Dpto. Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

# Elementos de Bases de Datos

Segundo Cuatrimestre 2015

## Clase 12:

## Nivel Físico (Parte II) – Procesamiento y Optimización

Mg. María Mercedes Vitturini
[mvitturi@uns.edu.ar]

# Leer (A) / [input ($B_a$)]

SI bloque $B_a$ con el dato A *no está* en el Buffer de Memoria ENTONCES

SI no hay espacio en el Buffer de Memoria para $B_a$ ENTONCES

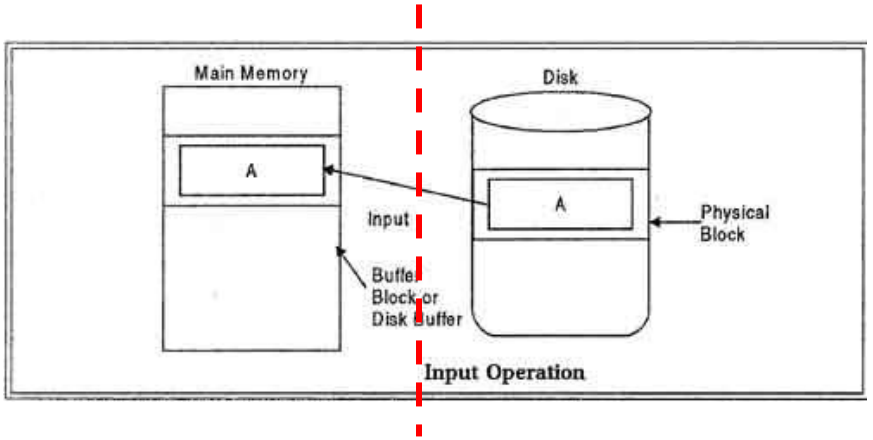
Liberar espacio para $B_a$

FIN-SI

Input ($B_a$)

FIN SI

Devolver A



# Escribir (A) /[input ($B_a$)] output ($B_a$)

SI bloque $B_a$ con el dato A no está en el Buffer de Memoria ENTONCES

SI no hay espacio en el Buffer de Memoria para $B_a$ ENTONCES

Liberar espacio para $B_a$

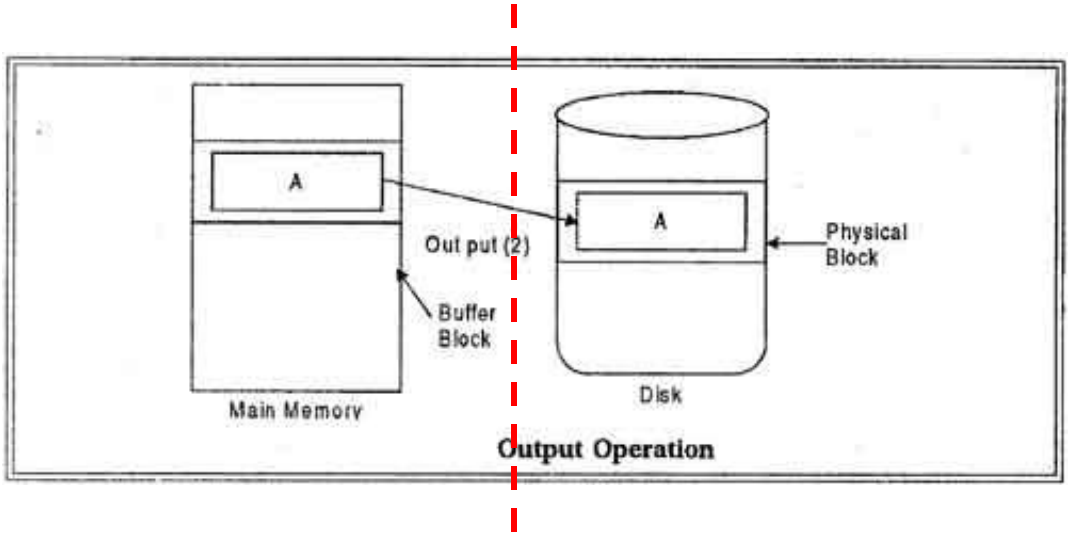
FIN-SI

Input ($B_a$)

FIN-SI

Escribir (A)

Output ($B_a$)



# Índices - Repaso

- Los índices aceleran el proceso de búsqueda de la información siguiendo una *clave de búsqueda*.
- **Clave de búsqueda:** no se requiere que sea una llave única

| *Clave de búsqueda* | Puntero |
|---|---|

## Tipos de Índices

- **Índices ordenados.** Basados en una disposición ordenada de los valores de clave de búsqueda.
- **Índices hash**. Basados en la distribución uniforme de los valores clave de búsqueda en *buckets*.

- **Índice primario (clustering):** cuando el archivo de datos está ordenado secuencialmente por la clave de búsqueda del índice.
- **Índice secundario (no clustering):** el orden del archivo de datos es distinto al orden de la clave de búsqueda del índice.

- **Índice Denso:** contiene todos los valores de clave de búsqueda.
- **Índice Ralo:** contiene algunos de los valores de la clave de búsqueda.

# Índices - Repaso

- Los índices aceleran el proceso de búsqueda de la información según una *clave de búsqueda*.
  - **Clave de búsqueda:** no requiere que sea una llave única.
- El comportamiento de los índices y su organización varía en algún sentido según la clave de búsqueda sea:
  - Por uno o varios atributos.
  - Sea un índice único (un único registro asociado a cada clave de búsqueda) o no.
  - Si se admiten valores nulos en alguno/s de los atributos que forman la clave de búsqueda.
  - Es una superllave.

