivo TM4. Modelo Relacional Avanzado Archivo TM5. Introducción y Conceptos básicos de ficheros TM6: Organizaciones Base TM7: Organizaciones auxiliares TM8: Acceso multiclave
Toda la información de éste documento está extraída de las diapositivas proporcionadas en la asignatura de F&BD de la universidad Carlos III de Madrid en el año 2014. He tratado de incluir toda la información posible para no perder ningún concepto (no es un resumen), pero confío en que el formato ayude a su comprensión. Lo ideal sería completarlo con información de algún libro, dado que en las diapositivas a menudo no se profundiza en muchos temas importantes.

Jorge Trincado
Jtrinc26@gmail.com

TEMA1: INTRODUCCION A LAS BASES DE DATOS:

- **Conceptos básicos**:
  - **Información**: comunicación o adquisición de conocimientos que permiten ampliar o precisar otros conocimientos que ya tenemos sobre una materia determinada.
  - **Dato**: representación de una información de manera adecuada para su tratamiento (informatizado). Se obtienen como resultado de tomar un suceso físico y normalizarlo mediante un convenio.
  - **Procesamiento de datos**: Para poder obtener un beneficio a partir de los datos realizamos procesos de análisis sobre los mismos. Para poder realizar dicho proceso, es necesario que los datos se encuentren almacenados.

  - **Almacenamiento**: El guardado de datos debe permitir inmediata disponibilidad, tener una perdurabilidad y un alcance. El almacenamiento puede ser permanente, masivo o portable. Existen dos tipos de almacenamiento: el principal y el secundario:
    - Principal: volátil, caro, capacidad reducida, rápido, con acceso privilegiado -- RAM
    - Secundario: permanente, barato, capacidad elevada, lento, externo -- HDD

  - **Tipos de soporte**: según el tipo de acceso (serial, secuencial, direccionado) o según sus características básicas (unidad de acceso, tiempo de acceso a bloque, capacidad, tipo de acceso y operaciones, coste…

  - **Fichero**: Conjunto organizado de informaciones almacenadas en un soporte común. (conjunto organizado y nominado de informaciones estructuradas almacenadas en un soporte común).

  - **En foques de almacenamiento:** dos tipos: Clásico (orientado a ficheros), son los sistemas de ficheros, y actual (orientado a los datos) son los sistemas de bases de datos.

  - **Base de datos**: colección de datos integrados con redundancia controlada, con una estructura que refleja las interrelaciones y restricciones del mundo real. Los datos deben ser independientes de la aplicación o el usuario que los muestre, y tendrán una definición y descripción única. Debe estar dotada de unos procedimientos que siempre preservarán la integridad de la base de datos, respetando además ciertas normas de disponibilidad y confidencialidad.

  - **La arquitectura ANSI/SPARC**: propone una estructura para las bases de datos en tres niveles:
    - Nivel interno: registros (representación física). Se describe mediante un esquema interno que muestra las correspondencias entre los datos y los soportes (útil para el Sistema operativo).
    - Nivel conceptual: Visión global de la estructura de datos, se describe mediante un esquema conceptual entre unos datos y otros (para los diseñadores).
    - Nivel externo: visión de la base según cada usuario, queda descrito gracias a un esquema externo entre los datos y los usuarios (es decir es la estructura empleada para mostrar la información al usuario).

  - **Sistema gestor de bases de datos**: Es un conjunto coordinado de herramientas que proporciona los medios necesarios para interaccionar con la base a todos los niveles. Cuenta con una serie de herramientas (programas, procedimientos, lenguajes) para interactuar con la base de datos y facilitar las tareas de describir, recuperar y manipular datos, preservando su integridad, confidencialidad y seguridad.

    Las funciones esenciales de un SGBD son:
    - Descripción: debe permitir definir los elementos de datos y su estructura así como las interrelaciones entre ellos y las reglas de validación semántica.
    - Manipulación: ha de posibilitar la operación del contenido de la base.
    - Utilización: tiene que incluir un conjunto de herramientas a través de las cuales el administrador pueda desarrollar su labor.

TM2-ESTATICA RELACIONAL:

☐ Conceptos básicos:

- Modelo o universo del discurso: Son los requisitos y restricciones que debemos representar en un grafo relacional mediante las herramientas del modelo relacional. Tiene propiedades estáticas y dinámicas.

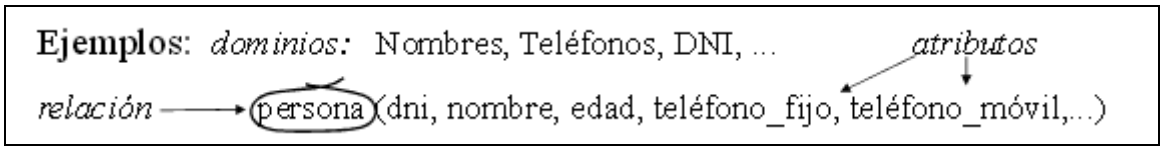
- Modelo de datos: Es un lenguaje orientado a describir los elementos de la realidad que intervienen en un problema determinado y la forma en la que estos elementos se relacionan entre sí. Permite describir:
  - Las estructuras de datos: El tipo de datos y la forma en la que se relacionan.
  - Las restricciones de integridad: un conjunto de condiciones que deben cumplir los datos para reflejar correctamente la realizad deseada. Existen dos tipos:
    - Condiciones inherentes: impuestas sobre la estructura del modelo:
    - Condiciones semánticas: Impuestas a determinados valores o asociaciones de valores. Las rest. Semánticas se deducen de supuestos semánticos explícitos o implícitos. Garantizan la integridad de la base y la validez semántica de su contenido.
      Es decir son lo que evita que la información guardada tenga significados ilógicos.

Cuando modelamos el universo del discurso creamos estructuras con las propiedades estáticas y operadores don las propiedades dinámicas.

☐ Estática relacional

- elementos:
  - Dominio: conjunto de valores de la misma naturaleza. Son los datos que conforman la información de una ocurrencia en una base.
  - Relación: subconjunto del producto cartesiano de n dominios. Es decir, una ocurrencia final, localizada mediante la concurrencia de al menos dos atributos.
  - Atributo: Propiedad común a los elementos de una relación. Se define sobre un dominio para restringir sus valores. Por ejemplo "teléfono" se define sobre el dominio de los números naturales de 9 dígitos.

- Conceptos de diseño:
  - Los dominios, relaciones y atributos concretos deben ser identificados mediante una <u>etiqueta</u>
  - Las etiquetas no pueden repetirse dentro de un mismo ámbito, aunque si en distintos.
  - Dominios y relaciones tienen una existencia independiente, no así los atributos que deben existir para un dominio determinado.



- Relación en Estática relacional: Es un conjunto de ocurrencias que representa y define una relación de individuos pertenecientes al mundo real.
  La ocurrencia de una relación se representa mediante una túpla (asociación de valores ordenados según un esquema de relación que mantiene una correspondencia biunívoca con un individuo del mundo real.
  - Def. por intensión: definición invariable de la semántica de una relación (esquema), por ejemplo: COCHE(MATRICULA,NºSERIE,MARCA,MODELO,AÑO)
  - Def. por extensión: conjunto de tuplas de una relación en un momento del tiempo.

- Clases de relación:
  - Persistentes: solo se borran con una acción explícita del usuario.

    1-relaciones base (las que se corresponden con el nivel lógico).

    2- vistas (que se corresponden con el esquema externo, sus datos se almacenan en otras relaciones, presentando redundancia lógica)

    3-las vistas materializadas (son como las vistas pero tienen datos almacenados y presentan redundancia física)

    4-las instantáneas (fotografía de una tabla en un momento determinado del tiempo.
  - Temporales: Desaparecen al ocurrir un determinado evento (sin especificar una acción de borrado) por ejemplo al acabar una sesión o una transacción.

- Propiedades de una relación:
  - El número de atributos se denomina grado
  - El número de túplas se denomina cardinalidad.

- Claves de identificación:
  - Clave candidata: atributo o conjunto de atributos mínimo que identifican unívocamente cada túpla de una relación.
  - Clave primaria: clave candidata privilegiada, se representa subrayada.
  - Clave secundaria: resto de claves candidatas, si se usa alguna se subraya en discontinuo.
  - Clave ajena: atributo (o conjunto) de una relación que se asocia con otro atributo de otra relación y su valuación se restringe a valores existentes en el atributo referenciado o al nulo (en caso que no se cumpla ninguna relación).Cuando realizamos borrados de datos debemos tener en cuenta que lo referenciado por una clave ajena debe existir.

- Restricciones inherentes:
  - El orden de las túplas y de los atributos no es significativo.
  - No pueden existir dos túplas iguales.
  - Cada atributo toma un solo valor del dominio en cada túpla.
  - Siempre existe una clave primaria y ninguno de sus atributos puede ser nulo.
  - Los atributos en los que el valor nulo es aplicable se marcan con el símbolo *
  - La integridad referencial declara que lo referenciado por clave ajena debe existir.

- Restricciones semánticas:
  - La elección de la clave primaria se expresa como (PRYMARY)
  - La elección de la clave alternativa si la hay se expresa mediante (UNIQUE)
  - La obligatoriedad se expresa mediante (NOT NULL)
  - Las reglas de integridad referencial son las reglas que se debe naplicar cuando exista el riesgo de romper la integridad referencial (que los datos dejen de tener sentido) en operaciones de borrado y de modificación. Se expresan mediante:

    -**R**-restrict (operación restringida) si la operación involucra padres con hijos no se lleva a cabo.

    -**NA-**No Action (operación cancelada) si la operación rompe la integridad referencial no se lleva a cabo.

    -**C**-CASCADE (propagación en cascada) Los calores no validos serán actualizados también en la clave ajena.

    -**SN**-Set Null (puesta a nulo) los valores no validos serán sustituidos por el valor nulo.

    -**SD-**Set Default (valor por defecto) los valores no validos serán sustituidos por un valor por defecto.

- Restricciones semánticas de rechazo: se utilizan para descartar las operaciones de inserción, modificación o borrado que no cumplen una determinada condición. Hay dos tipos: simple (se aplica a un solo elemento) y asertion (general a toda la BD).

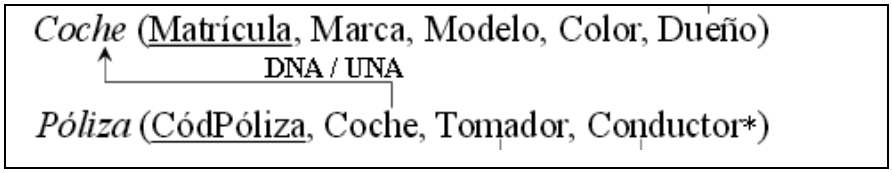
☐ Diseño relacional:

- Relaciones entre esquemas de relación: Para interrelacionar dos esquemas, se implementa un puntero lógico en uno de ellos, siempre en la clave primaria (para que identifique unívocamente dicha relación).

  La clave de la relación referenciada se denomina clave ajena de la relación referenciante.

- Tipología de las relaciones entre esquemas de relación en función del número de túplas que intervienen entre esquemas de relación.
  - 1 a 1 correspondencia biunívoca:

    Se podría fusionar ambos esquemas en uno solo, la clave ajena se puede posicionar en cualquiera de los dos, o bien pueden compartir la clave primaria.
  - 1 a n: correspondencia múltiple: Un esquema es padre de otro.
    - La clave ajena se sitúa en la relación que participa con múltiples túplas mediante propagación de clave.
    - EJ: un coche puede tener varias pólizas de seguros:



  - N a N: es necesaria una relación intermedia.

- Para toda interrelación se debe especificar que regla ha de aplicarse en cada acción que haga peligrar la integridad (borrado o actualización).

- El conjunto de todos los esquemas de relación de un a BDR con sus interrelaciones y RI que aplican, se denomina esquema relacional.

- El diseño básico se puede complementar añadiendo semántica mediante vistas y disparadores.7

TM3-DINMÁMICA RELACIONAL.

- Dinámica del modelo relacional: Una base de datos es una colección de conjuntos relacionados y bajo algún tipo de restricción. Las operaciones que se realizan sobre ella y alteran las relaciones o producen nuevos estados en la base de datos hacen que la base de datos se dinamice. Para introducir esos cambios es necesario utilizar un lenguaje de especificación, existen dos tipos:

ALGEBRA RELACIONAL SIMPLE		
Nombre	**Símbolo**	**descripción**
SELECCIÓN	$\sigma_{predicado}(Tabla)$	Genera una tabla con aquellas túplas en las que el atributo coincida con el especificado.
$\sigma_{Autor='Dumas'}(libros)$ => seleccionará de la tabla libros, todos los que en 'autor' tengan a Dumas		

PROYECCIÓN	$\prod_{atrib1,atrib2}(Tabla)$	Recoge un subconjunto de una relacion, y genera una nueva relación con los atributos seleccionados.
$\prod_{titulo,autor}(Relacion)$ =>genera una tabla en la que sólo aparecen los atributos seleccionados.		

