Optimización de Consultas

Autor: Sergio D'Arrigo

Motivación

- Supongamos que tenemos las siguientes consultas SQL…

```sql
SELECT E.apellido, E.nombre, D.nombre division
FROM EMPLEADO E
    LEFT JOIN DIVISION D
        ON E.id_division = D.id_division
WHERE E.anio_ingreso>=2020
;
```

```sql
SELECT E.apellido, E.nombre, D.nombre division
FROM DIVISION D
    LEFT JOIN EMPLEADO E
        ON E.id_division = D.id_division
WHERE E.anio_ingreso>=2020
;
```

- ¿cómo hace el motor de Base de Datos para resolverlas?
- ¿Qué estrategia usará en cada caso?
- ¿Una de las dos tardará más que la otra?
- Si tardan mucho… ¿se justifica agregar un índice para que resuelva más rápido?

- En esta parte trataremos de mostrar cómo hace el motor de BBDD para resolver una consulta eficientemente.

Repasando conceptos generales…

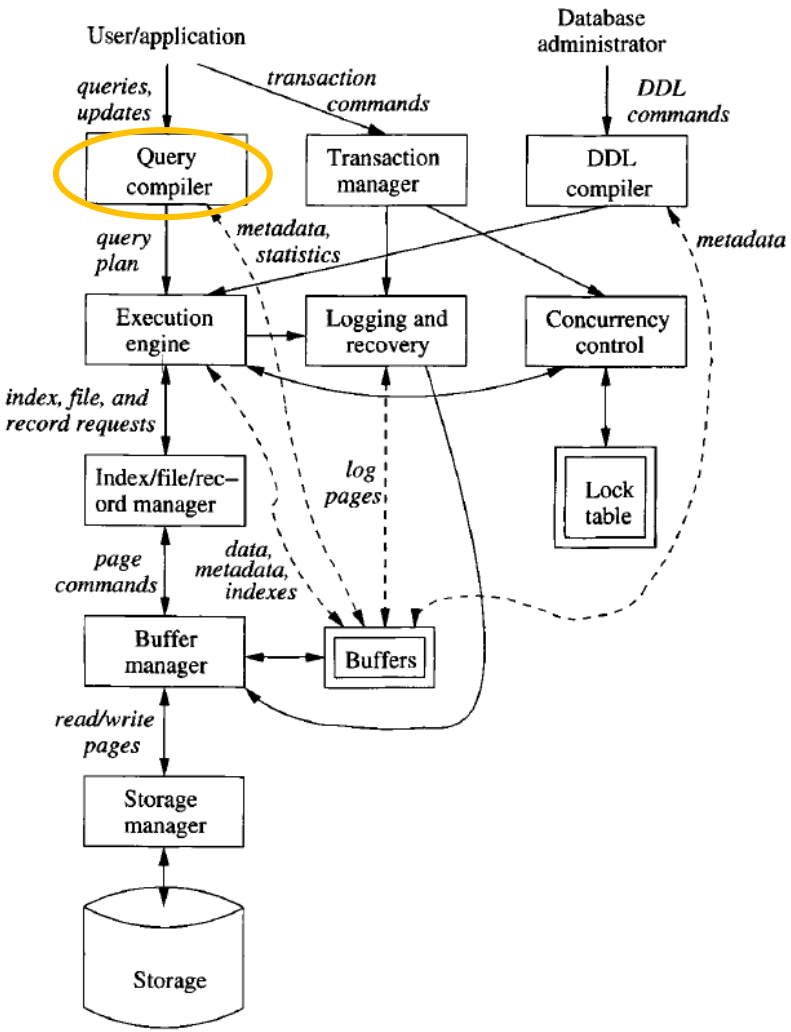


Figure 1.1: Database management system components

Figura de García Molina- Ullman-Widom "Database Systems: The Complete Book"

- El motor recibe una consulta escrita en lenguaje SQL.
- La consulta es parseada y optimizada por el **Compilador de Consultas**, generando como resultado un **Plan de Ejecución**.
  - *Plan de Ejecución: secuencia de acciones que debe realizar el DBMS para resolver la consulta*
- El Plan de Ejecución se le pasa al Motor de Ejecución, el cual genera una secuencia de pedidos de datos (acorde al plan de ejecución) al Gestor de Recursos.

(Y sigue…)

Repasando conceptos generales…

Veamos al Compilador de Consultas con más detalle..

Compilador de Consultas

- Traduce la consulta externa en un formato interno llamado Plan de Ejecución (secuencia de acciones que debe realizar el DBMS para resolver la consulta).