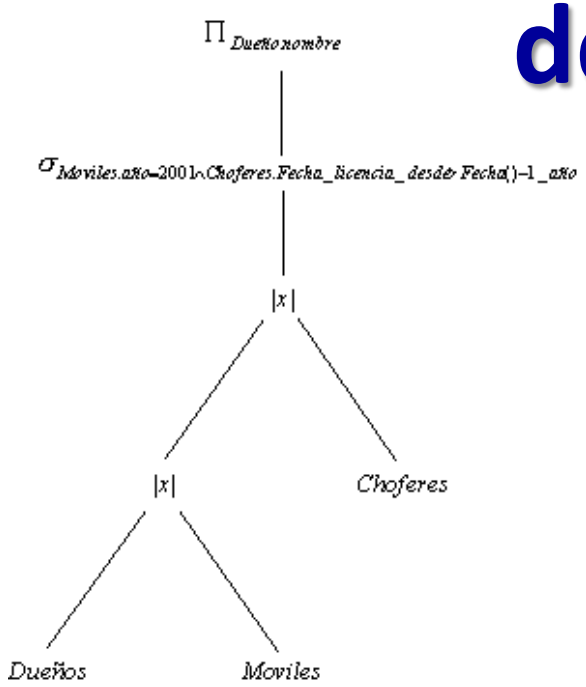
# Punteros

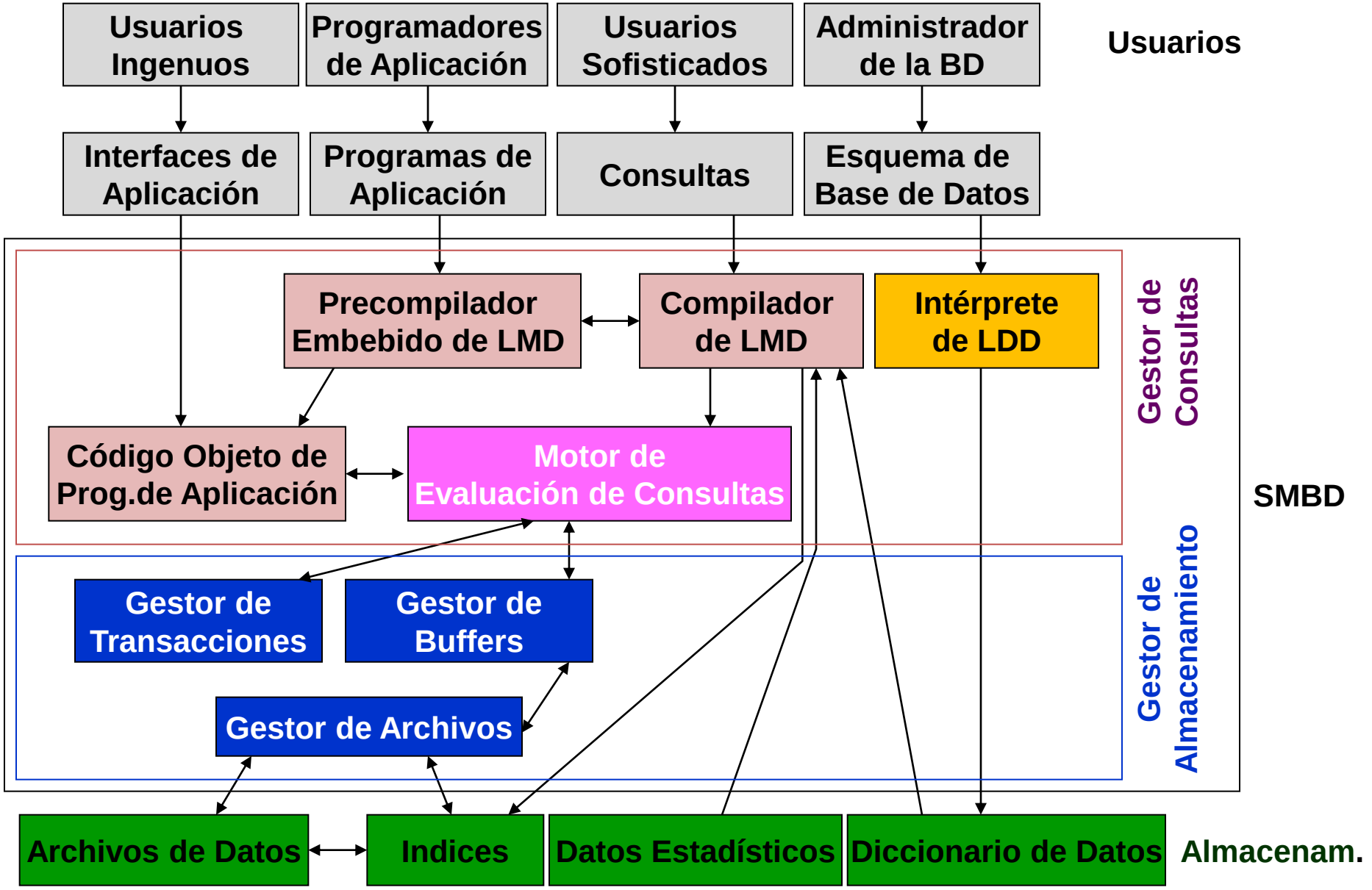
"An index record, or index entry, consists of a search–key value and **pointers** to one or more records with that value as their search-key value. The pointer to a record consist of the identifier of a disk block and an offset within the disk block to identify the record within the block – **Database System Concepts 6th Ed. A. Silberschatz"**

## Pointers

"All Files are organized by using two basic constructs to link one piece of data with another piece of data: sequential storage and *pointers*. With sequential storage, one field or record is stored right after another field or record. Although simple to implement and use, sometimes sequential storage is not the most efficient way to organize data. A pointer is a field of data that can be used to locate a related field or record of data. In most cases, a pointer contains the address, or locations of the associated data. Pointers are used in a wide variety of data storage structures ... We define pointer here only because you need to know what a pointer is for understanding file organizations. You will likely never work directly with pointers because the DBMS will handle all pointer use and maintenance automatically" – **Modern Database Management 9th Ed. Jeffrey A. Hoffer (2009)**

# Procesamiento y optimización de Consultas

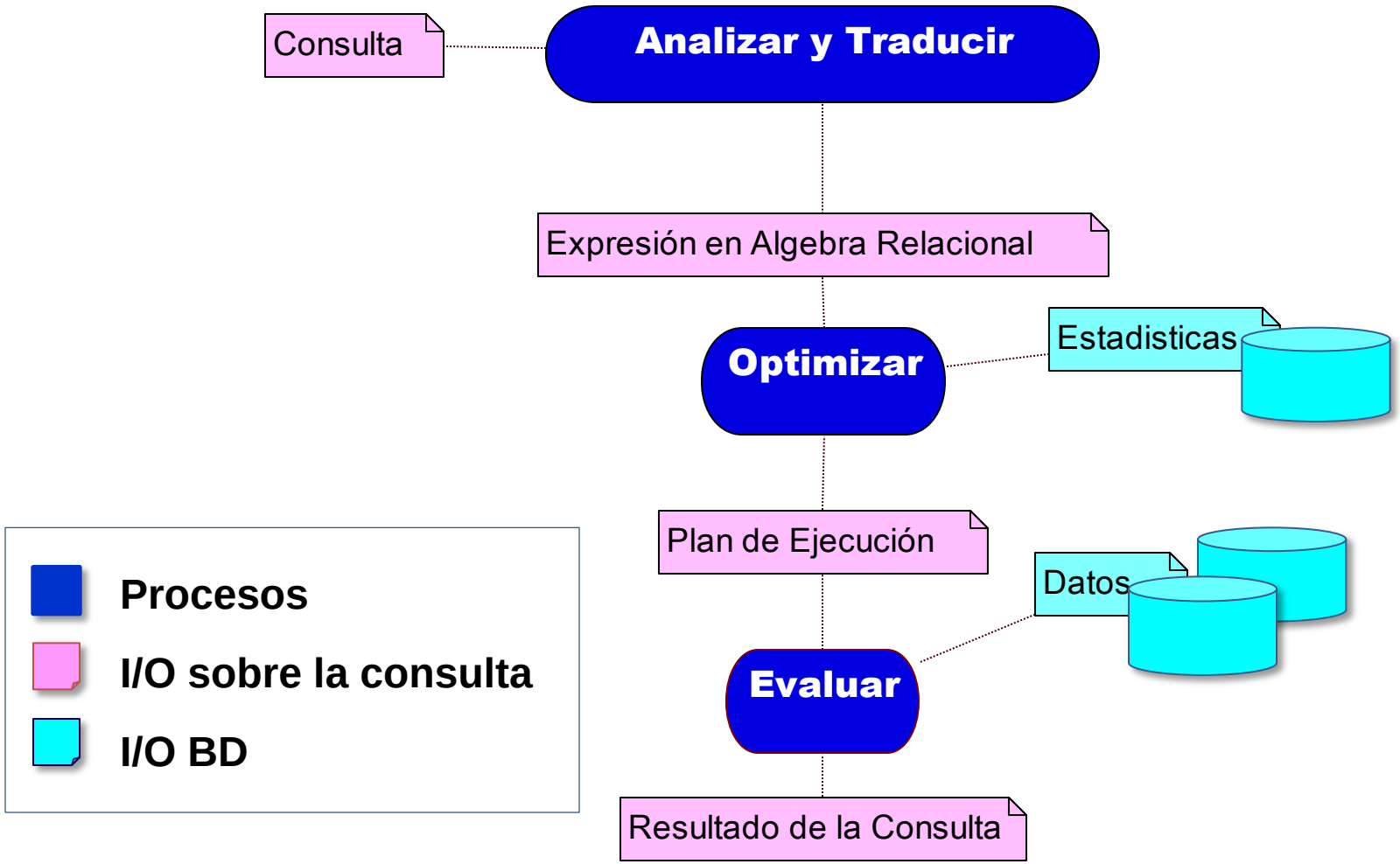




Usuarios Ingenuos
Programadores de Aplicación
Usuarios Sofisticados
Administrador de la BD
Usuarios
Interfaces de Aplicación
Programas de Aplicación
Consultas
Esquema de Base de Datos
Precompilador Embebido de LMD
Compilador de LMD
Intérprete de LDD
Gestor de Consultas
Código Objeto de Prog.de Aplicación
Motor de Evaluación de Consultas
SMBD
Gestor de Transacciones
Gestor de Buffers
Gestor de Archivos
Gestor de Almacenamiento
Archivos de Datos
Indices
Datos Estadísticos
Diccionario de Datos
Almacenam.