RENOMBRADO	$\rho_{símbolo}(expresión)$	Asigna el resultado de una expresión a un símbolo, se utiliza para acelerar el trabajo.
$\rho_A(\sigma_{Autor='Dumas'}(libros))$ => mete en el resultado de la función A $\equiv \sigma_{Autor='Dumas'}(libros)$		

UNIÓN (ó)	Tabla1 $\cup$ Tabla2	Genera una tabla con todas las túplas compatibles de dos tablas distintas eliminando las repetidas.
Aventuras $\cup$ Novelas => Se generará una tabla con las túplas de la tabla aventuras y con las de la tabla novelas, eliminando las repetidas. OJO! Solo tablas compatibles (mismos dominios)		

INTERSECCIÓN (ý)	Tabla1 $\cap$ Tabla2	Genera una tabla con todas aquellas túplas que estén en ambas tablas. Solo tablas compatibles!
Aventuras $\cap$ Novelas =>se generará una tabla con aquellos libros que sean novelas de aventura		

DIFERENCIAS	Tabla1 − Tabla2	Genera una tabla con las túplas que aparecen en la primera pero no en la segunda. Solo compatibles!
Novelas ∩ Aventuras => genera tabla con todas las novelas que NO sean de aventuras.		

PROD.CARTESIANO	Tabla1 **x** Tabla2	Genera una tabla con todas las posibles combinaciones de las túplas de ambas tablas.
Comidas **x** Vinos => producirá una combinación de todas las posibles comidas con cada uno de los posibles vinos que se ofrecen en el restaurante.		

COMBINACIONES	Tabla1 $\theta_{atributoT1(=,>,<)atributoT2}$ Tab2	Genera una tabla con las túplas del producto cartesiano que cumplen una condicional genérica.
(Clientes) $\theta_{Edad \geq Calificación}$(Películas) => producirá una tabla con todas las posibles películas que pueden ver los clientes que superen la edad requerida.		

COMB.NATURAL	Tabla1 $*_{atributo}$ Tabla2	Caso particular de combinación en el que la comparación es de igualdad.
Empleados $*_{dueño}$ Coches => producirá una tabla con las túplas en las que el nombre de empleado sea igual al dueño del coche. No hace falta poner “nombre” antes de empleados porque no hay confusión posible, dado que el DNI es numérico en el ejemplo de la diap13		

OJO!! Para poder aplicar los operadores de conjuntos Unión, Intersección y Diferencia , las relaciones deben ser compatibles. Dos relaciones son compatibles si tienen el mismo número de atributos y el atributo i-ésimo del primer operando está definido sobre el mismo dominio que el atributo i-ésimo del segundo operando. En general, el esquema de relación de dos relaciones no coincide pero puede aplicarse la operación de proyección para igualar estos esquemas

ALGEBRA RELACIONAL EXTENDIDA

Nombre	Símbolo	descripción
AGRUPACION	π_{criterio} **Group By** $_{\text{atributo}}$ (tabla) posibles criterios aplicables : count(), sum(), avg(), min(), max()	Forma de grupos según un conjunto de atributos seleccionados bajo un criterio. Puede ir precedido de una proyección y una selección.
$\pi_{\text{Count ('x')}}$ Group by $_{\text{Nacionalidad}}$ (Personas) => produce una tabla en la que se contabilizan el número de túplas de cada nacionalidad que hay en la tabla personas.		
DIVISION	relacion1 ÷ relacion2	Da como resultado todas la túplas que concatenadas con cada túplas del divisor estén contenidas en el dividendo.
Currícula ÷ $\pi_{\text{requisito}}(\sigma_{\text{perfil='Analista/Des'}}$ Puestos)		
SEMI-COMBINACION	Tabla1 ***|**$_{\text{atributo}}$ tabla2	Igual que la combinación, pero solo se toman las columnas del operando izquierdo |* o del derecho *|
Empleados ***|**$_{\text{dueño}}$ coche => empleados con coche Empleados **|***$_{\text{dueño}}$ coche => coches de los empleados		
ANTI-COMBINACION	Tabla1 $\triangleright_{\text{atributo}}$ tabla2 Tabla1 $\triangleleft_{\text{atributo}}$ tabla2	Igual que la semi-combinación, pero las túplas que se incluyen son las que no cumplen la condición definida
Empleados $\triangleright_{\text{dueño}}$ coche => empleados sin coche Empleados $\triangleleft_{\text{dueño}}$ coche => coches que no son de empleados		
COMBIN. EXTERNA	Tabla1 **]*** $_{\text{atributo}}$ tabla2 Tabla1 ***[** $_{\text{atributo}}$ tabla2 Tabla1 **]*[** $_{\text{atributo}}$ tabla2	Extensión de la combinación que incluye las túplas que no encajan de la relación seleccionada: left outer join]*, rigth outer join *[, full outer join]*[Las columnas que no aplican adoptan el valor nulo.
Empleados **]***$_{\text{dueño}}$ coche => empleados y sus coches si los tienen Empleados ***[**$_{\text{dueño}}$ coche => coches y sus dueños que son empleados Empleados **]*[**$_{\text{dueño}}$ coche => empleados y coches asociados como proceda		
ORDEN	$\pi_{\text{first,last,rank}}$ **OrderBy**$_{\text{atributo}}$ (tab) para el orden inverso: **ORDER BY DESC**	Se puede obtener un conjunto ordenado como resultado de aplicar un orden sobre una relación. Aporta orden de < en nºs y lexicográfico en caracteres. -Además sobre una lista ordenada se pueden conocer el primero, ultimo y la posición con : first, last, Rank(n)

$\pi_{\text{first,last,Rank(29)}}$ Order By $_{\text{edad}}$ (personas) =>

first	*last*	*rank(29)*
3	73	4

ACTUALIZACION	Tabla←operación	Se utiliza para definir operaciones de actualización sobre la base de datos.
Personas ←Personas –($\sigma_{\text{nombre='José'}}$(Personas))		

☐ El lenguaje SQL:

- Operaciones de actualización de la base de datos:
  - Inserción (INSERT INTO): La inserción incluirá todas o sólo una parte de las columnas.

    -Para una inserción total debemos tener en cuenta que el orden (cantidad) de los valores debe coincidir con el de la definición de las columnas. Y si algún valor es nulo, se debe especificar con la constante NULL.

    -En una inserción parcial de columnas, se debe utilizar algún tipo de criterio para seleccionar las columnas deseadas (definiendo un esquema de inserción). De esta forma podemos realizar inserción de túplas individualmente en una tabla determinada (esto da pie a generar consultas).

    - Borrado de túplas (DELETE [FROM] 'nombre tabla' [WHERE] 'criterio de selección.' Esto eliminará las túplas que cumplan el criterio de selección de la base de datos.

  - Modificación de túplas (UPDATE nombreTabla SET columna = nuevoValor WHERE crit.Seleccion.)

- Querys: Las consultas se generan mediante la creación de nuevas tablas siguiendo un criterio de selección. Los datos aportados pueden ser extraidos del área de trabajo o bien proceder del sistema (como por ejemplo SYSDATE), conversiones, codificaciones, extensiones…
  - Operaciones matemáticas: Pueden ser simples sumas, restas, productos…

    -podemos operar empleando fechas: -sysdate-1 (fecha de ayer) , sysdate-fecha_ini (número de días transcurridos desde la creación de la base de datos).

    -otras funciones: redondeos (round, floor, ceil, trunc), signo (sign, abs), otras (sqrt, log, ln)

  - Operaciones con cadenas de caracteres: concatenación (concat), longitud (length), substring (substr), reemplazamiento de caracteres (replace).

    -operaciones de conversión de caracteres: to_char(num, plantilla), to_number(char, plantilla), to_date (char, plantilla) siendo la plantilla una cadena con el resultado entre comillas simples.

    -operaciones de codificación/decodificación de caracteres: CASE (elige el valor a devolver) DECODE (sustitulye los valores por otros valores) NLV(sustituye null por otro valor) COALESDE (de una serie de atributos devuelve el primer valor no nulo).

  - Expresiones condicionales: (WHERE) pueden utilizarse para:

    -Comparaciones (>,<,=,>=,<=)

    -test de inclusión, ya sea en conjunto (expresión [NOT] IN exp_lista) o como rango (expresión NOT BETWEEN exp AND exp). Si no se incluye el NOT el resultado será el constrario.

    -test de valor nulo (exp IS [NOT] NULL)

    -test de semejanza (exp [NOT] LIKE patrón)

    -test de existencia (EXIST subquery)

    -operación lógica sobre otras condiciones (NOT, AND, OR)

  - Clausula FROM: Se utiliza para definir el área de trabajo, que puede componerse de una o varias tablas combinadas. También puede ocurrir que el área de trabajo sea otra consulta (subquery).

TM4: MODELO RELACIONAL AVANZADO:

☐ PL/SQL: Es un lenguaje orientado al usuario. Algunas de sus instrucciones útiles son:

- Gestión de transacciones: COMMIT (realizar), ROLLBACK (deshacer), SAVEPOINT (guardar)
- Gestión de privilegios: GRANT (conceder) y REVOKE (revocar).
- Este lenguaje combina las sentencias SQL con el manejo de un lenguaje de programación procedimental , lo que amplía la potencia del Sistema gestor de bases de datos (SGBD).
  Los bloques PL/SQL tienen tres partes: Declaraciones, cuerpo y excepciones.
- Cuando ocurre un error dentro de un bloque PL/SQL, se levanta una excepción, que es gestionado por el usuario o por un paquete (en oracle: DBMS_STANDARD.RAISE_APPLICATION_ERROR)
- Dentro del cuerpo, todas las funciones deben incluir una intruccion "return(valor)".
- Las invocaciones a procedimientos almacenados deben hacerse siempre desde dentro de un bloque (desde otro procedimiento, un disparador, etc).
- Podemos crear colecciones de variables y procedimientos denominados paquetes, para facilitar un fácil manejo. Estos paquetes primero se describen y luego se implementan. Para crear un paquete empleamos:
  CREATE OR REPLACE PACKAGE nombre AS declaración de variables, cabeceras…. END nombre
  CREATE OR REPLACE PACKAGE BODY nombre AS descripción del procedimiento…END nombre.

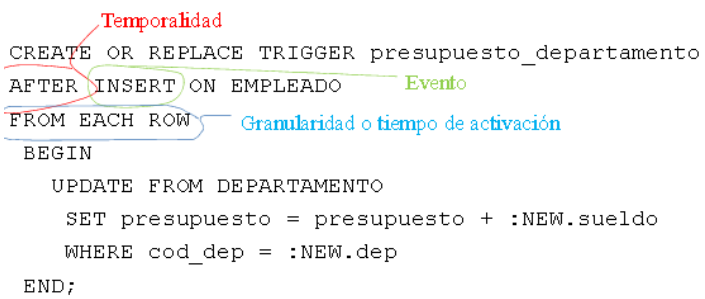
☐ Vistas:

- Una vista es una consulta accesible como una tabla virtual. Las vistas se realizan sobre las tablas fuente de la base de datos, por eso los datos que se recuperan en una vista tienen la misma estructura que dicha tabla.
- Al igual que ocurre con una tabla, se pueden insertar, actualizar, borrar y seleccionar datos de una vista, y dichos cambios afectaran a los datos guardados en la tabla.
- Las columnas obligatorias, si son omitidas deberán adquirir un valor por otros medios (un valor por defecto o mediante un disparador).
- La vista materializada tiene datos almacenados. Y las "snapshot" son materializadas y constantes.
- Si empleamos la función WITH CHECK OPTION, No se permitirán las actualizaciones de la vista si no se cumple una condición determiada. (si ocurriera se cancelaria la acción).

☐ Disparadores:

- Se incluyen en el estándar SQL3 de 1999.
- Son procedimentales (no declarativos). Es un procedimiento que se ejecuta cuando se cumple una condición establecida el realizar una operación. Pueden ser de inserción, actualización o borrado, según lo defina el usuario (a diferencia de las restricciones de rechazo, que son definidas por el sistema).
- Estructura SQL:
  CREATE OR REPLACE TRIGGER [nombre] <Tiempo de activación de la acción><Eventos de disparador>
  ON <nombre_de_tabla> <nivel de activación><bloque definiendo la acción disparada>

  - El tiempo de activación de la acción del disparador se utiliza para definir que la acción ocurra AFTER, BEFORE, INSTEAD OF (esta última se utiliza asociada a una vista para realizar operaciones de actualización que no están permitidas en las vistas).
  - El evento indica qué debe hacer el disparador (INSERT, DELETE, UPLOAD)
  - El nivel de activación indica la granularidad de la acción (para cada columna, para una columna, etc).




```
CREATE OR REPLACE TRIGGER presupuesto_departamento
AFTER INSERT ON EMPLEADO
FROM EACH ROW
 BEGIN
   UPDATE FROM DEPARTAMENTO
    SET presupuesto = presupuesto + :NEW.sueldo
    WHERE cod_dep = :NEW.dep
 END;
```

TM5 –INTRODUCCIÓN A FICHEROS Y ALMACENAMIENTO SECUNDARIO:

- **Fichero:** Conjunto organizado y nominado de informaciones estructuradas almacenadas en un soporte no volátil → conjunto de informaciones almacenadas en un soporte común.
  Los datos almacenados se corresponden con la información que nos rodea, pero podemos distinguir dos ámbitos:
  - Cuando dicha información proviene de ocurrencias del mundo real, que son normalizadas mediante un enfoque lógico. Al organizarlas generamos Archivos.
  - Cuando la información trata de sucesos físicos registrados, normalizados mediante un enfoque físico. Al almacenarlos en un soporte físico tras la normalización obtenemos Ficheros.