Consiste de 3 principales unidades:

- ***Parser de Consultas***: construye una estructura de árbol que representa la consulta recibida de forma textual
- ***Preprocesador de Consultas***: realiza chequeos semánticos y realiza transformaciones al árbol para convertirlo en un árbol de operaciones algebraicas que representan el plan de ejecución inicial
- ***Optimizador de Consultas***: transforma el plan de ejecución inicial en una secuencia optimizada de operaciones sobre los datos actuales.

✍ *El Compilador de Consultas utiliza metadatos y estadísticas de la Base de Datos para resolver cuál secuencia de operaciones sería más eficiente.*

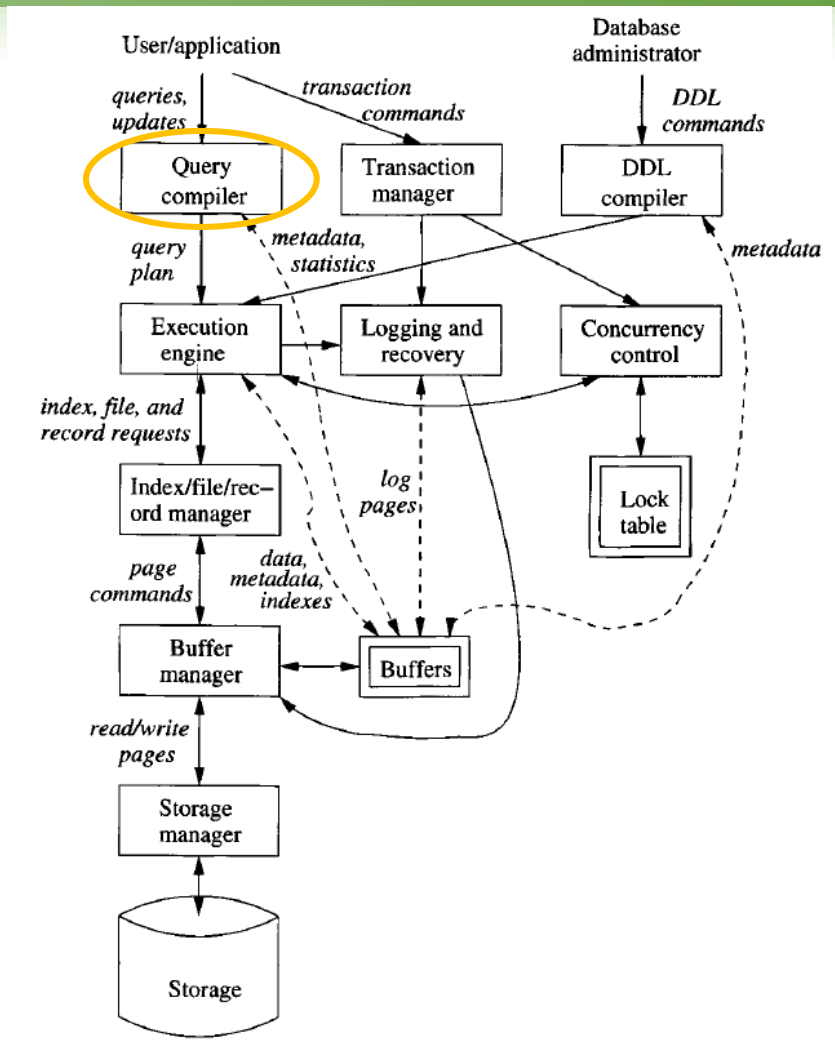


Figure 1.1: Database management system components

Figura de García Molina- Ullman-Widom "Database Systems: The Complete Book"

Estrategias de Procesamiento de Consultas

El compilador de consultas, una vez validada la sintaxis y la semántica, genera una representación interna de la misma llamada **Árbol de Consulta.**

A continuación se muestra una consulta SQL y un posible árbol de consulta

```
SELECT m.nomMateria,  d.nomDepto
FROM MATERIA m, DEPARTAMENTO d
WHERE m.cod_depto=d.cod
```