# El Compilador de Consultas

1. *Parsing*, ie, análisis sintáctico de la consulta.
2. *Reescribir la consultar.* Definir un plan inicial para la consulta (una representación algebraica de la consulta)
   - ¿Cuál de las formas algebraicamente equivalentes de una consulta conduce a algoritmos eficiente para responder a la consulta?
3. *Generar el Plan Físico*. El plan físico de ejecución selecciona los algoritmos para implementar cada uno de los operadores. Incluye decidír cómo acceder a los datos
   - ¿Existe un índice para acceder? ¿Cuál? ¿Es beneficioso su uso?
   - ¿Cómo se las operaciones van a pasar los datos de una a la otra, por ejemplo, pipeline, en el buffer de memorial, o vía el disco?

# El Compilador de Consultas



# Procesamiento de Consultas

Actividades involucradas:

1. **Análisis y traducción**:
   - Se comprueba sintaxis,
   - Se traduce la consulta a una representación interna basada en el *Álgebra Relacional* (AR).
2. **Optimización**
   - En base a una *expresión en AR optimizada, índices existentes y datos estadísticos*, especifica cómo evaluar cada operación.
3. **Evaluar**
   - Medir las alternativas y ejecutar el plan más adecuado.
   - Una mala estrategia de ejecución puede resultar en una consulta que tome mucho más tiempo del necesario.

# Traducción a AR

- Una consulta puede tener distintas expresiones en el álgebra relacional.

**SELECT** saldo
**FROM** *cuentas*
**WHERE** saldo>= 50

$\sigma_{saldo >= 50}(\pi_{saldo}(cuentas))$

ó

$\pi_{saldo}(\sigma_{saldo >= 50}(cuentas))$

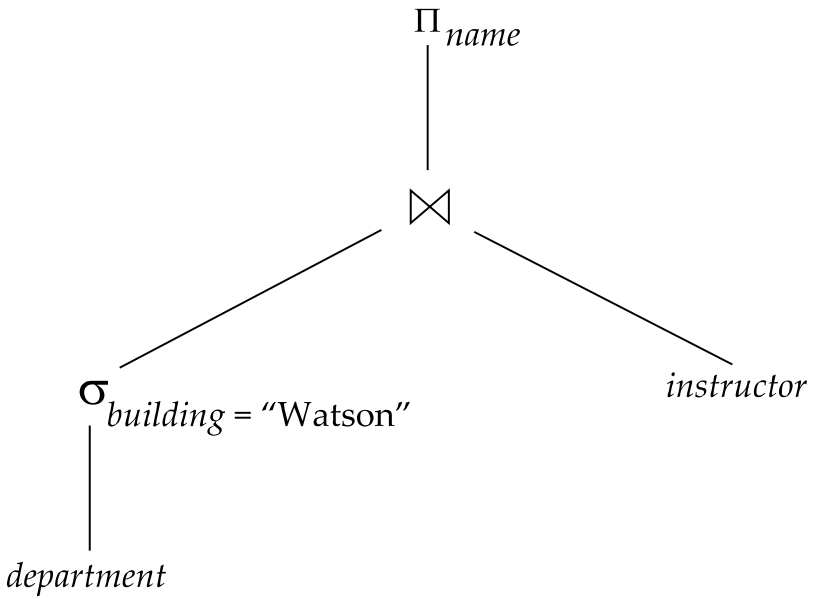
Estrategia general:

1. Producto cartesiano entre las relaciones del **FROM**
2. Selección con las condiciones del **WHERE**.
3. Realizar una proyección de las columnas del **SELECT**

} *árbol canónico*

# Árbol y Plan de Ejecución

- Dada una consulta, se construye un **árbol** que consta de expresiones del AR.
- Las hojas contienen a las relaciones y los nodos interiores las operaciones algebraicas.
- A partir de la expresión del AR se arma un **plan de ejecución** agregando **anotaciones**




**CONSULTA SQL ⇨ VS. ÁRBOLES ⇨ VS. PLANES DE EJECUCIÓN**

# Optimizador de Consultas

¿Por qué existen varios planes de ejecución para una consulta SQL?

- Porque *algebraicamente se puede escribir de maneras distintas* lógicamente equivalentes.
- Y porque, dada una expresión algebraica *existen distintos algoritmos "físicos" para resolverla.*

- El **optimizador de consultas** es el componente del DBMS responsable de generar distintos planes
- Al evaluar se elige un plan de ejecución eficiente.

# Costo de una consulta

- En el costo de evaluar una consulta intervienen varios factores: tiempo acceso a disco, tiempo de transferencia de datos, tiempo de CPU, capacidad de ejecución en paralelo o distribuida, etc.

"In large database systems, **the cost to access data from disk is usually the most important cost**, since disk accesses are slow compared to in-memory operations. Moreover, CPU speeds have been improving much faster than have disk speeds. Thus, it is likely that the time spent in disk activity will continue to dominate the total time to execute a query. The CPU time taken for a task is harder to estimate since it depends on low-level details of the execution code. Although real-life query optimizers do take CPU costs into account, for simplicity in this book we ignore CPU costs and use only disk-access costs to measure the cost of a query-evaluation plan."

# Costo de una consulta

We use the ***number of block transfers*** from disk and the ***number of disk seeks*** to estimate the cost of a query-evaluation plan. If the disk subsystem takes an average of $t_T$ **seconds to transfer a block of data**, and has ***an average block-access time*** (disk seek time plus rotational latency) of $t_S$ seconds, then an **operation that transfers *b* blocks and performs *S* seeks would take $b * t_T + S * t_S$ seconds**. The values of $t_T$ and $t_S$ must be calibrated for the disk system used, but typical values for high-end disks today would be $t_S$ = 4 milliseconds and $t_T$ = 0.1 milliseconds, assuming a 4-kilobyte block size and a transfer rate of 40 megabytes per second. We can refine our cost estimates further by distinguishing block reads from block writes, since block writes are typically about twice as expensive as reads (this is because disk systems read sectors back after they are written to verify that the write was successful).

**Database System Concepts 6th Ed. A. Silberschatz"**

# Optimización

**Objetivo de la optimización**: *"minimizar los accesos y transferencias desde disco"*, para ello se debe considerar:

- Adecuada *expresión algebraica.*
- Diversos *algoritmos disponibles para procesar consultas algebraicas* que implementan las operaciones del AR (selección, proyección, unión, intersección, join, diferencia)
- *Distribución y organización física de los datos,* hash, clustering, sin orden, etc.
- *Índices existentes*.
- *Información estadística existente,* tamaño de los archivos, cantidad de registros, cantidad de registros por bloque, cantidad de bloques, etc.