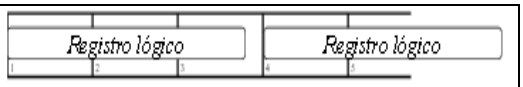
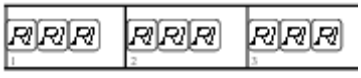
- Estructuras de un fichero: Podemos diferenciar dos tipos:
  - La estructura **lógica**: Que muestra la organización de los datos para los usuarios.
  - La estructura **Física**: Que muestra la organización de los datos para el soporte en que se guarda

  Esto presenta un problema de optimización entre la **eficacia** de una organización realista que cubra las necesidades de los usuarios y la **eficiencia** de una organización práctica a nivel de recursos.

- El contenido de un fichero: Los ficheros están formados por con juntos de registros, en los que almacenamos la información que queremos describir. Podemos diferenciar dos tipos de registros:
  - Registros **lógicos**: -Son los datos referidos a un elemento del fichero.
    -Es la unidad mínima de proceso de una aplicación.
    -Su tamaño viene definido por el número de caracteres que contiene.
    -También se denomina **registro**. (es el autentico registro en esta materia)
  - Registros **físicos**: -Son los caracteres que se almacenan y se acceden juntos.
    -Es la unidad mínima de acceso al soporte de almacenamiento.
    -Su tamaño viene definido por el número de caracteres que caben.
    -También se denomina **bloque**.

- Unidades mínimas de almacenamiento (sub-atómicas):
  - Enfoque **lógico**: "**Campo**" Es la unidad mínima interpretable. Los campos tienen asociado un tipo de campo que es una abstracción que describe sus características en la realidad. Ej: entero
  - Enfoque **Físico**: Es la unidad mínima registrable, dependiendo del soporte puede tratarse de "**caracteres**" en memoria privilegiada o de "**bloques**" en soportes secundarios.

- Correspondencia a nivel físico-lógica a nivel de registro: Los registros físicos y los registros lógicos no suelen coincidir en tamaño. Distinguimos dos ocurrencias:

"**Registros expandidos**" cuando el Rl excede el tamaño de los Rf que lo albergan.	
"**Bloque**" Cuando el Rf es mayor que el Rl, lo que permite guardar varios Rl en cada Rf.	

A este nivel, los ficheros pueden quedar estructurados de dos maneras:

Organización consecutiva: todo registro lógico comienza siempre a continuación del anterior. *Se producen **BLOQUES** *Ocurre en ficheros seriales y secuenciales	
Organización no consecutiva: Todo registro físico comienza con el inicio de un registro lógico. *Se producen **CUBOS**	

- El concepto de CUBO: Conjunto de bytes con una condición de acceso común (comparten dirección, información de control, organización, espacio libre…), que suele procesarse como una unidad. Dentro de un cubo la organización de los registros es consecutiva, en cambio, entre cubos es no consecutiva.
    - El **espacio de cubo (Ec)** es la cantidad de información asignada al cubo (medida en bloques, bytes, MB). Habitualmente es un múltiplo del tamaño de bloque si es mayor que éste o, en caso que sea menor, se denomina 'celdas' a las diferentes divisiones del cubo.
      *La memoria intermedia suele paginarse al espacio de cubo.
    - El **tamaño de cubo (Tc)** son en número de registros lógicos que contiene el cubo, descontando la información de control y el espacio libre distribuido. $T_c = E_c/T_{reg}$

- Correspondencia físico-Lógica a nivel de operación:
  Existe una correspondencia a nivel práctico que conecta las estructuras lógica y física, mediante una aplicación (alto nivel) o bien mediante el SO (bajo nivel).

☐ **Diseño de ficheros:**

- **Diseño lógico**: Descripción de la estructura lógica de los registros de un fichero. Es decir, es la descripción de la disposición de los elementos de un registro, que en conjunto definen un individuo. El tipo de elementos es una abstracción de un campo o un agregado que describe las características del objeto de interés (Consta de un tipo de datos y una longitud):
    - Elementos de datos: (campo) La unidad mínima de datos que interviene en un proceso.
        - Notación: Campo tipo (tamaño) , si es opcional se encerrará entre corchetes [].

    - Agregado de datos: Colección de elementos en forma de:
        - **Vector**: agregado compuesto por un número fijo de elementos cuya interpretación es complementaria (entre todos los elementos definen un concepto).
            - Notación: (elmto1, elmto2,… elmtoN) → *fecha (año N(4)'-'mes N(2)'-'dia N(2))*

        - **Grupo repetitivo:** Agregado compuesto por un número fijo o variable de elementos cuya interpretación es común. Es decir se utiliza el mismo concepto para definir cada elemento.
            - Notación: (elemento)* → de 0 a N elementos.
              (elemento)+ → de 1 a N elementos.

  *Una **Ocurrencia** es el valor concreto que toma un tipo de elemento (referido a un individuo).

Ejemplo: Definimos los tipos de datos de un individuo siguiendo la estructura <<Nombre Tipo de dato (longitud)>> y luego le damos valores:		
Nombre Char(8)	→	JORGE
DNI Number(8)	→	12345678

    - Objetivos: Ofrecer una representación fiel de la realidad, una organización física transparente al usuario que facilite la interacción con los datos y evitar redundancias lógicas.

- **Diseño físico:** Determinación de la organización física de un fichero. Es decir, de la disposición de los registros en el soporte, relativa a su implementación, orden, direccionamiento, punteros, etc. El diseño físico depende en gran medida del soporte en el que se implementa, el tamaño de bloque ofrecido, el tiempo de localización y acceso, y la tasa de transferencia.
    - **Objetivos:** Principalmente la **eficiencia**, gastar la menor cantidad de recursos posible para un mismo resultado. Para ello, un diseño físico debe disminuir el espacio ocupado, lo que también reduce el tiempo de respuesta dado que se minimizan los accesos. También es importante lograr un bajo coste de reorganización, desarrollo y modificación.

- **Optimizaciones a bajo nivel:**
  - Buscar soportes adecuados, que ofrezcan un tamaño de bloque y tipo de acceso adecuados.
  - Minimizar el tiempo de localización (distribución adecuada)
  - Introducir redundancia física (grabar lo mismo varias veces)
  - Uso de memorias intermedias

- Principios de diseño de registros físicos: Los factores que influyen en la organización del fichero son la actividad que resulte de cada tipo de proceso, la volatilidad y el crecimiento del fichero.
  Los medimos mediante:

Tasa de actividad	% de registros tratados en un proceso	$T_A = \frac{\text{registros procesados}}{\text{registros totales}} \cdot 100$
Tasa de volatilidad	% de cambios por unidad de tiempo, distinguimos tres tipos:	
	Tasa de inserción	$T_I = \frac{\text{registros insertados}}{\text{registros totales}} \cdot 100$
	Tasa de borrado	$T_B = \frac{\text{registros borrados}}{\text{registros totales}} \cdot 100$
	Tasa de modificación	$T_M = \frac{\text{registros modificados}}{\text{registros totales}} \cdot 100$
Tasa de crecimiento	Proporción en la aumentan los registros del fichero por unidad de tiempo. Permite reservar espacio para los nuevos registros en función de la organización del fichero.	$T_C = T_I - T_B$
Tiempo de acceso a bloque	Tiempo necesario para acceder a un registro determinado.	$T_{bloque} = T_{localizacion} + T_{transferencia}$
Tiempo de acceso a registro aleatorio	Frecuentemente utilizado para comparar organizaciones, evaluando su coste temporal.	$T_{aleatorio} = Nº_{accesos} \cdot T_{bloque}$
Coste Global	Cada proceso que accede a un sistema de ficheros presenta un coste que depende de la organización. El coste de todos los procesos se define como:	$C(O_k,P) = \sum Ci \cdot fi$
Medidas de Espacio ocupado y útil	**Volumen**: número de caracteres ocupados. **Ocupación útil**: caracteres útiles del registro. Volumen >= ocupación útil	→Tamaño real de un registro. →Tamaño medio de un regist.
	Densidad ideal de un registro: Relación entre cantidad de información útil y cantidad de información almacenada. Disminuye al aumentar el espacio desperdiciado y la información de control.	$d_i = \frac{\text{ocupación útil}}{\text{volumen real}}$
	Densidad Real de un fichero: relación entre la cantidad de información útil y la cantidad de información almacenada. Referida a un soporte y a una organización. Siempre **di >= dr**	$d_r = \frac{\text{ocupación fichero}}{\text{volumen fichero}} = \frac{\text{ocupación/reg} * \text{nºregistros}}{n\,\text{bloques} * T_{bq}}$
	Densidad de ocupación de un fichero no consecutivo: Relación entre la cantidad de registros almacenados y la cantidad de ellos que caben en el soporte. Mide el % de espacio potencialmente utilizado.	$d_{oc} = \frac{\text{nºregistros}}{N\,\text{cubos} * T_c}$

- **<u>Diseño físico-lógico:</u>** Es la descripción de las cadenas de bytes utilizadas para almacenar registros, y de los convenios necesarios para su implementación. Por otro lado el "Diseño físico del registro lógico" → Es la implementación de un registro lógico en secuencias de bytes.
  El objetivo principal de éste diseño es reducir el espacio, proporcionando un menor número de accesos y mejorando así la eficiencia dado que cuanto menor es el volumen real, mayor es la densidad.

  - Optimización mediante **campos de control (marcas)** mejoran el manejo en soporte. Las marcas de elemento de datos ahorran espacio evitando el padding. Dichas marcas pueden ser de varios tipos:
    - Asociadas a elementos de datos:
      - De existencia: En campos opcionales indican si aparece o no.
      - De longitud: En campos de tamaño variable indican el número de caracteres.
      - De reiteración: En grupos repetitivos indica el número de ocurrencias.
      - De fin de campo: En campos indefinidos indican que acaba.

    - Asociadas a Registros:
      - Fin (inicio) de registro: Separa registros consecutivos en el soporte.
      - Tipo: Indica el tipo de registro a continuación (cuando hay ficheros heterogéneos).
      - Mapa: Indica los registros que se aplican (cuando hay ficheros heterogéneos).

  ** Importante ver y hacer ejemplos de cálculo de densidades con y sin marcas… Cae fijo.

  - Optimización mediante **codificación de campos**: Consiste en sustituir una información por otra equivalente pero de menor tamaño. Existen varios sistemas:
    - Codificación numérica: en lugar de almacenar dígitos decimales en bytes ASCII (base 256), se puede decidir almacenar dígitos binarios en signo/magnitud.
    - Tipos enumerados: equivalen a una codificación numérica natural (se puede establecer una codificación de ese tipo) para representar valores repetitivos mediante un solo número.
    - PseudoEnumerados: incluyen el valor 'otros' cuya ocurrencia implica la aparición explícita del valor a continuación en el registro.
    - Agrupación de varios campos: por ejemplo booleanos en un mapa.

  - <u>Observaciones de la optimización:</u> Suponen un cambio de convenio en el almacenamiento, por lo que si se hacen, deben ser completamente transparentes al usuario. Los que necesitan conocer y compartir el convenio son la aplicación que escribe los registros y la que los lee. En cambio la aplicación que los muestra lo hará del modo que favorezca más al usuario.
    -El número de bytes necesarios para codificar ***n*** símbolos es → $\boldsymbol{log_{256}n.}$
    -La codificación disminuye el tamaño pero no significativamente la densidad.

- **<u>Taxonomías:</u>** Existen varias taxonomías atendiendo a distintos criterios:

  - Atendiendo al **tamaño** de sus registros:
    - Longitud fija: Los registros fijos y de formato definido.
    - Longitud variable: Registros variables y de formato definido.
    - Longitud indefinida: Registros variables y sin formato definido.
  - Atendiendo a los **tipos** de registros:
    - Longitud fija: siempre se utiliza todo su tamaño.
    - Longitud variable: sus ocurrencias son de diversa longitud.
    - Longitud indefinida: Se desconoce la longitud de sus ocurrencias.

- Atendiendo al **formato** de sus registros:
  - **Homogéneos**: todos los registros siguen el mismo patrón.
  - **Heterogéneos**: Contienen varios tipos de registro.
    -Suelen ser variables o indefinidos.
    -La heterogeneidad puede afectar a todo el registro o sólo a una parte.
    -Distinguimos dos casos de heterogeneidad:
      1- **Única** (que solo aparece uno de los tipos o ninguno) → En éste caso utilizaremos un campo tipo para saber cuál es en cada caso.
      2- **Múltiple** (pueden aparecer varios tipos) → necesitaremos un campo 'mapa' que es un binario de tamaño igual al nº de tipos distintos considerados (un bit por cada tipo, que adopta el valor 1 si ese tipo ocurre y 0 si no ocurre).
      *Ver ejemplo en diapositiva 39.

- Atendiendo a su **unidad básica** de información dentro del fichero:
  - Binarios: unidad básica, el bit.
  - Textuales: unidad básica, el carácter.
  - Tipados: Unidad básica, el registro.

- Atendiendo a la **función** del fichero:
  - Permanentes: orientados al almacenamiento.
  - Temporales: orientados al procesamiento.

- Atendiendo a la función del **contenido** del fichero:
  - Maestros (de situación): Información inversa (variada).
  - Constantes (de referencia): Pocos valores y poco volátiles.

- Atendiendo a la **oportunidad** del contenido del fichero:
  - Borrador: aquellos que aún no han empezado a usarse.
  - Vigente: Los que están en uso.
  - Histórico: De uso pasado.