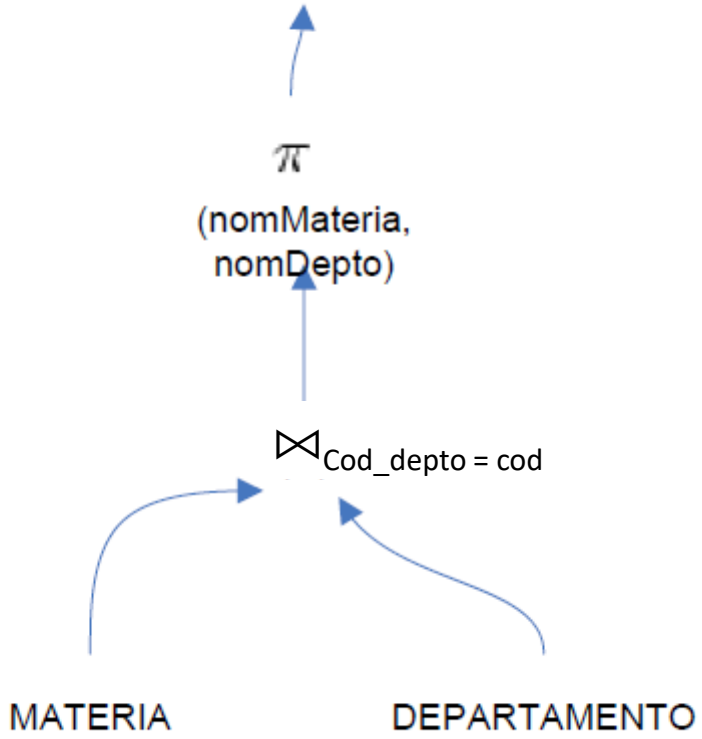


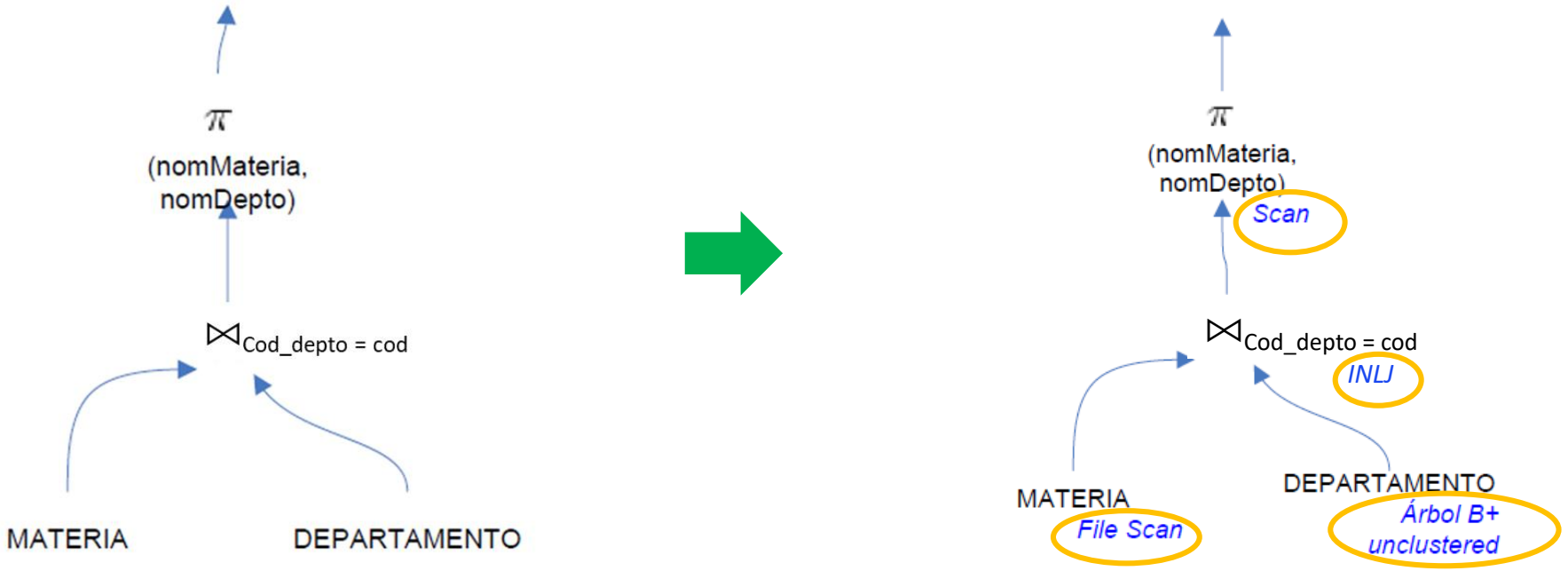
Figura del Apunte de Optimización de Consultas

Estrategias de Procesamiento de Consultas

Luego de la traducción inicial, el compilador de consultas tiene que desarrollar una estrategia concreta de ejecución para extraer los datos de la BBDD: el plan de ejecución.

Cada operación algebraica tiene una o más implementaciones físicas. A cada una de esas implementaciones, lo llamamos **operador físico**.

El plan de ejecución indica una secuencia de operaciones físicas.



Figuras del Apunte de Optimización de Consultas

Veremos primero algunos algoritmos para determinadas operaciones algebraicas, y posteriormente veremos la lógica del optimizador de consultas (simplificada).