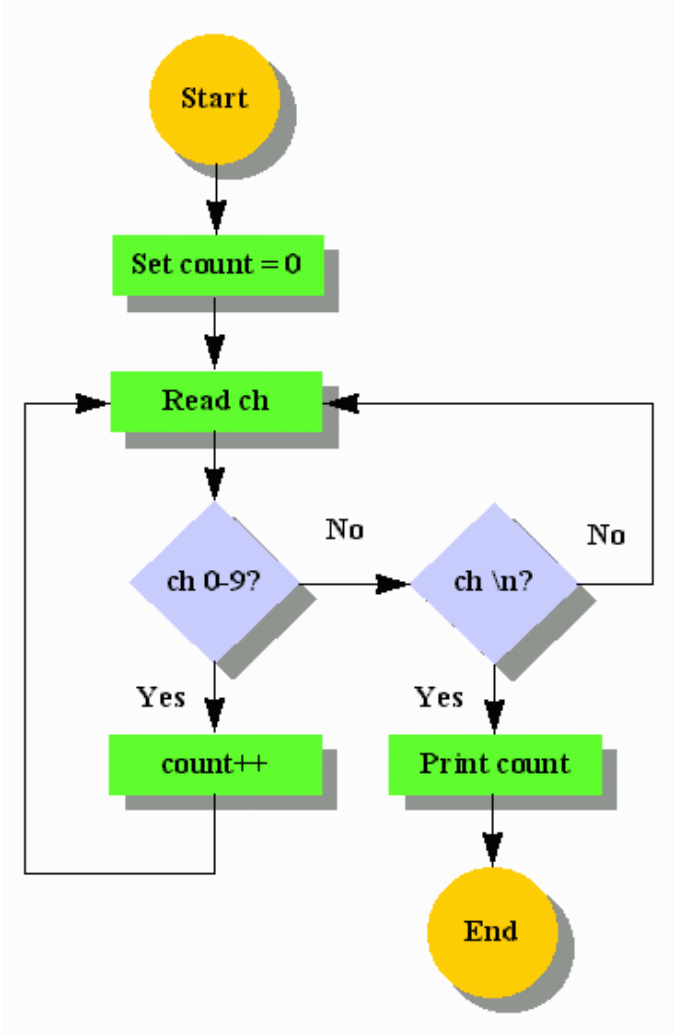
# Optimización

- Cada **plan de ejecución** se construye especificando estrategias o **anotaciones**:

Por ejemplo:

  - Evaluar la consulta usando el índice secundario $i_j$ sobre saldo.
  - Evaluar la consulta haciendo recorrido secuencial sobre la relación

- Entre los distintos planes de ejecución posibles el optimizador elige el que se estima más económico.




# Algunos Algoritmos para resolver Expresiones Algebraicas

# Medidas de Costo

Típicamente, el acceso a disco predomina en el costo para resolver una consulta. Considerando:

- **b** número de bloques a transferir.
- **S** número de bloques a buscar.
- $t_T$ tiempo para transferir un bloque.
- $t_s$ tiempo para la búsqueda de un bloque (localización del bloque en disco).

El costo de transferir **b** bloques con sus **S** búsquedas

$$b * t_T + S * t_s$$

# Simplificación

Una estimación más acertada debería considerar otros factores, que no son tenidos en cuenta en una análisis de propósito general.

- Otros factores no considerados:
  - Velocidad del procesador.
  - Tamaño de buffer.
  - Si los datos han sido usados recientemente.
- Las estimaciones son sobre el peor caso.

# Referencias usadas

- **b** número de bloques a transferir.
- **S** número de búsquedas
- $\mathbf{t_T}$ tiempo para transferir un bloque
- $\mathbf{t_s}$ tiempo para una búsqueda en disco
- $\mathbf{b_r}$ número de bloques ocupados por la relación *r*.
- $\mathbf{h_i}$ peso del índice B+-tree.

# Selección + File Scan

**Estimación para File scan:** se considera una operación de selección sobre una relación cuya tuplas están almacenadas consecutivamente.

- **Algoritmo A1** (búsqueda lineal): recorrer secuencialmente toda la relación y verificar registro a registro si satisface la condición de selección.
  - **Costo estimado: $t_s + b_r * t_T$**
  - Si la selección **es igualdad sobre un atributo clave candidata**: **$(b_r/2) * t_T + t_s$**

- Características: File scan se puede utilizar *cualquiera sea la condición de selección, el orden de los registros de datos en el archivo y disponibilidad de índices*.

# Selección + Indice B+Tree

## Index Scan – Algoritmo de búsqueda con índice primario

⇨ **Algoritmo A2 –** índice primario (clustering), condición: igualdad sobre atributo clave candidata – ie, se espera retornar un único registro como respuesta.

- **Costo estimado:** $(h_i + 1) * (t_T + t_S)$

```
SELECT *
FROM personas
WHERE id="12345"
```

⇨ **Algoritmo A3 –** índice primario (clustering), condición: igual sobre atributo *no clave candidata* – ie. se espera recuperar varios registros.

- Los registros están consecutivos, sea b el número de bloques que contienen los registros que satisfacen la condición de igualdad:
- **Costo estimado:** $h_i * (t_T + t_S) + t_S + t_T * b$

```
SELECT *
FROM personas
WHERE apellido= "Perez"
```

# Selección + Índice Secundario B+Tree

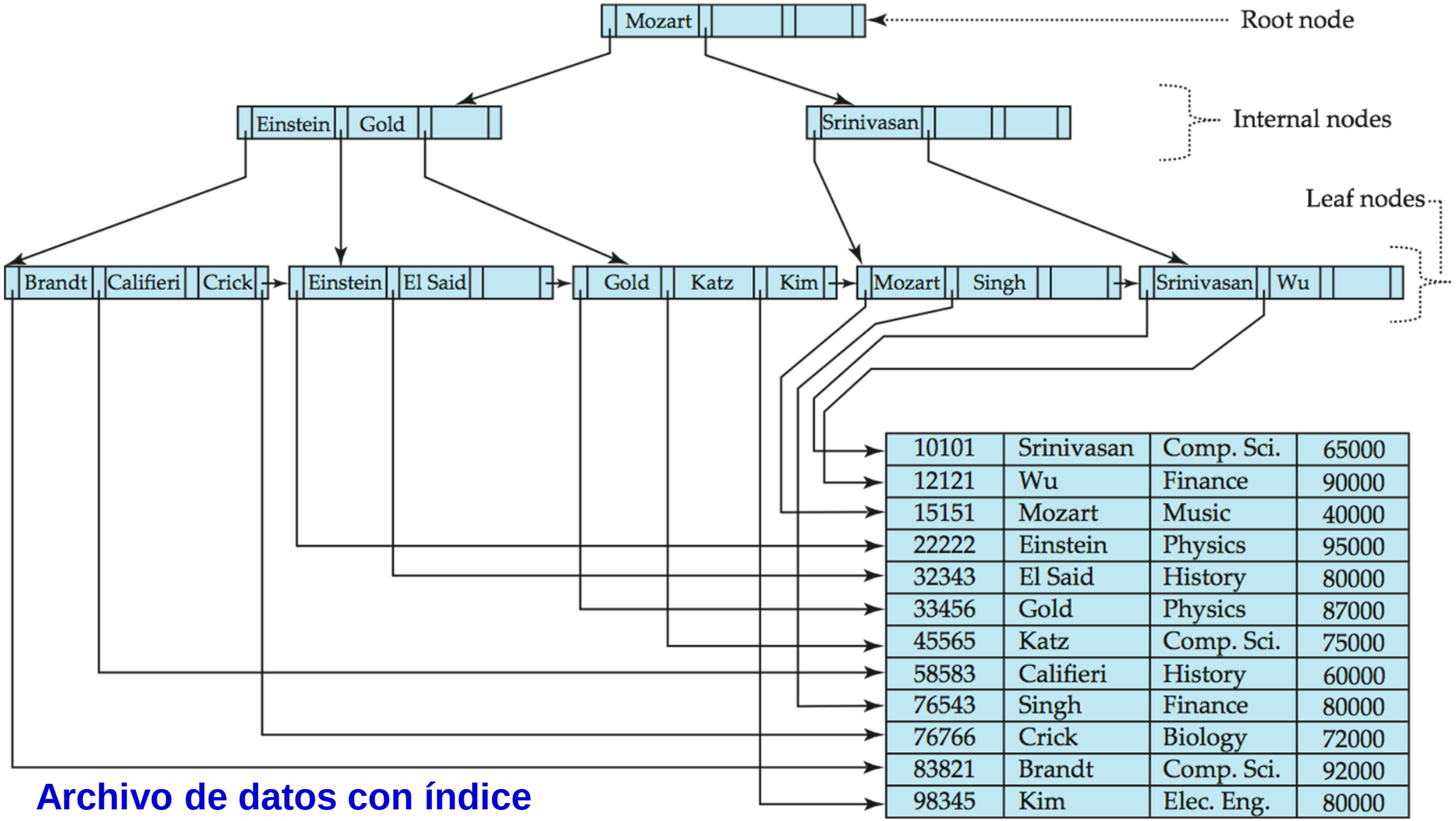
## Index Scan – Con índice secundario.