☐ **Operaciones sobre ficheros:**

<table>
<tr><th>Tipo</th><th colspan="2">operación</th><th>Rango de acción</th></tr>
<tr><td rowspan="2">De gestión (de ficheros)</td><td>Creación</td><td>destrucción</td><td rowspan="3">Selectivas</td></tr>
<tr><td>Apertura</td><td>cierre</td></tr>
<tr><td rowspan="2">Recuperación de información</td><td colspan="2">Consulta a la totalidad</td></tr>
<tr><td colspan="2">Consulta selectiva</td><td rowspan="4">A la totalidad</td></tr>
<tr><td rowspan="3">Actualización de información</td><td colspan="2">Inserción</td></tr>
<tr><td colspan="2">Modificación</td></tr>
<tr><td colspan="2">Borrado</td></tr>
</table>

- **Gestión de ficheros:** El sistema gestor de ficheros (SGF) del sistema operativo es el que se encarga de identificar localizar y atribuir los ficheros a cada area individual de almacenamiento. Soporta operaciones de gestión y acceso a nivel de bloque.
- **Memoria intermedia:** Permite anticipar las próximas lecturas para que estén disponibles cuando sean necesarias recogiendo todo el bloque leído y realizando lecturas en tiempo de procesamiento. También permite demorar escrituras, ahorrando accesos si caben varios registros en el bloque de escritura.

- **Claves:** Son pampos que facilitan la interacción de los usuarios con el fichero. Posibilitan la identificación y la localización de registros. Existen varios tipos de clave:

  - **Clave de identificación:** identifica unívocamente un registro en el fichero. Toma valores diferentes para cada registro.
  - **Clave de Direccionamiento:** Localiza la posición de un registro en el fichero, aunque no tiene por qué ser completamente identificativa. Dado que puede referenciar a un espacio en el que encontraremos el registro buscado, pero podría ser común a varios otros guardados en la misma zona (dentro de la cual puede existir otra clave de direccionamiento, o se puede realizar una búsqueda serial en ése espacio de memoria si es reducido).
  - **Clave de ordenación:** Son criterios de ordenación física (referente al fichero) o lógica (referente al proceso). No necesariamente identificativa.
  - **Clave de búsqueda:** Información utilizada por los usuarios para localizar registros.

  Todas las claves deben ser mínimas. No debe existir ningún subconjunto de la clave que cumpla las mismas funciones que la totalidad de dicha clave (en tal caso sería reducible). A continuación se muestran una serie de claves incorrectas o inadecuadas:

  - Clave de identificación: Año, Productora
    (probablemente tenga dos películas que coincidan en productora y año)
  - Clave de direccionamiento: Formato
    (pocos valores posibles, poca capacidad de direccionamiento)
  - Clave de Ordenación: Título, Año
    (si está ordenado por título, que es único, no tiene sentido el año)
  - Clave de Búsqueda: Título, Director
    (no es mínima, porque el título por sí solo identifica un solo registro)

TEMA 6: ORGANIZACIÓN DE FICHEROS - ORGANIZACIONES BASE:

☐ **Factores de decisión**: La elección de una determinada organización para un fichero es una decisión de diseño, que depende de ciertos parámetros, que producen una serie de condiciones. Separadas en cuatro tipos podemos diferenciar:

- **Naturaleza de los procesos:**
  - Según la naturaleza de la operación, si es de actualización o de recuperación.
  - Según el tipo de acceso (en atención al orden) pueden ser ordenados o desordenados.
  - Según la tasa de actividad del proceso:
    - elevada (>10%) si contamos con varios procesos que realizan accesos seriales/secuenciales, utilizaremos organizaciones consecutivas. Por ejemplo en la lectura de cadenas de datos.
    - Reducidas (<10%), en procesos con acceso mixto, utilizaremos organizaciones invertidas.
    - Si emplean selección unívoca (de un elemento), utilizaremos organizaciones direccionadas.
    - Si contamos con varios procesos diferentes puede ser útil emplear organizaciones indizadas.

- **Factores a optimizar del fichero**:
  - Tiempo de respuesta de actualizaciones o recuperaciones, debemos disminuir los accesos, emplearemos organizaciones en árbol, memorias intermedias.
  - Espacio de almacenamiento: Podemos optimizarlo incrementando la densidad, minimizando el almacenamiento auxiliar, utilizando compresión de campos, buscando tamaños de cubo adecuados, etc. <<Optimizar el espacio de almacenamiento también mejora el tiempo de respuesta>>.
  - Tiempo de procesamiento: Debemos evitar reorganizaciones en tiempo de procesamiento.
  - Coste de desarrollo y mantenimiento: Reduciendo el coste de programación podemos reducir el coste de actualizaciones y mantenimiento, minimizar las reorganizaciones etc. Esto se consigue con un diseño cuidadoso del sistema.

- **Parámetros del fichero:**
  - El volumen y ocupación de cada registro y del total del fichero (aumenta la densidad).
  - La volatilidad (en función de las tasas de inserción, modificación, supresión y crecimiento).

- **Características del soporte:**
  - En función del tipo de soporte (si presenta una organización serial, direccionada, etc).
  - El tamaño y el tiempo de acceso a cada bloque de memoria.
  - El espacio de direccionamiento (tamaño de la dirección).
  - Memorias intermedias (caché).

*Imp: No confundir 'tipo de soporte' con 'organización de fichero'.

☐ **Organizaciones consecutivas:**

- **Organización SERIAL:** Es el tipo de organización básica, surge de los soportes seriales que proporcionan los registros físicos uno detrás de otro, y se acceden en ese orden. → Ejemplo: la cinta magnética.
  - Instrucciones soportadas: Leer bloque y reset.
  - Comportamiento ante procesos:
    - Procesos selectivos, buscan uno o varios registros teniendo en cuenta que:
      -Antes de buscar, siempre se apunta al principio del fichero (reset).
      -Se van leyendo todos los registros hasta identificar el buscado.
      -Si se quiere localizar varios bloques, se lee todo el fichero.
    - Procesos a la totalidad: que no precisan de localización y orden, sólo una operación detrás de otra, por lo que son óptimos.

- Interacción con ficheros seriales:
  - Recuperación: se realizan por medio de consultas que recuperan el contenido.
  - Actualización: 3 tipos
    1. Inserción (añadir registros al final del fichero)
    2. Borrado: -Físico: vaciando el registro → desplazando el resto.
       -Lógico: Marcándolo como vacio → se genera un hueco.
    3. Modificación: -De registros fijos: alterando el contenido.
       -De registros variables: Borra el antiguo e inserta el modificado.
- Cálculo del tiempo de acceso:

Tiempo máximo	Es el tiempo necesario para leer todos los registros. (tiempo del peor caso)	$t_{max} = n^o\ registros \cdot t_{acceso\_registro}$
Tiempo medio	Se calcula como el acceso a la mediana (el de en medio):	$t_{medio} = \frac{n^o registros + 1}{2} \cdot t_{acceso\_registro}$

*Como la organización serial puede ser consecutiva (en bloques) o no consecutiva (en cubos), estas fórmulas podrán leerse como Nº de bloques, o Nº de cubos y Tacceso_bloque etc…

- Mantenimiento Serial: Gestión de huecos en el borrado:
  - El borrado Físico con registros fijos: Se toma el último registro y se escribe sobre el que se desea eliminar, luego se vacía el último. Esto implica 3 accesos más la selección.
  - El borrado físico con registros variables: Cuando existe diversidad de tamaños se desplazan todos los registros posteriores al eliminado → altamente ineficiente.
  - El borrado lógico (en general): Se introduce una marca de borrado lógico (implica sólo 1 acceso para la escritura). Al insertar un nuevo registro se recorrerá el fichero buscando un hueco marcado como vacío de tamaño adecuado. Si no lo hubiera se insertaría al final.
  - Compactación: Se van desplazando registros para eliminar huecos.

- **Organización Secuencial:** Surge a partir de la serial, introduciendo un orden de registros mediante una clave de ordenación física que permite avanzar y retroceder entre registros).
  - Instrucciones**:** Esta organización permite acceder a los bloques aleatoriamente, por ello precisa un mecanismo para poder localizar el comienzo del primer registro de cada bloque.

  - Comportamiento ante procesos**:**
    - Selectivos: pueden aprovechar el que vayan ordenados, realizando búsquedas dicotómicas en la localización de un registro (o dicotómicas + serial cuando son varios registros), o si la búsqueda es sobre una clave alternativa (no principal) sería sólo mediante búsqueda serial.
    - Procesos a la totalidad: serial. Gran eficiencia en procesos ordenados.

  - Interacción con ficheros secuenciales CONSECUTIVOS: El primer carácter de un bloque puede no corresponder al primer reg. por lo que se hace imprescindible usar marcas de inicio (o fin) de reg.
    - La recuperación (consulta) puede recuperar el contenido de un registro, o de varios (de forma ordenada o desordenada).
    - Las actualizaciones:
      1. De inserción: añaden registros al final del fichero → altera el orden.
      2. El borrado: Es igual que en serial pero es más difícil reutilizar los huecos.
      3. Modificación: -De registros fijos: Se altera el contenido si no altera el orden.
         -En reg. Variables: Se borra el antiguo y se reinserta modificado, por lo que se altera el orden.

*Observar que se genera un área desordenada al final del fichero.

- Interacción con ficheros secuenciales **NO** CONSECUTIVOS: El primer registro del bloque comienza en el primer carácter del bloque, por lo que se puede prescindir de la marca separadora de registros. En cada bloque (cubo) encontramos espacio libre en el que se van insertando los elementos si caben, de lo contrario se insertan en un área desordenada al final del fichero.
  - Las operaciones de recuperación se basan en el mismo mecanismo que en las consecutivas
  - Las operaciones de actualización consisten en q todos los bloques (cubos) tienen un hueco.
    1. Inserción: Se inserta en su bloque si cabe (si no en el área desordenada)
    2. Borrado: Se agrande el hueco libre del cubo donde se borra, pero no afecta al resto.
    3. Modificación: Se emplea borrado y reinserción, dado que la modificación podría implicar guardarlo en otro cubo.
  - Espacio libre Distribuido: Consiste en dejar un porcentaje de espacio libre en los cubos para futuras inserciones o modificaciones.
  - Desbordamiento: Técnicas de resolución de desbordamiento >importante<
    1. Rotaciones: traspasar elementos de un cubo lleno a su vecino (si tiene espacio)
    2. Intercalar cubos completamente vacíos al crear o reorganizar el fichero.
    3. Partición celular: cuando desborda, se intercala un cubo vacío y se reparten los regs.

- Localización de ficheros secuenciales
  - Búsqueda dicotómica: Consiste en mirar el bloque (o cubo) de la posición central y, si coincide <fin> si no coincide, se escoge la mitad relevante del espacio de búsqueda (dependiendo de si el dato es mayor o menor) y se realiza una nueva iteración.
  - Búsqueda dicotómica extendida (para claves no univocas): Se busca el primer elemento por búsqueda dicotómica normal sobre una clave y luego se realiza un recorrido serial hacia arriba hasta encontrar uno distinto (fallo), entonces se comienza un recorrido serial hacia abajo hasta encontrar un nuevo fallo.

- Tiempos de acceso:
  - Búsqueda dicotómica:

<table>
<tr><td>Con búsqueda dicotómica</td><td>Considerando sólo el peor caso en el que se encuentre el registro en la última vuelta</td><td>Nº de accesos $_{max} = \log_2(n+1)$
$Tiempo_{max}=nºaccesos_{max}\cdot Tacceso\_elemento$</td></tr>
<tr><td>Secuencial ordenado</td><td>Búsqueda dicotómica + serial sobre el área desordenada (añadimos la media ponderada)</td><td>Nº de accesos $_{max} = \log_2(n+1)+nºacc\_serial$</td></tr>
<tr><td colspan="3">*En ficheros no consecutivos (con cubos) el tiempo de acceso a elemento es Tac_el=Tac_bloq · Ec</td></tr>
<tr><td rowspan="2">Con búsqueda dicotómica extendida</td><td rowspan="2">Para hallar el nº de accesos consideramos el máximo (peor caso). Se emplea una clave que presenta k coincidencias medias y que tiene v valores distintos.
k·v=total registros</td><td>No consecutivas:
Nºacc_max= $\log_2(n+1) + [(k+1)/Tc]$</td></tr>
<tr><td>Consecutivas:
Nºacc_max= $\log_2(n+1) + [T_{reg}\cdot(k+1)-1/Tb]$</td></tr>
<tr><td colspan="3">-El nº de elementos en la búsqueda (n) depende de la relación físico-lógica.
-Si los k registros caben en un bloque se utilizará n=número de bloques.
-En el resto de casos, se debe utilizar el número de valores distintos (v).
-Si existe, se debe seguir considerando el área desordenada (se recorre entera siempre).</td></tr>
</table>

- Mantenimiento:
  - Reutilización de huecos: Podemos emplear una lista ordenada de huecos (en función de la clave anterior, la posterior y el tamaño) que se actualice cuando realizamos un borrado y, al insertar, se comprueba si existe un hueco en el que realizar la inserción. Si no lo hay se incluye al final.

- Espacio libre distribuido (solo en org. Secuencial no consecutiva). Cada cubo reserva un porcentaje de su espacio para modificaciones y para inserciones. Al insertar se recupera su cubo por búsqueda dicotómica, si el registro a insertar cabe en ese cubo (respetando el espacio libre para modificaciones) se inserta allí. De lo contrario se aplica una gestión de desbordamientos (rotación, partición celular, o área de desbordamiento desordenada).