- **Algoritmo A4** - índice secundario, condición por igualdad. Los registros de datos una distribución distinta a la de la clave del índice.
  - Para **un índice secundario único**, recupera un único registro.
  - **Costo estimado:** $(h_i + 1) * (t_T + t_S)$

```
SELECT *
FROM personas
WHERE id="12345"
```

  - **Indice secundario no único**. La **igualdad** espera recuperar *n* registros. En el peor caso, cada uno de los *n* registros que coinciden con la búsqueda están en bloques distintos.
  - **Costo estimado:** $(h_i + n) * (t_T + t_S)$

```
SELECT *
FROM personas
WHERE apellido= "Perez"
```




**Archivo de datos con índice secundario B+Tree (no clustering)**

# Selección por rango + B+Tree

**Selección y comparación: a <= v (ó <, >, >=) usando índices.**

- **Algoritmo A5 –** índice primario, selección por comparación.
  - >=, usar el índice para encontrar la primera tupla. Seguir recorriendo el archivo de datos en forma secuencial.
  - <= recorrer secuencialmente el archivo de datos hasta alcanzar la primera tupla que es mayor a valor.
  - **Costo estimado:** $h_i * (t_T + t_S) + b * t_T$
- **Algoritmo A6** – índice secundario, comparación.
  - Recorrer secuencialmente el índice, por cada entrada de índice que cumple la condición y seguir su puntero.
  - **Costo estimado:** $(h_i + n) * (t_T + t_S)$
  - **Conviene usar A6 si el número de registros a recuperar es bajo, sino es mejor recorrer secuencialmente el archivo de datos (A1).**

- Ref.

Database System Concepts

A. Silberschatz (cap. 12)

| | Algorithm | Cost | Reason |
|---|---|---|---|
| A1 | Linear Search | $t_S + b_r * t_T$ | One initial seek plus $b_r$ block transfers, where $b_r$ denotes the number of blocks in the file. |
| A1 | Linear Search, Equality on Key | Average case $t_S + (b_r/2) * t_T$ | Since at most one record satisfies condition, scan can be terminated as soon as the required record is found. In the worst case, $b_r$ blocks transfers are still required. |
| A2 | Primary $B^+$-tree Index, Equality on Key | $(h_i + 1) * (t_T + t_S)$ | (Where $h_i$ denotes the height of the index.) Index lookup traverses the height of the tree plus one I/O to fetch the record; each of these I/O operations requires a seek and a block transfer. |
| A3 | Primary $B^+$-tree Index, Equality on Nonkey | $h_i * (t_T + t_S) + b * t_T$ | One seek for each level of the tree, one seek for the first block. Here $b$ is the number of blocks containing records with the specified search key, all of which are read. These blocks are leaf blocks assumed to be stored sequentially (since it is a primary index) and don't require additional seeks. |
| A4 | Secondary $B^+$-tree Index, Equality on Key | $(h_i + 1) * (t_T + t_S)$ | This case is similar to primary index. |
| A4 | Secondary $B^+$-tree Index, Equality on Nonkey | $(h_i + n) * (t_T + t_S)$ | (Where $n$ is the number of records fetched.) Here, cost of index traversal is the same as for A3, but each record may be on a different block, requiring a seek per record. Cost is potentially very high if $n$ is large. |
| A5 | Primary $B^+$-tree Index, Comparison | $h_i * (t_T + t_S) + b * t_T$ | Identical to the case of A3, equality on nonkey. |
| A6 | Secondary $B^+$-tree Index, Comparison | $(h_i + n) * (t_T + t_S)$ | Identical to the case of A4, equality on nonkey. |

EBD

**Figure 12.3** Cost estimates for selection algorithms.

# Join – Loop anidado

**Algoritmo $r \triangleright\triangleleft s$ – Loop anidado**

PARA cada $t_r$ in $r$ HACER

  PARA cada $t_s$ en $s$ HACER

    SI ($t_r$,$t_s$) satisface condición de join ENTONCES

      Agregar ($t_r$,$t_s$) al resultado

    FIN-SI

  FIN-PARA

FIN-PARA

*student* 5000 registros en 100 bloques
*takes* 10000 registros en 400 bloques

# *student* ▷◁ *takes*

No hay índices para *student* ni *takes.* Análisis del **algoritmo join con loop anidado.** Se deben examinar $5000 * 10000 = 50 * 10^6$ pares de tuplas.

**Mejor caso**: "cargar" en memoria ambas relaciones una sola vez. Se requieren a lo sumo 100+400 = 500 transferencias ($t_T$) de bloque y 2 accesos a disco ($t_S$).

Con ***student*** como **relación externa y *takes*** como **relación interna**

**Peor caso:** bloques a transferir ($t_T$) $5000 * 400 + 100 = 2.000.100$, más $5000+100 = 5100$ accesos a disco ($t_S$).

200010
+20400

Con ***takes*** como **relación externa y *student*** como **relación interna**

**Peor caso:** $t_T$ $10000 * 100 + 400 = 1.000.400$, con $t_S$ $10000 + 400 = 10400$.

$t_T$ es significativamente menor, y creció $t_S$.

100040+
41600

Asumiendo $t_S$ = 4 milisegundos and $t_T$ = 0.1 milisegundos takes como relación externa tiene mejor desempeño

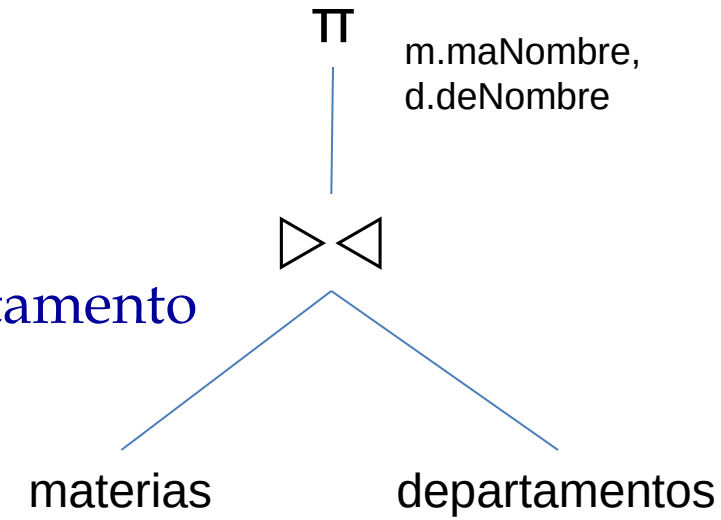
# Otros algoritmos

- **Conjunción**: $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$
- **Disyunción:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r)$
- **Negación:** $\sigma_{\neg \theta}(r)$
- **Ordenamiento,**
- **Join**

# Plan de ejecución

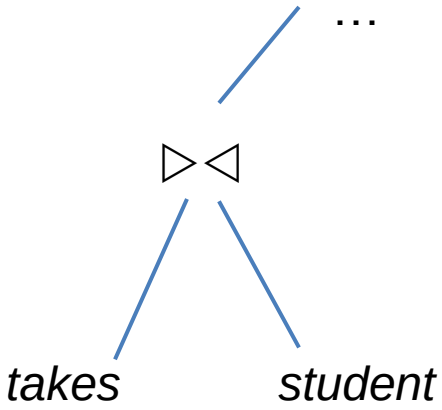
- Para la resolución de consultas primero se construyen representaciones algebraicas, en la forma de árbol.
- Por ejemplo:

```
SELECT m.maNombre, d.deNombre
FROM materias m, departamentos d
WHERE m.departamento = m.departamento
```



# Árbol Algebraico

- En las hojas están las *relaciones*.
- Los nodos interiores representan *operaciones algebraicas* (π, σ, ▷◁, etc.)
- Un *arco hacia los sucesores* representa los parámetros de entrada del sucesor.
- Un *arco hacia su antecesor* representa el resultado o output de la operación.
- La salida de la raíz es el resultado final de la expresión.