- Reorganización: Reescribir todos los registros ordenados. Presenta el problema de que los algoritmos sobre el propio fichero pueden ser muy pesados (coste elevado). Habitualmente se usan algoritmos basados en almacenamiento auxiliar.

☐ **Organizaciones Direccionadas**: Surgen gracias a los soportes direccionados, que proporcionan registros físicos localizables mediante una dirección física en el soporte.

- Instrucciones soportadas: leer (n), escribir (n).

- Comportamiento ante procesos:
  - Selectivos: Ésta organización permite la localización inmediata con clave de direccionamiento unívoca (que debe mantener una correspondencia con una dir.física).
  - A la totalidad: En éste tipo de procesos la localización no es una ventaja.

- **Hashing**: La clave de direccionamiento no suele coincidir con la dirección física, sino que requiere de una transformación. La clave de direccionamiento se corresponde con un espacio de direccionamiento y éste a su vez se corresponde con una dirección física.

Elemento:	clave	←→	dir. cubo	←→	dir. bloque
Ámbito:	lógico	←→	físico-lógico	←→	físico

  - Espacio de direccionamiento (N): Es el rango de direcciones relativas disponible del fichero.
  - Clave de direccionamiento: Es el dato que localiza sobre el dominio de la clave.
  - Dirección relativa: Ordinal sobre el espacio de direccionamiento (N)
  - Dirección base: Dirección física del bloque en el disco.
  - Algoritmos de transformación:
    T1 → Proporciona la dirección relativa a partir de la clave de direccionamiento.
    T2 → Proporciona la dirección base a partir de la dirección relativa.

o **Organización direccionada directa**: Cada registro tiene su dirección reservada, de tal forma que existe una dirección única para cada clave de direccionamiento. Esto permite que la clave de direccionamiento también sea la clave de identificación con lo que se pueden realizar localizaciones inmediatas en 1 acceso. Existen dos tipos:

- Absoluta: La clave de direccionamiento es la propia dirección física, por lo que no es necesaria la transformación pero ofrece una densidad bajísima.

- Relativa: La clave de direccionamiento necesita transformación (empleando una función de transformación biyectiva) pero ofrece una mejor densidad.

La principal problemática de este tipo de direccionamiento es encontrar una clave de direccionamiento y una función de transformación adecuadas (que ofrezcan alta densidad y sea biyectiva).
Por otro lado, como cada clave de direccionamiento tiene una posición de almacenamiento reservada, si no se encuentran todas las claves e direccionamiento en el fichero, quedará mucho espacio desperdiciado.
La solución a éste problema pasa por dispersar las claves de direccionamiento.

- **Organización direccionada dispersa**: (a cubo) En esta organización, varias claves de direccionamiento coinciden con una misma dirección física (claves sinónimas). Esto reduce mucho el espacio de direccionamiento a N/m con m>=1, y reduce también el número de posiciones vacías, con lo que aumenta la densidad.
  - Optimizaciones posibles:
    - Si se producen cúmulos o huecos libres (por distribuciones normales de la naturaleza), debemos emplear una función de transformación que reparta mejor los registros.
    - Si continúan produciéndose colisiones, deberemos incluir en cada dirección física varios registros lógicos, es decir, debemos emplear cubos de tamaño "n" en los que quepan varios registros, de tal manera que, aunque exista colisión, nos caben varios registros en cada dirección.
    - Si hay más colisiones que espacio disponible (coinciden en un cubo más registros de los que caben) → **Desbordamiento** , tres posibles soluciones:
      1. Aumentar el tamaño del cubo (pero esto reduce la densidad)
      2. Aumentar el espacio de direccionamiento N' y redistribuir los datos (si es posible).
      3. Aplicar políticas de gestión de desbordamientos como buscar otra ubicación para los registros excedentarios en un **área de desbordamiento** situada al final del fichero y cuyo acceso es serial o secuencial.

  - Algoritmos de transformación: tres pasos:
    1. Si la clave de direccionamiento es alfanumérica, transformarla a numérica
    2. Aplicar una función de transformación para dispersar más los datos (la función debe estar definida sobre el espacio de direccionamiento (N) y si no es así se aplicará otra función para ajustarla a ese espacio).
    3. Transformar la dirección relativa obtenida en dirección base.

    A tener en cuenta:
    →La mejor función de transformación es aquella que proporcione una distribución uniforme sobre el espacio de direccionamiento (dispersión lineal). No existe una solución universal, la elección de una función de transformación u otra depende del problema en particular.
    →La función escogida puede ser el resultado de la combinación de otras funciones comunes, para conseguir el resultado deseado.
    →Si la clave de direccionamiento produce pocos valores por más que se transforme, no aumentará su capacidad de direccionamiento (se tratará de una clave pobre).

  - Funciones de transformación:
    - Truncamiento: consiste en dehechar parte de la clave (p.ej: los X primeros dígitos).
    - División-Resto o Resíduo: Consiste en dividir la clave entre un número natural y tomar el resto. Resulta muy útil para adaptar cualquier resultado de otra función al espacio de direccionamiento empleado: CD mod N
    - Plegado: consiste en dividir la clave en varios grupos numéricos y combinarlos (por ejemplo, partirla por la mitad y sumar ambas mitades).
    - Cambio de base: Tomando la clave de direccionamiento en decimal se realiza un cambio a otra base cualquiera. (por ejemplo 527 mod 11 = 10)
    - Método de LIN: (referido a los primos **p** y $\mathbf{q^n}$) se tomará una clave de direccionamiento en base decimal como si estuviera expresada en base p (normalmente superior a 10). Se expresa en decimal y se divide entre q^n. (ejemplo: P=11, q=7, n=2, CD=95 →
      $9 \cdot 11^1 + 5 \cdot 11^0 = 104$ → 104 DIV $7^2 = 2$

☐ **Tratamiento de desbordamientos:** repasemos primero dos conceptos:

-Colisión: Dos claves de direccionamiento tienen una misma dirección física.

-Desbordamiento: Un elemento que ha producido una colisión no cabe en la dirección física proporcionada.

Empleamos diferentes políticas para redireccionar esos elementos de forma poco costosa para los procesos, que podemos clasificar por dos criterios:

Según la zona donde se ubique el registro desbordado	-Saturación: Se emplea otra dirección dentro del espacio de direccionamiento. -Área de desbordamiento: Se asigna otra dirección fuera del área de datos
Según el mecanismo de ubicación	-Direccionamiento abierto. -Encabezamiento. -Otros (organización independiente).

m \ z	Saturación	A.Desbmto.
Dir. Abierto	✓	×
Encadmto.	✓	✓
Otros	×	✓

*A continuación se analiza cada tipo:

- **Saturación con Direccionamiento abierto:** La nueva dirección se calcula a partir de la dirección antigua (desbordada) si esta estuviese ocupada se produce un **CHOQUE,** si además no cupiera en ése cubo se producirá un **REBOTE.** En éste caso se buscará una **nueva dirección** hasta encontrar una posición libre o hasta hacer recorrido todo el espacio (área de datos saturada).

  Si el área de datos saturada se precisa otra gestión de desbordamientos, se suele aplicar una extensión automática del espacio de direccionamiento a N' mayor que N.

  - Byte de desbordamiento: Cualquier búsqueda terminará al encontrar el elemento (si la clave de direccionamiento es identificativa) o sondear todo el área. Paramejorar esto se introduce un byte en el cubo que informa de que está desbordado o no.
  - Técnicas de direccionamiento abierto:
    - Saturación progresiva (sondeo lineal): Al desbordar, la nueva dirección es la siguiente dirección base: D'=(D+1) MOD N
    - Rehashing (sondeo aleatorio): Al desbordar se suma una constante: D' = (D+k) MOD N
      -N y k deben ser primos relativos para que se sondee todo el espacio de direccionamiento N
      -Observar que la saturación progresiva es un caso de rehashing con k=1.
    - Sondeo cuadrático: Progresión D'=(D+k^2)mod N, tomando valores D+1, D+4… D+n^2
    - Doble transformación: Se aplica cualquier función de transformación, quedando D'=f(D) mod N. Observar que el rehashing es un caso particular de doble transformación.
  - Encadenamiento: Cuando un cubo se encuentra desbordado, en lugar de un byte de desbordamiento se escribe un puntero apuntando a la ubicación del desbordamiento (bien sea otro cubo del area de datos o un cubo de desbordamiento).

    -Se pueden encadenar cubos enteros o registros individuales (si se encadenan cubos, cada uno tendrá un puntero al siguiente cubo. En cambio, al encadenar registros el cubo desbordado que los contiene apunta al primero y luego cada registro individualmente apunta al siguiente).

    -Normalmente cada elemento encadenado tiene un puntero al siguiente elemento encadenado, pero también se pueden almacenar varios punteros a posiciones encadenadas (lista de encadenamiento con o sin orden).

- **Saturación progresiva encadenada:** Busca una nueva posición dentro del espacio de direccionamiento, y almacena allí el registro desbordado, guardando su posición en un puntero.

  -Este encadenamiento es siempre a registro (al primer elemento libre del cubo desplazado).

  -El apuntamiento debe ser identificativo.

  -Como es una dirección independiente de la dirección anterior, puede utilizarse cualquier otro cubo como destino. Se suele utilizar el primer cubo vacío.

- Área de desbordamiento: Consiste en almacenar los registros desbordados en un área especial, fuera del área de datos. De ésta manera se eliminan los choques y los rebotes pero es necesario emplear más espacio (auxiliar).
  *El uso de espacio auxiliar no es tan mal por el gasto de espacio, sino porque implica aumentar el espacio de búsqueda en procesos basados en claves de búsqueda alternativas (no privilegiadas).

- **Área de desbordamiento encadenada:** Es un encadenamiento fuera del área de datos.
  - Encadenamiento de registros: Es menos eficiente para procesos que realicen búsquedas por clave de direccionamiento:
    -*get all* → Con CD no identificativa tendrá que recorrer todo el encadenamiento.
    -*get one* → Con CD identificativa se suma la probabilidad de desbordamiento por el recorrido del encadenamiento hasta la mediana (k+1)/2.
    Es más eficiente para accesos por clave alternativa (reduce notablemente el tamaño del área de desbordamiento).

  - Encadenamiento de cubos: El puntero de encadenamiento apunta a un cubo, (solo requiere parte alta), de forma que cada cubo encadenado es apuntado por un solo puntero (se reserva esa posición).
    -*get all* → Con CD no identificativa se recorrerá serialmente todo el cubo de datos y sus cubos encadenados.
    -*get one* → Con CD identificativa es igual, pero reduciendo los cubos hasta la mediana.

  - Perjudica a los procesos por clave alternativa, dado que el area de desbordamiento se hace excesivamente grande.

- **Área de desbordamiento independiente:** El área de desbordamiento no se encadena, sino que se trata y maneja como un dichero independiente con su propia organización.
  Su organización es generalmente serial, pero también podría hacerse secuencial empleando un subconjunto de la clave de direccionamiento o por un direccionamiento secundario:
  -Se definen sobre un espacio de direccionamiento bastante menor que el otro, que será el direccionamiento principal.
  -La CD del área de desbordamiento suele ser la misma, o un subconjunto suyo.
  -Pueden desbordar, lo que produce una reorganización automática del direccionamiento principal, sobre un espacio de direccionamiento mayor.

TEMA 7: ORGANIZACIÓN DE FICHEROS - ORGANIZACIONES AUXILIARES:

Las organizaciones base estudiadas en el tema anterior suelen beneficiar sólo a un determinado tipo de proceso cada una (Serial para procesos de inserción, Secuencial para procesos de acceso ordenado y Direccionados para aquellos que emplean la clave privilegiada del sistema a la hora de localizar un dato). No obstante, dichas organizaciones no benefician en nada al proceso si realiza otra función que no sea la específica o si emplea otra clave de búsqueda que no sea la privilegiada, la localización se hace muy pesada (dado que pasa a ser una búsqueda serial)

Solución → almacenar la información de direccionamiento en un archivo a parte, en el que guardaremos únicamente la correspondencia de claves con la ubicación del registro. Este archivo se llama INDICE.

☐ **INDICES:**

- **Concepto de índice**: directorio cuya entrada se refiere a un solo registro (es una tabla de traducción de punteros), que se almacenan en ficheros independientes del fichero en el que se almacenan los datos.
  -El acceso selectivo a los registros a través del índice se realiza mediante la "clave de indización".
  -Es deseable que el índice esté almacenado en memoria principal (total o parcialmente) para optimizar el acceso.

- **Tipos de puntero (según su dominio):**
  - Punteros Físicos : Son direcciones máquina (la dirección física del registro), son rápidos de localizar pero dependen en gran medida del sistema operativo o de traducciones.
  - Punteros relativos: Dirección Relativa de un registro en el fichero (la dirección del cubo). Indican la posición del registro en el espacio de direccionamiento del fichero principal. Son aceptablemente rápidos de indizar pero también requieren cierta dependencia.
  - Punteros lógicos: Son punteros simbólicos referentes al archivo. Son muy lentos, pero altamente independientes a la hora de localizar un registro. Diferenciamos dos tipos:
    - P.L. identificador: Es la identificación lógica de un registro cuando es univoca (1 reg./cubo)
    - P.L. no identificador: Caracterización lógica de un conjunto de registros (varios reg. / cubo)

  A la hora de construir un índice, las claves se almacenarán en punteros lógicos, pero el tipo de puntero optimo para indicar la posición de los registros en el fichero es el puntero relativo (dirección del cubo). Pero esto implica que:

  - Si la organización base es secuencial, habrá que reorganizar el índice cada vez que se reorganice el área ordenada. Si se aplica partición celular puede interesar emplear puntero físico.
  - Si el direccionamiento es virtual extensible o dinámico, es preferible utilizar la dirección virtual, dado que ahorrará muchas actualizaciones del índice.
  - Para ganar independencia se puede hacer un índice intermedio con puntero relativo al índice intermedio o incluso a un puntero lógico.