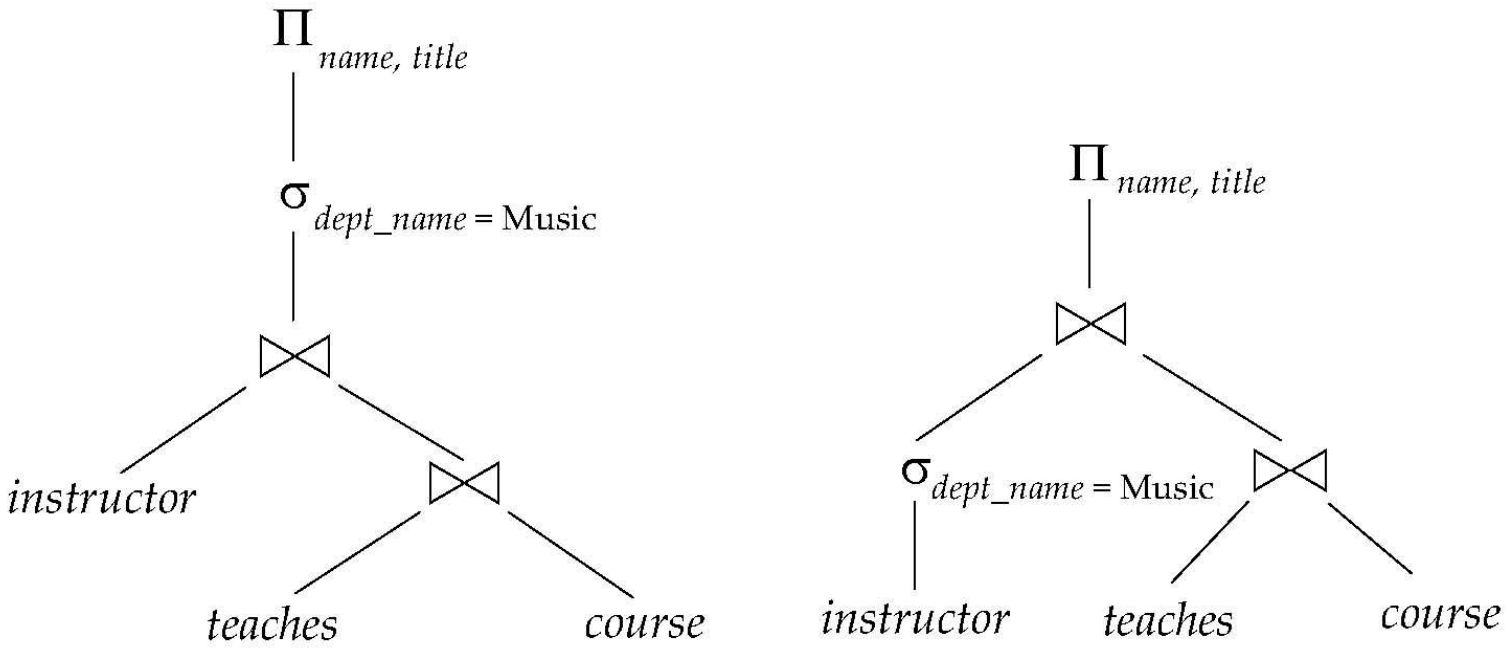



Evaluaciones:

- Cada operación algebraica tiene un ó más algoritmos que la resuelven.
- Cada relación tiene una organización y métodos de acceso.

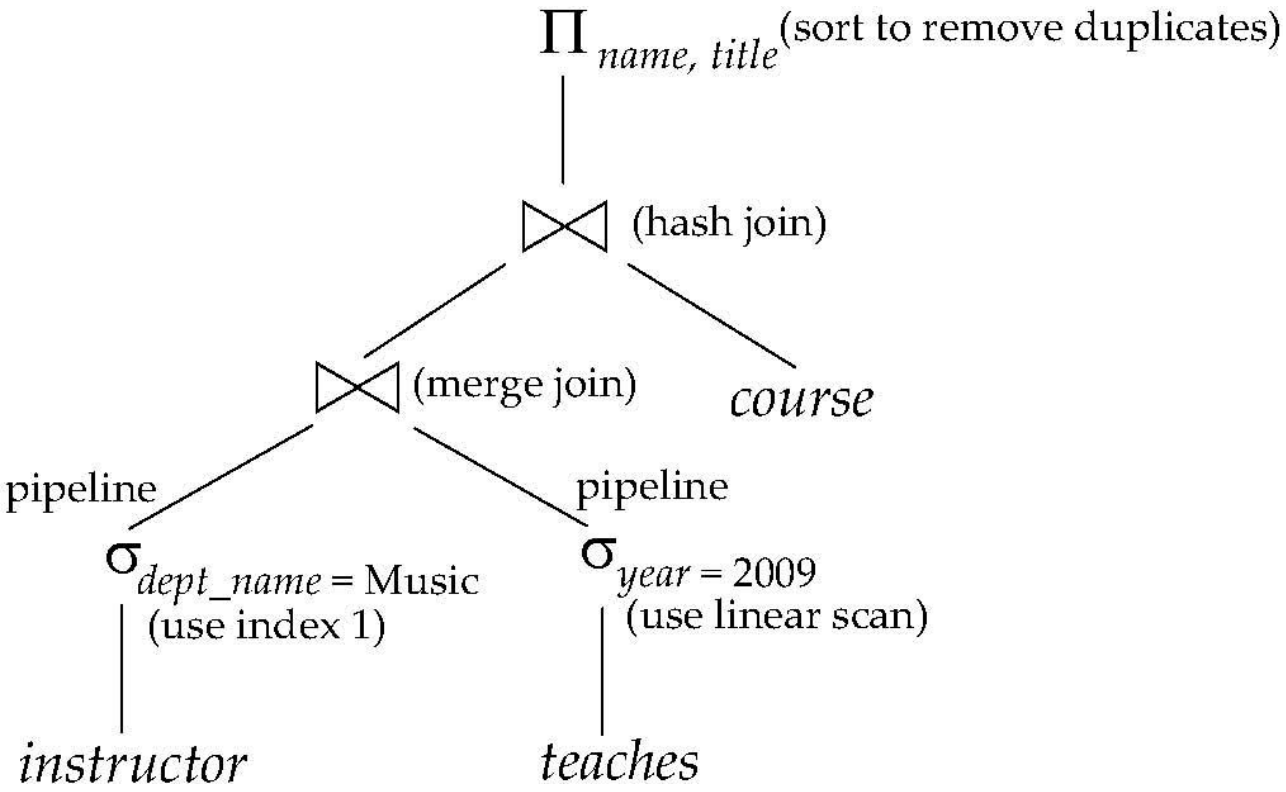
# Árbol algebraico

- Expresiones equivalentes implicaran diferentes algoritmos y planes y costos distintos.
- Ejemplo:



# Plan de ejecución

- En el *plan de ejecución* se define además exactamente los algoritmos a usar en cada operación.



# Observaciones generales

- La diferencia de costo de distintos planes de ejecución puede ser muy grande.
- Pasos para optimización de consultas basada en costos:
  1. Generar expresiones lógicamente equivalentes, usando reglas de equivalencia.
  2. Agregar anotaciones con las alternativas.
  3. Elegir el plan más económico basado en el costo estimado.
- Herramientas para la estimación de costos:
  - Información estadística de las relaciones.
  - Estimación estadística para resultados intermedios.
  - Costo estimado de los algoritmos

# Reglas de equivalencias

1. **Cascada de σ:** $\sigma_{\theta 1 \wedge \theta 2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$
2. **Conmutativa de σ:** $\sigma_{\theta 1}(\sigma_{\theta 2}(E)) = \sigma_{\theta 2}(\sigma_{\theta 1}(E))$
3. **Cascada de π:** $\pi_{L1}(\pi_{L2}(E)) = \pi_{L1}(E) = \pi_{L1 \cap L2}(E)$
4. **Conmutativa de σ con respecto a π:** $\pi_X(\sigma_C(E)) = \sigma_C(\pi_X(E))$, si C referencia a atributos de X
5. **Conmutativa del Producto Cartesiano (y Join):** $E_1 X E_2 = E_2 X E_1$
6. **Distributiva σ y Producto Cartesiano (y Join):** $(\sigma_C(E_1 X E_2)) = \sigma_{C1}(E_1) X \sigma_{C2}(E_2)$, con $C = C_1 U C_2$.
7. **Distributiva π y Producto Cartesiano:** $(\pi_L(E_1 X E_2)) = \pi_{L1}(E_1) X \pi_{L2}(E_2)$, con $L = L_1 U L_2$.
8. …

# Heurísticas

- Elegir árboles sesgados a izquierda.
- Descomponer las selecciones conjuntivas en selecciones simples.
- Llevar la selección lo más cercano a la hoja del árbol.
- Remplazar productos cartesianos seguidos selección por joins. Evitar los productos cartesianos.
- Descomponer la lista de atributos de proyección y llevarlas lo más cercano posible a las hojas.
- Realizar primero los joins más selectivos (resultado menor cantidad de tuplas).
- En cada nodo, optar por los planes menos costosos.
- Considerar los índices.
- Mantener siempre que sea posible **los resultados en memoria** (pipeline)

# Pasos para la optimización

1. Construir el árbol canónico.
2. Construir árboles equivalentes, utilizando alguna heurística.

   No sobredimensionar la búsqueda del mejor plan.
3. Por cada árbol, hacer tantos planes de ejecución como combinaciones interesantes existan.
4. Hacer las anotaciones.

# Ejemplo

- Sean las relaciones
  - Libro (nroLib (4), tituloLib(84), pagLib(8)) – Tamaño registro 96 bytes
  - Autor (nroAut (4), nombreAut(40), nacionalidad (10)) – Tamaño reg 54 bytes
  - Escrito_por (nroLib, nroAut) – Tamaño registro 8 bytes

  - Tamaño de bloque = 4000 bytes
  - Cantidad de Tuplas en *libro* $T_{libro}$ = 100000 registros
  - Cantidad de Tuplas en escrito_por $T_{Escrito\_por}$ = 130000 registros
  - Cantidad de Tuplas en autor $T_{autor}$ = 5000 registros
- Índices
  - índice B+ cluster sobre nroLib para libro – índice sobre atributo clave
  - índice B+ cluster sobre nroLib y nroAut para escrito_por – índice sobre atributo clave
  - índice B+ cluster sobre nroAut para autor – índice sobre atributo clave
  - índice B+ secuendario sobre nacionalidad para autor

# Consulta

```
SELECT l.nroLib, l.tituloLib
FROM autor a, escrito_por ep, libro l
WHERE a.nroAut = ep.nroAut
        AND ep.nroLib = l.nroLib
        AND a.nacionalidad = "argentino"
```

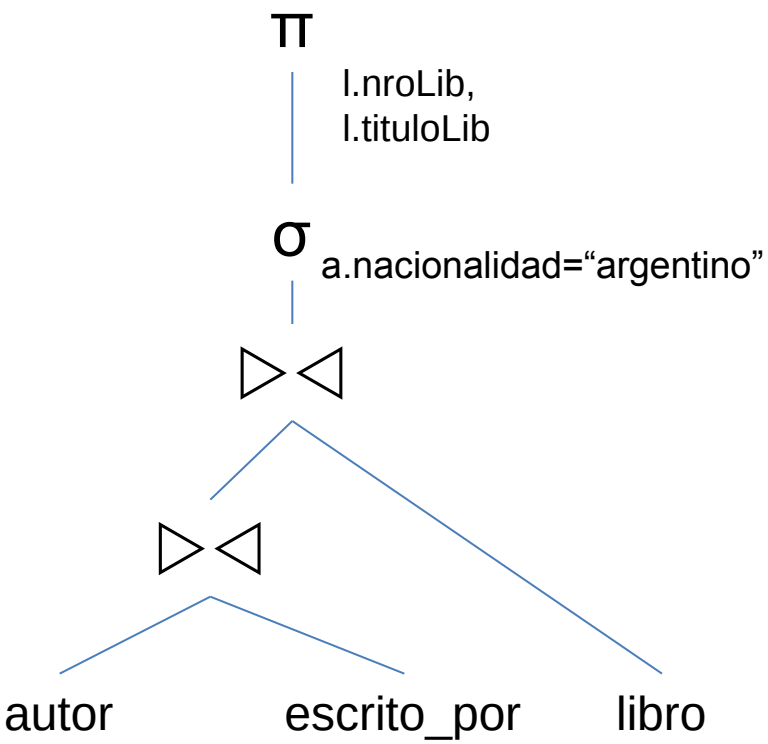
- Consulta canónica

$\pi_{l.nroLib,\ l.tituloLib}\ (\sigma_{a.nacionalidad="argentino"}\ (autor \rhd\lhd escrito\_por \rhd\lhd libro))$

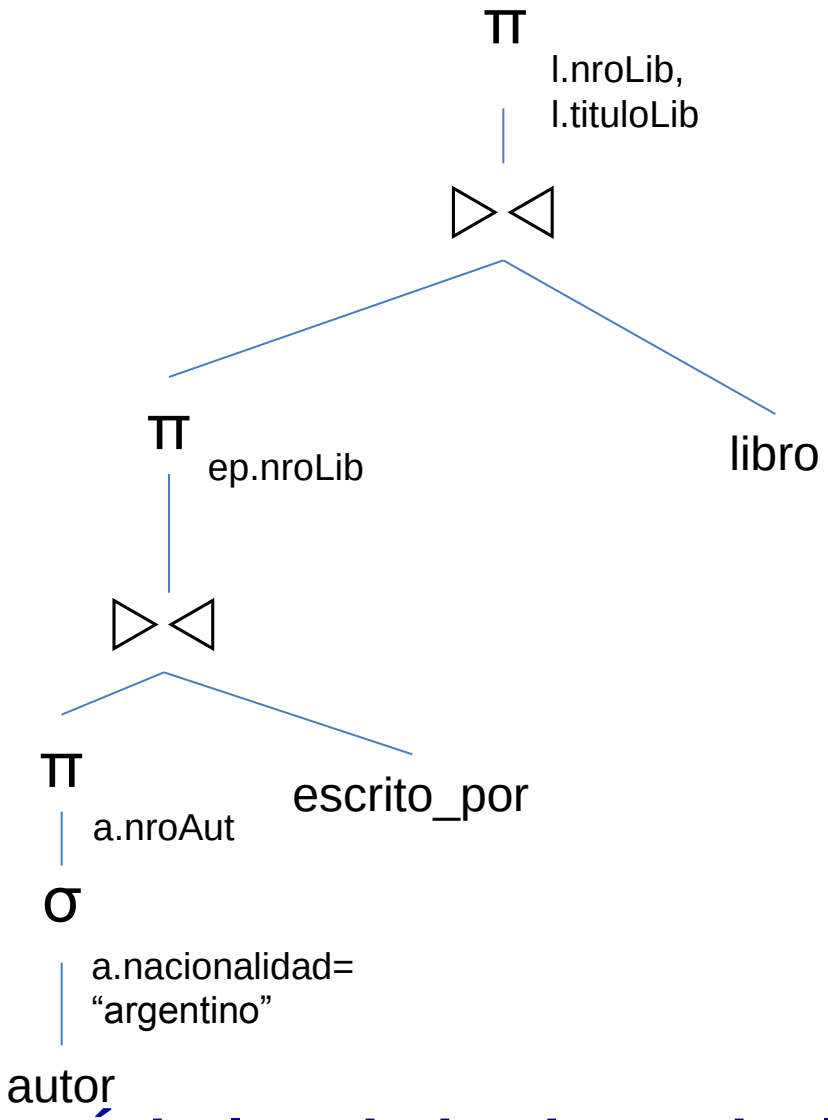
- Consulta equivalente optimizada

$\pi_{l.nroLib,\ l.tituloLib}\ ((libro) \rhd\lhd \pi_{l.nroLib}(\pi_{l.nroAut}(\sigma_{a.nacionalidad="argentino"}\ autor) \rhd\lhd escrito\_por))$