- **Beneficios de indizar:**
  - Acceso por claves alternativas: Los procesos basados en localización por clave no privilegiada ganan notable eficiencia, al no tener que realizar búsquedas seriales sobre el fichero.
  - Aumento de la tasa de acierto (M_{interm}) la tasa de acierto es el porcentaje de accesos a memoria intermedia que no precisan acceso al soporte, con una gestión de memoria intermedia adecuada, esta tasa en los accesos al índice es muy elevada (hasta del 100%).7
    El coste efectivo se calcula como: C.global · $(1\text{-}T_{acierto})$. A mayor tasa de acierto menor coste efectivo y por tanto mayor eficiencia.
  - Reorganización menos costosa, los procesos basados en un orden aprovechas la ordenación física. Esa ordenación hay que mantenerla pero es menos costoso hacerlo en un índice (que ocupa pocos bloques) que sobre un fichero de datos completo.

- **Inconvenientes de indizar:**
  - Procesos de actualización más costosos: Los índices hay que crearlos y mantenerlos actualizados, lo que implica un coste adicional en inserciones, borrados y algunas modificaciones.
  - Necesidad de almacenamiento auxiliar.
  - Necesidad de operaciones de mantenimiento.

- **Operaciones sobre ficheros indizados:**
  - Operaciones sobre la totalidad del fichero:
    - Creación → crear el índice al crear el fichero o posteriormente.
    - Borrado → Si borramos todo el fichero de datos hay que eliminar el índice por completo.
    - Consulta → Generalmente, en procesos a la totalidad nos interesa realizarlos sin contar con el índice, es decir, realizándolos directamente sobre el fichero. (excepto si se trata de un fichero de datos desordenados con índice ordenado o si es un fichero de datos serial)
  - Operaciones selectivas:
    - Localización → se hará a través del índice.
    - Actualización → Si afectan al índice será necesario actualizarlo, teniendo cuidado de respetar el direccionamiento.

- **Coste de procesos en ficheros indizados:**

Tipo		Cálculo o extensión
Localización a través del índice		Acceso al índice (según su naturaleza)
Recuperación		C(Oi, Pj) = acceso_índice + acceso_datos (Ec)
Actualización	Inserciones	Pueden localizarse los nuevos registros empleando un índice no denso y luego insertar entradas nuevas en el índice (si es necesario)
	Borrados	Se localiza el reg. con el índice y luego se elimina o modifica la entrada.
	Modificaciones	Se localiza el reg. con el índice y luego puede ser necesaria la modificación de la clave o del puntero.

- **Taxonomía de índices:** *más adelante se especifican las características de los mas relevantes.
  - Según el carácter (identificativo) de la clave de indización :
    - Si es clave de identificación → índice primario
    - Si es cualquier clave de búsqueda no univoca → índice secundario.
  - Según la correspondencia (biyectiva o no) entre entradas y registros:
    - Denso (1:1) → existe una entrada del índice para cada registro.
    - No denso (1:N) → varios registros indizados por una sola entrada.
  - Según el recubrimiento del índice:
    - Exhaustivo→ Todos los registros que deben tener entrada la tienen
    - Parcial → No se indizan todos los registros (se dejan a parte los que se acceden rara vez, los últimos en ser introducidos, o bien solo se indiza el primero de cada tipo).
      *Si el índice exhaustivo falla, indica que el elemento buscado no existe, pero si un índice parcial falla no aporta ningún valor informativo.

☐ **Índice Secundario (Indización mediante claves secundarias):**

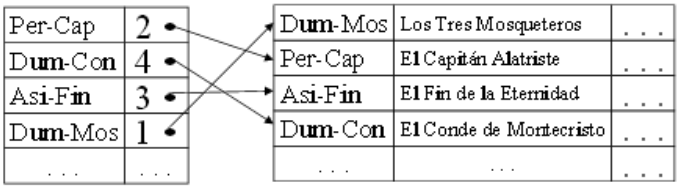
- La principal diferencia con respecto del índice primario es que la selección no es unívoca (pueden existir varias entradas de índice para cada valor de la clave).
  - La coincidencia (k) de una clave es el número de registros por valor, y será proporcional al número de registros (r) e inversamente proporcional a la cardinalidad del dominio de su clave.
  - El número de entradas de un índice secundario (exh) es la cardinalidad del dominio de su clave.
  - tipos de índice secundario: -Ind. Exhaustivo: Todos los registros indizados.
    -Ind .Parcial: Solo se indizan los valores más interesantes.

- Optimización: Habrá varias entradas con la misma clave (valores repetidos), por lo que resulta muy interesante almacenar sólo una vez el valor de la clave, y junto a ella todos sus punteros (lista de punteros).
  - Corolarios: Solo habrá una entrada por valor de la clave:
    - Mejora las búsquedas dado que solo hay que recorrer el índice hasta encontrar una entrada
    - Empeora las inserciones dado que hay que buscar la entrada correspondiente.
    - Empeora mucho en caso de que no exista una entrada, dado que tendrá que recorrer todo el fichero.

  *OJO→ con listas conviene tener el índice ordenado y siempre no consecutivo, y procurar que la longitud media de la lista sea una coincidencia de la clave k=r/card(Dom(IK)).

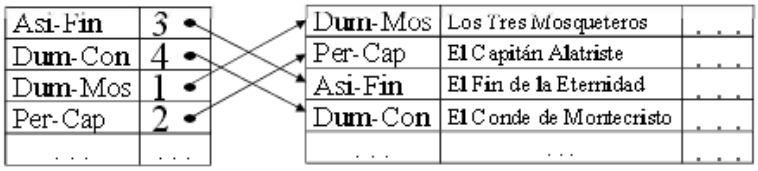
☐ **Índices primarios densos (tipos):**

- **Índices seriales:**



  - Uso: localización serial sobre el índice (mucho tiempo de procesamiento).
  - Coste: generalmente alto, ya que se recorre serialmente y suele ser grande. Del orden de (n+1)/2 accesos medios para recuperar una entrada; n accesos para un rango.
  - Restricciones: Se aplica ehxhaustivamente sobre claves no privilegiadas (si la organización base es secuencias $C_{ind} \neq C_{ord}$, y si es direccionada $C_{ind} \neq C_{dir}$ → excepción: Si se usa la proyección de un acceso invertido.

- **Índices Secuenciales:**

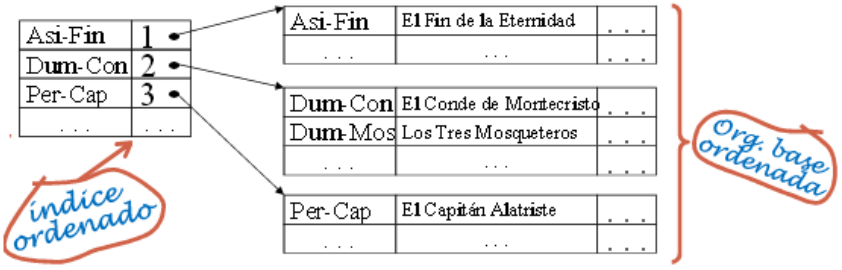


  - Uso: 1) localización ordenada sobre el índice (búsqueda dicotómica) .
    2) Proceso ordenado a la totalidad → Recorrido serial del índice.
  - Coste: Menor que en serial pero aparece la necesidad de la inserción ordenada (y reorganización local, o bien inserción desordenada y reorganización global).
  - Restricciones: -Sobre organizaciones base secuenciales, Si $C_{ind} = C_{ord}$ permite usar distintas técnicas para la localización (índice) y para listados (recorrido org. Base).
    -Sobre organizaciones base direccionadas, una Ci = Cd solo se justifica por un proceso ordenado a la totalidad por Ci (o por un acceso invertido sobre una Ci).

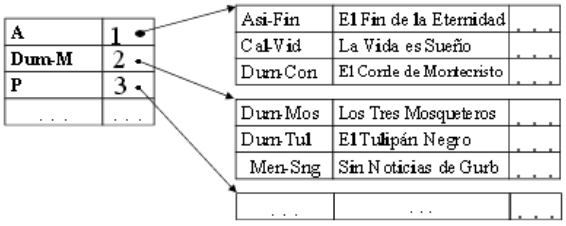
- **Índices direccionados:**
  - Uso: Índices muy eficientes para la localización de una entrada (un solo acceso para recuperar la entrada).
  - Restricciones: Deben tener buena dispersión, además es muy difícil que no degeneren.
  - Inconvenientes:
    - Dejan de ser eficientes si se producen muchos desbordamientos (no interesan si tienen más de uno o dos cubos de desbordamientos).
    - Son poco densos: No pueden mantenerse en memoria, solo mantendrían los desbordados.
    - Fuerte dependencia de la dispersión.
    - Funcionan mejor si se actualiza poco → índices poco volátiles.

  *Debido a estas restricciones, se suelen utilizar como índices temporales y para hacer clustering.

☐ **Índices primarios no densos:**



- Concepto: Emplean una entrada por cubo de datos (en lugar de una entrada por registro)
- Restricciones: Incide y organización base deben ser necesariamente secuenciales (ordenados), y con clave de indización == clave de ordenación.
- Uso: Aporta varias posibilidades de acceso:
  - Procesos ordenados a la totalidad → Acceso serial de la organización base (ordenada)
  - Procesos selectivos (solución única) → Acceso a través del índice
  - Procesos mixtos (selección de un rango) → Acceso Indizado (1er elemento) + serial
- Inconvenientes: -Por ser necesariamente ordenado, precisa gestionar desbordamientos
  -Solo puede existir un índice no denso para cada archivo.
- Ventajas: -Al ser de tamaño muy reducido, tiene menor coste (y mayor tasa de acierto).
  -En lugar de utilizar toda la clave en la entrada, se pueden usar prefijos.



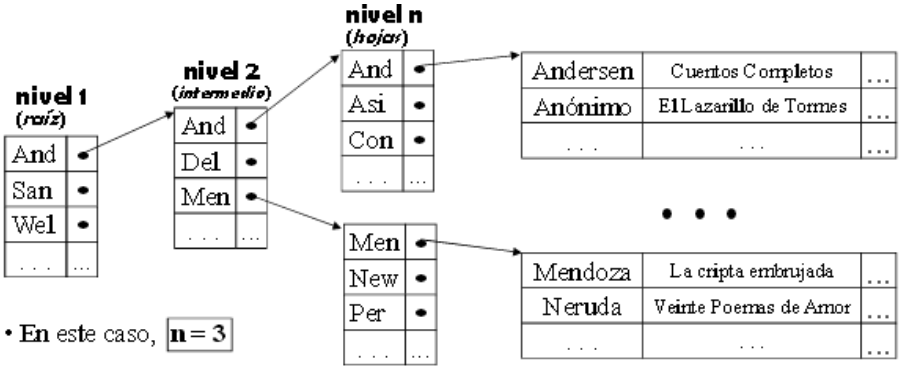
- Caso importante: Inserción de un elemento que desborda y reorganización … posibles soluciones:
  - Gestión mediante área de desbordamiento. Lo que implica tener otro índice denso para los registros desbordados.
  - Reorganización local como la **rotación** → puede producir cambios en el índice
  - Reorganización local "Al insertar un cubo desborda": El elemento desbordado se pasa al siguiente cubo y se cambia la clave que indiza dicho cubo por una que indique que se está produciendo ésta situación. Esto es un sistema peligroso que puede generar reorganizaciones en cadena y mucho desorden.
    →solución1: introducir espacio libre distribuido en cada cubo.
    →solución2: Intercalar cubos vacíos desde el principio automáticamente (partición celular)

☐ **Índices multinivel:**

Los índices primarios densos pueden resultar demasiado grandes, lo que implica un elevado coste de almacenamiento y de reorganización (si además es ordenado). Un índice no denso precisa menos entradas por eso es apropiado cuando hay muchas entradas y el índice no cabe en memoria. Aún así es posible tener índices tan grandes que no quepan en memoria y sea necesario **fraccionarlos.** La forma de reorganizar dichos fragmentos de índice (páginas), consiste en crear índices de dos o más niveles (índices que apuntan a índices).

Los índices multinivel son índices con N niveles en los que el nivel N es índice del nivel N+1.

-Como el nivel n es ordenado, puede ser un índice no denso (siempre que el archivo sea secuencial y Ci=Co)
-Igual que en los no densos, las claves en las entradas pueden ser prefijos.
-Se comportan mal en procesos de actualización (tienen mala y difícil reorganización), por eso se usan sobre archivos constantes (no volátiles) → El número de niveles es fijo hasta que se reorganicen.



- **Tiempo de acceso:** Con un índice multinivel es ventajoso que todo el nivel 1 (raíz) se bloquee en memoria intermedia, con lo que se ahorra un acceso al soporte. De tal manera que, si un cubo de cualquier nivel se lee en un acceso ($T_{cubo} = T_{bloque}$), el numero de accesos para conseguir la dirección de los datos será N-1. Y si el cubo de datos se lee en un acceso en total serán N accesos.

- **Problemas**:
  - El índice puede crecer y si no aumenta el cubo acabara desbordando.
  - El numero de accesos puede hacerse muy elevado
  - Mantener el índice ordenado es muy costoso (requiere mucha reorganización).

☐ **Indización en árbol:** Es un caso particular de índices multinivel. Existen varios tipos:

- **Árbol binario:** Cada nodo es una entrada del índice con dos punteros internos
  - Ventajas: Sencillez y uso eficiente de la memoria (intermedia o principal).
  - Inconvenientes: problemas de vecindad y desequilibrio.
  - Se almacenan para leerlos completos, usarlos y volcarlos completos.

- **Árbol AVL:** Resuelve el desequilibrio mediante procesos de reorganización local.
- **Árboles binarios paginados:** Resuelven el problema de vecindad.
- **Árboles AVL-paginados:** Tienen buen rendimiento pero necesitan muchos punteros internos, presentan una bajísima densidad y reorganizaciones frecuentes.

- **ÁRBOLES B:** Son árboles que contienen varias entradas en cada nodo (y por tanto varios descendientes), que se construyen de forma ascendente, de tal forma que el separador pertenece al nodo que desborda. Cada nodo contiene entradas de índice (pares clave indización-puntero a datos) y punteros a los nodos hijo.

  - Construcción: Inicialmente se comienza por las hojas del árbol, guardadondo las entradas de cada nodo de forma ordenada. Cuando un nodo desborda, se divide en dos y se promociona el elemento intermedio hacia el nivel superior (ese elemento se lleva **dos** punteros, uno hacia cada hijo, es decir, hacia cada uno de esos dos nuevos nodos).
  - Orden del árbol: indica la capacidad de los nodos (y por ende del árbol), puede obtenerse de dos formas: -Según las entradas del árbol: el nº mínimo de claves de un nodo.
    -Según los punteros (hijos): el nº máximo de hijos de un nodo.
  - Observaciones:
    - Si el árbol es de orden **m**, cualquier nodo tendrá como máximo **m** descendientes.
    - Si un nodo tiene m descendientes (no hoja), tendrá m-1 entradas
    - Corolario: Un nodo de un árbol de orden **m** tiene a lo sumo **k=m-1** entradas.
    - En un nodo (T_{nodo} bytes) caben **m** punteros internos y **k** entradas, luego:

$$\mathbf{m \cdot T_{ptro\_interno} + k \cdot Tentrada \leq Tnodo}$$

- El nodo raíz tiene al menos un elemento y, por tanto, al menos dos hijos.
- T_{nodo} es múltiplo de T_{bloque} y suele ser lo menor posible (típicamente 1bq).
- $T_{entrada}$ es la suma del tamaño real de la clave (fija / marcada / modificada) más el o los punteros internos (pueden ser muchos si es secundario).

- Propiedades:
  - Todos los nodos menos la raíz garantizan una ocupación mínima: $K_{min}=k/2$.
  - El nº de descendientes mínimo que tienen los nodos intermedios suponiendo política de 'dividir cuando desborda' es: $\mathbf{m}_{min} = \mathbf{k}_{min} + 1 \rightarrow \mathbf{m}_{min}=(\mathbf{m}+1)/2$
  - Tamaño máximo del fichero de índices se puede obtener mediante:

$$N^{o}_{máx}\ nodos\ fichero = nº\ \text{entradas fichero} / k_{mín}$$

$$T_{máx.}\ fichero = nº_{máx}\ \text{nodos fichero} \cdot T_{nodo}$$

  - El nº de niveles (n) para un árbol de orden m y e entradas tiene cota superior.

$$Cota\ Superior \leq n \leq 1 + log_{\lfloor \frac{m+1}{2} \rfloor} \left(\frac{e+1}{2}\right)$$

  - Para recuperar una entrada, el numero de accesos a soporte coincide (en general con el nº de niveles): -Dado que la raíz estará siempre en memoria, contamos un acceso al menos.
    -Hay que contar un acceso más, desde la entrada de índice al fichero de datos.
  - Para recuperar un registro aleatorio, el coste será el de recuperar una entrada, mas el acceso a tantos cubos como punteros tenga la entrada, es decir, la coincidencia c de la clave (número medio de registros por valor).

$$C(O_i,P_j) = (n-1)\cdot T_{nodo} + c \cdot E_c$$

  - T_{nodo} suele ser 1, y en un índice primario c=1, luego **$C(O_i, Pj)=(n-1)+E_c$**
  - En un índice secundario, **c= r / card(Dom(IK)),** r es el numero de registros y **card(dom(IK))** es la cardinalidad del dominio de la clave de indización.
  - En un árbol B todas las hojas están al mismo nivel → Es un árbol plano
  - La inserción de una única entrada puede dar lugar a una o más particiones y el borrado de una entrada puede dar lugar a una o más concatenaciones.
  - El coste de cualquier actualización sobre el índice en árbol b es el coste de localización más un acceso de escritura (del correspondiente nodo hoja).
  - El coste extra de una partición es de dos accesos de escritura (actualizar el nodo antecesor y escribir el nodo nuevo).
  - El tiempo de acceso calculado es una cota superior del tiempo de acceso en cualquier momento de la vida del fichero (mientras no varíe su contenido).
  - Puede calcularse una cota inferior (en base al numero de niveles del árbol pefectamente construido), para conocer su coste optimo y valorar el beneficio de ejecutar la reorganización del índice.

- Ventajas:
  - Existe una cota superior razonable del numero de accesos a soporte
  - Generalmente, las operaciones de inserción o borrado requieren reestructurar una página, y si son más, suelen ser pocas páginas.
  - En el peor caso, las páginas están ocupadas a la mitad aproximadamente.

- Desventajas:
  - Si las entradas son grandes, el orden puede ser demasiado pequeño
  - La densidad mínima de los nodos es muy mejorable
  - En las hojas se desperdicia mucho espacio (no necesitan punteros).

- **Árbol B*:** Son una mejora de los arboles B que pretende mejorar la densidad de los nodos. Para ello en lugar de dividir un nodo en dos, se dividirán dos nodos en tres, así en lugar de conseguir una ocupación mínima del 50% se obtendrá el 66%. Su funcionamiento se puede resumir en:
  - Cuando un nodo desborda, en lugar de dividirse, trata de pasarle uno de sus elementos a su vecino. A éste proceso lo llamaremos **ROTACION.**
  - Si el nodo vecino también está lleno, se dividen los dos nodos llenos en tres semi-llenos.
  - El resto del funcionamiento es igual que el de los arboles B.

- Propiedades:
  - Todos los nodos menos el raíz garantizan una ocupación mínima: $k_{mín} = \lfloor \frac{2k}{3} \rfloor$
  - Los nodos intermedios cuentan con (2K/3)+1 descendientes (suponiendo la política de dividir cuando desborda): $m_{mín} = \lfloor \frac{2m+1}{3} \rfloor$

- Cálculo de costes:
  - Como la localización es idéntica al árbol B, también es igual el cálculo de costes.
  - El coste extra de una rotación es de tres accesos (lectura del nodo contiguo, más la escritura de ése nodo y del nodo antecesor).
  - Cuando falla la rotación (un acceso de lectura) se tienen cuatro accesos extra (esa lectura, más la escritura de los nodos contiguo, nuevo y el antecesor).
  - También se puede contemplar la rotación bidireccional. En este caso la densidad es del 75% kmin=(3k/4), pero también el coste de inversión (rotación 4 y partición 5).

- Ventajas:
  - Aumento de la densidad al 66%
  - Un desbordamiento no siempre supone una partición (menor coste computacional).

- Desventajas:
  - Aumenta la probabilidad de desbordamiento (nodos más llenos).

- **Árbol B^+:** Implementación mejorada de los árboles B que pretende reducir el tamaño de las entradas de índice. Para ello se considera que los nodos intermedios son separadores y que todas las entradas de índice están en las hojas. En los nodos 'no hoja' solo es necesaria la clave (por los que se ahorra el espacio que ocupa el puntero a fichero de datos). Así en cada nodo hay más claves y más descendientes (mayor orden). Por otro lado, como las hojas no tienen hijos, no utilizan estos punteros. En su lugar tienen un puntero externo (que apunta al cubo de datos del registro) o una lista de punteros externos (si el índice es secundario). Su funcionamiento es similar al de los arboles B normales:

  - Solo cambia en una proporción de entradas de nodo hoja
    -El elemento que promociona desde una hoja debe permanecer en una hoja resultante de la partición.
    -En los nodos no hoja, se opera igual que en los arboles B.
  - En los nodos hoja, hay tantos punteros como claves. Sin embargo, se usa un puntero adicional para apuntar al siguiente nodo hoja. Este apuntamiento se realiza durante la partición, asignando:
    -Al encadenamiento del nodo derecho, el puntero existente.
    -Al encadenamiento del izquierdo, la dirección del nodo nuevo.
  - Se consigue así un encadenamiento de hojas que proporciona un mecanismo de acceso alternativo eficiente para ciertos procesos.

- Propiedades:
  - Orden del árbol (m): Se calcula para nodos no hoja, como en arboles B, teniendo en cuenta que las entradas de éstos nodos carecen de punt. ext: $\mathbf{m \cdot T_{puntero\_interno} + (m-1) \cdot T_{clave} \leq T_{nodo}}$
  - Ocupación máxima (k) de los nodos hoja: Si los tamaños de los punteros intr y extr son distintos, convendría calcularla por separado. $\mathbf{K \cdot (T_{clave} + T_{puntero(s)\_extr}) + T_{punt\_intr} \leq T_{nodo}}$
  - La ocupación minima de las hojas será (suponiendo política de dividir cuando desborda):

    $\mathbf{K_{min} = k+1 / 2}$
  - La ocupación minima de los nodos intermedios será **(k/2) → m_{min}=(m+1/2).**
  - Cálculo del nº de niveles (profundidad):
    1. el nivel de las hojas (n) indica el numero de hojas que hay → **nº hojas = [e/k_{min}]** siendo **e** el número total de entradas.
    2. El nº de nodos en el nivel n-1 depende del número de nodos del nivel n. →**nº nodos(n-1) = [nºnodos(n)/m_{min}]**
    3. Cuando se llega a un nivel con un solo nodo (raíz), éste será el nivel 1 (se tiene que el nivel **n-x=1**, y se puede despejar **n=profundidad del árbol**).
  - Tamaño máximo del fichero índice: Se calcula como la suma de los nodos necesarios para cada nivel multiplicado por el tamaño de un nodo: **∑ nodos(i) * Tnodo**

- Observaciones:
  - Los nodos intermedios son como un índice no denso (del nivel siguiente).
  - Las entradas de estos nodos solo necesitan parte de la clave (prefijos).
  - Recordar que los nodos no hoja del árbol B+ se operan como nodos B.
  - El orden es mayor que el de arboles B (especialemte en índ.secundarios), lo que hace que existan menos nodos separadores.
  - En consecuencia, la profundidad del árbol será menor (menor o igual en primarios, y mucho menor en secundarios) a pesar de que se repitan las entradas.
  - Todos los elementos tienen el mismo coste de acceso (aunque algunos elementos empeoren, la media del coste de acceso mejora).
  - Se pueden combinar las aproximaciones B* y B+ para crear árboles B+*.
  - Son muy utilizados en el ámbito profesional, sobre todo para crear índices secundarios.

- El encadenamiento de hojas proporciona mecanismos de acceso alternativo:
  - Éste tipo de arboles permite un uso tanto arbóreo como secuencial
  - Resulta especialmente útil para procesos ordenados cuyo orden lógico coincida con la clave de indización.
  - Los procesos que se pueden beneficiar del encadenamiento son: (ejemplos en base a un índice en árbol b+ con clave de indización fecha(dd-mm-aaaa):
    1. Procesos a la totalidad ordenados: Listar todos los registros ordenadamente
    2. Procesos selectivos con tasa de actividad elevada: listar registros con fecha “mayo”
    3. Procesos ordenados con varios resultados (rangos) → Acceso mixto: recuperar todos los registros entre el día (…) y el día (…). Los accesos mixtos consisten en recuperar la primera entrada a través del árbol y el resto de entradas se recuperan con el encadenamiento.

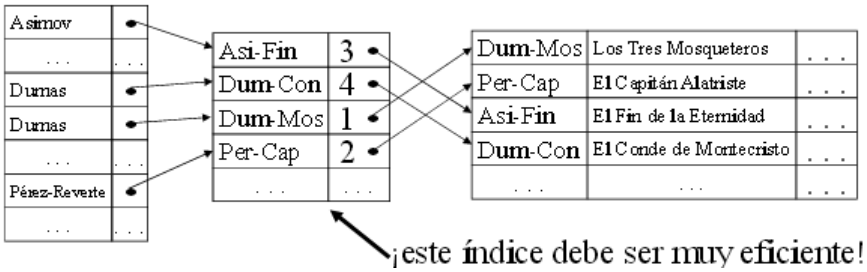
- **Familias de árboles B, consideraciones finales**: Organizar un fichero de datos en Árbol B o B* solo resulta interesante si el orden es alto (registros pequeños y paginas grandes). En general, el ‘registro’ es muy grande respecto a la clave, por lo que es preferible emplear arboles B+. aunque las mejoras logradas con arboles B+ y B* con combinables obteniendo sistemas con menor número de accesos, profundidad y coste de actualización, mejorando además el orden y la ocupación.