# Árboles




**Árbol Canónico**




**Árbol optimizado equivalente**

# Bloques ocupados

- Longitud de *autor* $L_a$=54 by
- Bloques que ocupa :
  - [tamaño_bloque/ $L_a$] =[4000/54] =74 registros por bloque
  - [5000/74] = **68 bloques.**
- Longitud de *escrito_por* $L_{ep}$=8 by
- Bloques que ocupa escrito_por:
  - [tamaño_bloque/$L_{ep}$] =[4000/8] =500 registros por bloque
  - [130000/500] = **260 bloques.**
- Longitud de *libro* $L_l$=96 by
- Bloques que ocupa libro:
  - [tamaño_bloque/$L_l$] =[4000/96] =41 registros por bloque
  - [100000/41] **= 2440 bloques.**

# Costo estimado por índices

- índice B+ cluster sobre nroLib para *libro* – índice sobre atributo clave
  - Tamaño registro índice 12
  - [4000 / 12] 333 registros índices por bloque.
  - 2440 bloques de datos, nivel del árbol 2

- índice B+ cluster sobre nroLib y nroAut para escrito_por (nroAut+nroLib)– índice sobre atributo clave
  - Tamaño registro índice 16
  - [4000 / 16]  250 registros índices por bloque.
  - 260 bloques de datos, nivel del árbol 2

# Costo estimado por índice

- índice B+ cluster sobre nroAut para *autor* – índice sobre atributo clave.
  - Tamaño registro índice 12
  - [4000 / 12] 333registros índices por bloque.
  - 68 bloques de datos, nivel del árbol 1.
- índice B+ secundario sobre nacionalidad para autor
  - Tamaño registro índice 18
  - [4000 / 18] 222 registros índices por bloque.  1 bloque para el índice y un bloque por indirección.
  - Suponiendo una distribución  equitativa entre 50 nacionalidades, 100 autores por nacionalidad.

# Análisis del plan ejecución

$\sigma_{\text{nacionalidad= “argentino”}}$

- Costo File scan = **68 bloques ✓ -> $t_S$ + 68 * $t_B$**
- Costo usando índice secundario: 1 acceso al índice + 1 acceso indirección + en el peor de los casos cada autor exige traer el bloque lo aloja, 102 accesos a bloque. -> $2t_S$ + 100 * ($t_S$ + $t_B$)

- $\pi_{\text{nroAut}}$ pipeline en memoria

# Análisis plan de ejecución

nroAut ▷◁ escrito_por

- Resultado anterior en memoria. Costo join con FileScan = 260 bloques para 100 autores -> 100 * ( $t_S$ + 260 * $t_B$)

- Usando el índice B+ de escrito_por, los datos están en el índice y asumiendo distribución uniforme, a lo sumo 100 autores, por dos niveles, **200 bloques** para el peor de los casos ✓ **->** 200 * ( $t_S$ + $t_B$)

- $\pi_{nroLib}$ pipeline en memoria

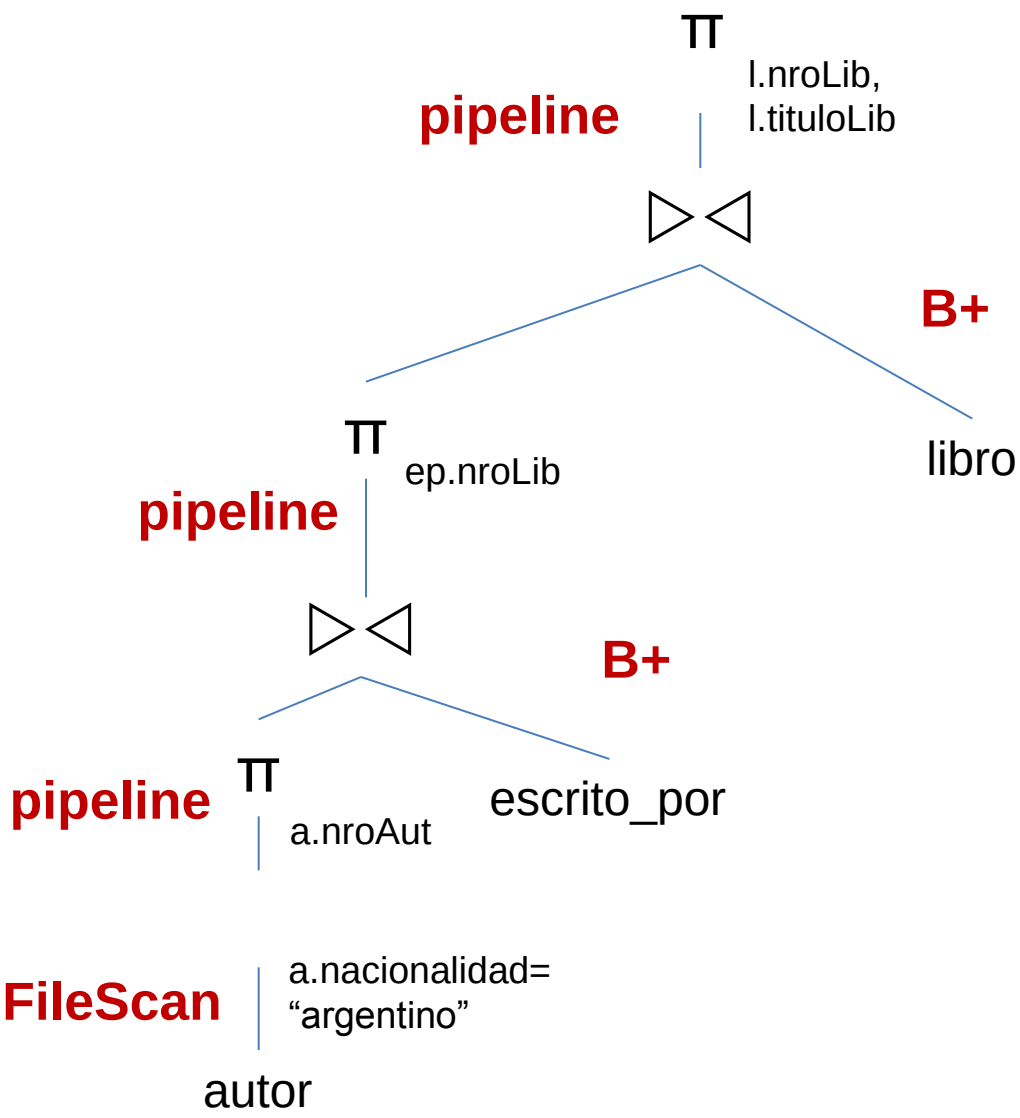
# Análisis del plan de ejecución

nroLib ▷◁ libro

- Resultado anterior en memoria. Costo join con FileScan = 2600 * 2440/2 bloques -> 2600 * ( $t_S$ + 2440/2 * $t_B$)

- Usando el índice B+ de libros y asumiendo distribución uniforme, a lo sumo 5000 autores en 50 nacionalidades, 100 autores argentinos, 130000 escrito_por de 5000 autores, 26 libros por autor. Libros para la operación 2600, con nivel de arbol 2, **(1 + 1) * 2600 bloques** para el peor de los casos -> 2600 * 2 ( $t_S$ + $t_B$) ✓

# Plan de Ejecución propuesto



# Temas de la clase de hoy

- Procesamiento de consultas
  - Etapas
  - Algoritmos
- Optimización de consultas
  - Expresiones algebraicas equivalentes.ç
  - Plan de ejecución
  - Ejemplo

- **Bibliografía**:
  - *Database System Concepts* – Abraham Silberschatz – Capítulos 12 y 13 (ed. 2010)
  - *DataBase System – The Complete Book* – H. Molina, J. Ullman. Capítulo 15 y 16.