- **Ilustraciones de los diferentes árboles:**

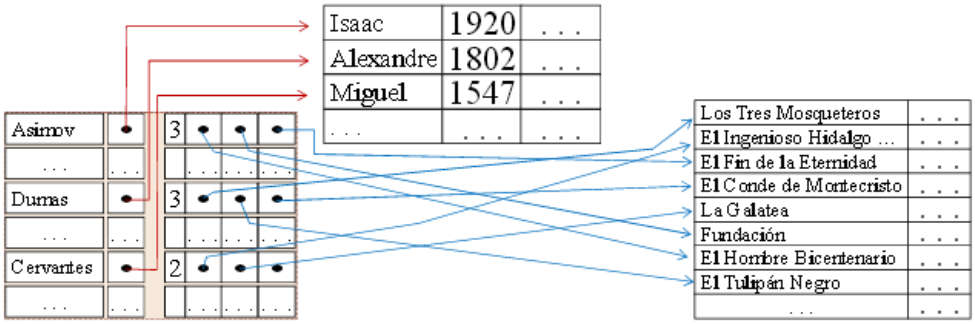
Árbol	
Árbol B	
Árbol B+	
Árbol B*	

☐ **Índices especiales:**

- **El índice intermedio:** Si se cuenta con más de un índice, se puede considerar un índice intermedio y referir el resto a este. Interesa que éste índice sea:
  - Primario (cada valor de clave identifica unívocamente un registro)
  - De tamaño reducido (para que quepa en memoria principal)
  - Casi constante (poco o nada volátil).



- **Los cluster:** Cuando una clave CI es común a varios archivos (clave ajena) puede crearse una estructura única de indexación. La entrada tendrá una clave de indización y uno o más esquemas de punteros (un puntero o una lista de punteros, según sea la naturaleza de la clave en el archivo indizado).



  - Para Oracle, un cluster es la definición de clave privilegiada.
  - A través del cluster, varias tablas pueden almacenar físicamente los datos combinados mediante ésa clave (eficiente para join y accesos por clave provilegiada, ineficiente para todo lo demás).
  - El cluster garantiza que toda la fina combinada (el resultado del join de todas las tablas implicadas para un valor del custer) se almacena físicamente en el mismo cubo (cuyo tamaño permite definir, aunque no se corresponda con el tamaño de la página).
  - Los diferentes tipos de registros que podemos encontrar se abordan con diferentes clusters:
    1. Simples para registros pequeños
    2. Indizados para registros grandes
    3. Dispersos para procesos selectivos
    4. Ordenados para procesos ordenados.

TM 8 – ACCESO MULTICLAVE

El acceso selectivo tiene dos vertientes: una **simple** que busca un registro (o varios) a través de una clave. Y otra **Multiclave**, en la que el proceso de selección afecta a varias claves, el criterio de búsqueda contempla varias claves aunque el resultado no implica el registro completo, sino una clave.

El acceso multiclave suele basarse en claves no identificativas, por lo que se puede aplicar en organizaciones clásicas, si bien algunas organizaciones específicas pueden reducir el coste. El acceso multiclave o n-dimensional es de gran interés por sus posibilidades y aplicaciones como las BBDD espacio-temporales.

☐ **Selección y direccionamiento multiclave:**

- **El acceso multiclave básico:** La búsqueda serial:
  - Consiste en recorrer todos los registros del fichero de modo consecutivo comprobando cuales cumpen la condición de búsqueda.
  - Ventajas:
    - Por compleja que sea la condición, el coste (acceso) no se ve afectado
    - No precisa de almacenamiento auxiliar
  - Inconvenientes:
    - Es una búsqueda serial (full-scan): recorre todo el área de desbordamiento
    - El coste es muy elevado (n) y se hace inmanejable en ficheros grandes.
  - Las mejoras basadas en ordenación no cubren todas las combinaciones, es decir, si hemos ordenado por "nombre, ap1, ap2" y no se aporta el nombre, no se aprovecha la ordenación.

- **Acceso multiclave con claves privilegiadas:**
  - El conjunto de direcciones relevantes (conjunto resultado) se puede ver reducido aplicando filtrado por claves privilegiadas (CD, CI…).
  - En cada paso del filtrado, el filtro solo se aplica si el coste de hacerlo es inferior al numero de direcciones candidatas.
  - Si la búsqueda es unívoca (exact match, por clave identificativa) y esta clave es privilegiada, el acceso se realiza por esta (ignorando el resto).
  - Si esta clave identificativa no fuera privilegiada, se pueden ejecutar filtros por otras claves, y luego las direcciones relevantes se recorren a la mediana. El coste en éste caso seria (n'+1)/2 siendo n' el numero de direcciones relevantes (después de filtrar).

- **Indización multiclave:** Sólo se aplica sobre índices secundarios (dado que si se involucrara un primario, el resultado de aplicar la función de indización sería una dirección directamente, y obtendríamos el registro en un solo acceso). Las consultas multiclave se realizan en tres pasos:

  - Se descompone la consulta en subconsultas de un solo índice cada una, de tal forma que cada subconsulta aportará un conjunto resultado (un parcial).
  - Se operan los conjuntos resultado mediante algebra de conjuntos para reconstruir la consulta original.
  - Se obtiene un conjunto de direcciones candidatas (donde pueden existir encajes).
    - Ejemplos: $a \wedge b \equiv A \cap B$ $\quad a \wedge \text{-}b \equiv A \text{ - } B$ $\quad a \vee b \equiv A \cup B$ $\quad \text{-}a \equiv \aleph \text{ - } A$
  - El coste de un acceso multiclave indizado se computa como la suma del acceso a los índices involucrados y el acceso a los cubos candidatos.

- **Indización multiclave en Árbol R:** La indización multi-clave también admite la creación de índices especiales que no estén basados en una clave, sino en varias simultáneamente. Éste es el caso del Árbol R, una evolución del árbol B para *d* dimensiones, en los que cada entrada no es un punto en una línea sino un intervalo d-dimensional.
  - Esto permite que la raíz tenga dos o más entradas (salvo si es un nodo hoja).
  - Permite una altura balanceada.
  - Los intervalos pueden solaparse (al buscar un intervalo hay que recorrer todos los descendientes que intersecan con el intervalo en cuestión).

☐ **El direccionamiento Disperso multiclave**: El acceso multiclace sobre organización direccionada consiste en ampliar el algoritmo de transformación para dispersar combinaciones no univocas de calores de varias claves sobre cubos. Es decir, se aplica un direccionamiento en el que la clave de direccionamiento n oes a+b+c, sino que se realiza mediante CD1=a, CD2=b , CD3=C.

- **Inserción:**

Propiedades:

- $N = \prod_{i=1,n} N_i$
- Transformación 1 en dos pasos: sub-transformación i + combinación

$$T1 \equiv f_1 + f_2 \cdot N_1 + f_3 \cdot N_1 \cdot N_2 + \ldots + f_n \cdot \prod_{i=1,(n-1)} N_i = \Sigma_{i=1,n} f_n \cdot \prod_{i=1,(n-1)} N_i$$

- **Búsqueda:** La localización de un registro se hará obteniendo un conjunto de direcciones relevantes (conjunto resultado). El algoritmo para codificar este direccionamiento para *d* dimensiones requerirá d bucles anidados. **El filtrado utiliza todas las claves o alguna de ella.**

☐ **Acceso invertido:** El acceso invertido es un tipo de acceso indizado multiclave orientado a optimizar el coste de acceso en procesos muy concretos, en los que se pretende averiguar información delimitada de ciertos archivos con condiciones muy concretas. (por ejemplo ¿Cuál es el valor del campo X en el archivo Y, para los registros en cuyo campo Z vale 'valor'?).

- El acceso invertido procurará averiguar toda esta información accediendo solo a los índices (sin acceder al archivo de datos), gracias a que sus punteros relativos deben localizar unívocamente cada registro, es decir, los punteros con parte alta (bloque) y parte baja (posición en el bloque).
- A la hora de realizar un acceso invertido, se ejecutarán primero las condiciones (para obtener el conjunto resultado), y después se busca en los 'indices objetivo' (incógnitas) pero.. ¡al revés! Es decir, a partir de cada dirección (puntero relativo) buscamos el valor (puntero lógico). Ejemplo:

- **Inversión por listas:** el acceso invertido es eficiente si requiere acceder a pocos índices (si accediera a varios no contenidos en M_{int}) costaría más que acceder a los datos.
  - Los índices involucrados pueden ser primarios o secundarios y puede utilizarse cualquier organización indizada (invertida, con punteros completos), pero también existen estructuras específicas como:
  - **Listas invertidas de un campo**: Consisten en una lista ordenada de todos los posibles valores de ese campo, a los que se asocian referencias a todos los registros que llevan ese valor. Estas listas tienen una serie de características particulares:
    - Son entradas de un índice (como un índice secundario agrupado por valores), o bien como un índice que en lugar de uno tiene varios punteros relativos.
    - Se trata de un índice ordenado (por la clave)
    - Cada puntero consta de partes alta (dirección de bloque) y baja (posición en el bloque).
    - Esas entradas de índice tienen tamaño variable (de clave y de lista de punteros).
    - Los valores se repiten en general: -Un valor en varios registros → clave secundaria.
      -Varios valores para un registro → campo multivaluado (grupo repetitivo)

- **Esquemas de bits (para un campo):** Es un vector de valores booleanos, dispuestos me manera que a cada posible valor se le hace corresponder una posición. Por ejemplo:

  Idioma (castellano, inglés, francés, alemán, italiano).

  El esquema de idioma para alguien que solo hable español es → (10000).

  - Si el numero de valores posibles de un campo fuera pequeño, se podría mantener un índice que en lugar de entradas [valor, puntero] tuviera [puntero, esquema de bits], además este sistema permite que se puedan considerar varios esquemas de campos concatenados.
  - Los esquemas de bits pueden hacer referencia a uno o varios campos (ficheros parcialmente invertidos) o incuso a todos (totalmente invertido), indizando algunos valores (i parcial) o todo el dominio (i exhaustivo).

Ejemplo:

apuntamiento	*departamento categ. idioma sexo*
18, 11	0001000000 0010 10100 10
10, 8	0001000100 0100 10000 10
12, 5	0001000000 0010 11100 01
. . .	. . .

- **Máscaras para condiciones de igualdad**:
  - Se realizan con un bit para cada posible valor, en el conjunto {0,1}
  - La selección comprueba la condición **S AND Q = Q.**

    EJ: empleadass del dep.informática que solo sepan castellano: Q= 0001000001..

- **Máscaras con bits que admitan cualquier valor:**
  - Se realizan cun un bit para cada posible valor en el conjunto {0,1,q}
  - La selección comprueba en lógica trivaluada la condición **S XOR – Q = 1.**

    Ej: qqq1 qqqqqq qqqq 0100 qq

o **<u>Coste del acceso invertido:</u>** El coste del acceso invertido es la suma de los costes de sus dos partes.

- Selección : obtención de los punteros
  - Listas invertidas no ordenadas: coste máx→ **n;** medio → **(n + 1)/2**
  - Listas invertidas ordenadas → **$\log_2(n+1)$**
  - Esquemas de bits → **n**
  - Otro tipo de índice → El coste correspondiente a esa estructura
- Proyección: obtención de vlaces correspondientes a los punteros
  - Esquemas de bits con puntero implícito → **min(n,r).**
  - Cualquier otro caso → **n**
- Simbología: **n** es el número de bloques del índice, **r** es el numero de resultados
- Si hubiera varios índices implicados en la condición o en la proyección, el coste en la cada parte seria la suma de los costes individuales de cada índice implicado.

o **<u>Fichero invertido:</u>** Se dice de un archivo (fichero) que está invertido si tiene índices invertidos para todos sus campos (totalmente invertido) o bien solamente para algunos de ellos (parcialmente invertido).

- Un dichero totalmente invertido es redundante se podría decidir no almacenar los campos invertidos (que cuentan con una lista invertida) y después ensamblar esa información cuando sea necesario.
- Un caso extremo de esto sería un fichero totalmente invertido , que podría minimizar el almacenamiento en el dichero maestro (hasta desaparecer).
- Este ahorro de espacio supone un coste muy elevado en tiempo si existe algún proceso de recuperación del registro completo.
- Admite optimizaciones dependiendo del propósito del sistema.

- **Sistemas duales:** El acceso serial tiene la ventaja de no precisar almacenamiento auxiliar, lo que implica una fácil inserción y además se puede aplicar a cualquier condición compleja. Por otro lado, el tiempo de localización es maño (peor cuanto más grande es el fichero).
    - Por todas estas razones, es muy habitual proponer sistemas duales, que implican una doble organización física de los datos (redundancia controlada).
        - Se cuenta con una organización base consecutiva, que simplifica las actualizaciones y que posibilita selecciones complejas.
        - Se incorpora una organización invertida (adecuada a las necesidades) para facilitar el acceso a través de determinadas claves (y valores).
        - Una de las organizaciones (la invertida) no se mantiene actualizada (dado que solo se actualiza cada X tiempo) lo que obliga a repetir muchas consultas.