Estrategias de Procesamiento de Consultas

Caminos de Acceso: se llama así a los diferentes métodos para recuperar tuplas

- File scan: recorrido sobre el archivo de datos
- Index Scan: recorrido sobre las entradas de un archivo de índice

Coincidencia con un índice (index matching)

Intuitivamente, un índice coincide con una condición de selección si puede ser utilizado para recuperar las tuplas que cumplen esa condición.

Dado un predicado p que representa una condición conjuntiva de búsqueda simple

- ❑Un índice B+ coincide con el predicado p si p es una conjunción de términos de la forma a op c, involucrando a la totalidad de los campos de búsqueda de la clave o a un prefijo de ellos.
- ❑Un índice Hash coincide con el predicado p, si p es una conjunción de términos de la forma a=c involucrando a la totalidad de los campos de búsqueda de la clave.

Algoritmos para operaciones algebraicas

- Para poder resolverlas, los motores deben contar con algoritmos que las implementen.
- El **costo de ejecución** de estos algoritmos se compone de
- **Costo del Input**: es el costo de acceder y procesar los datos. Depende fuertemente de la organización e archivos e índices disponibles. Para una misma operación algebraica puede haber más de una forma de hacerlo.
- **Costo del Output**: sólo depende de la operación algebraica, no depende del mecanismo que se utilice para acceder y procesar los datos.

    La cantidad de tuplas dependerá de la selectividad de sus condiciones

    El tamaño de cada tupla dependerá de los tamaños de los campos que se proyecten

Nuestro Modelo Simplificado

Convenciones de denominación

- B_R: Cantidad de bloques que ocupa R
- FB_R: Cantidad de tuplas por bloque de R (factor de bloqueo)
- L_R: Longitud de una tupla de R
- T_R: Cantidad total de tuplas de R
- $I_{R,A}$: Imagen del atributo A de R. Denota la cantidad de valores distintos de ese atributo en la relación.
- **X**: altura del árbol de búsqueda
- FB_I: Cantidad de entradas por bloque del índice I (factor de bloqueo del índice)
- BH_I: Cantidad de bloques que ocupa el índice i para nodos hoja (el índice debe ser árbol B+). Si no se tiene el dato y se desea calcularlo, se utilizará el criterio de peor caso (cada nodo hoja estará completo en su mínimo, es decir, d/2, donde d es el orden del árbol).
- $MBxB_I$: Cantidad máxima de bloques que ocupa un bucket del índice i (el índice debe ser basado en hashing)
- CBu_I**:** cantidad de buckets del índice I (el índice debe ser basado en hashing)
- **B**: Cantidad de bloques disponibles en la memoria principal

Algoritmos para Proyección (π)

- ¿Qué métodos se les ocurre que se pueden usar si se quiere resolver una operación de proyección simple sobre una relación que está en una única tabla dedicada, sin eliminar repeticiones?

Comencemos por archivo sin índices ni orden…

¿ y si estuviese ordenado?

¿y si tuviese índices B^+?

¿Y si tuviese índices hash?

Algoritmos para Proyección (π)

- Pasando en limpio
- Algoritmos
  - Búsqueda lineal en archivo
  - Búsqueda lineal en índice B^+
  - Búsqueda lineal en índice Hash
- Costo del Output

Algoritmos para Proyección (π)

- Operación: $\pi_{l}(R)$
- Algoritmo **Búsqueda lineal en archivo**
- Precondición: Ninguna
- Descripción: se recorren todos los bloques de la relación input (filescan), y a medida que se recuperan las tuplas se seleccionan los atributos
- Costo del Input (CI) : B_R

Algoritmos para Proyección (π)

- Operación: **$\pi_{l}(R)$**
- Algoritmo **Búsqueda lineal en índice B^{+}**
- Precondición: Todos los atributos a recuperar forman parte de la clave del índice
- Descripción: se accede al primer nodo hoja del índice, y luego se recorren todas las hojas. A medida que se recuperan las entradas de índice se seleccionan los atributos
- Costo del Input (CI) : Costo de llegar al puntero al primer bloque hoja más el costo de recorrer los bloques hoja

$$CI = X - 1 + BH_{l}$$

Algoritmos para Proyección (π)

- Operación: **π_I (R)**
- Algoritmo **Búsqueda lineal en índice Hash**
- Precondición: Todos los atributos a recuperar forman parte de la clave del índice
- Descripción: se recorren todos los bloques de todos los buckets. A medida que se recuperan las entradas de índice se seleccionan los atributos.
- Costo del Input (CI) : Calcularemos el peor caso, que todos los buckets tengan el máximo de bloques.

$$CI = CBu_I * MBxB_I$$

Algoritmos para Proyección (π)

- **Costo de Output - Operación: $\pi_l(R)$**
- Cantidad de tuplas: No estamos considerando eliminación de duplicados, con lo cual tendremos tantas tuplas resultado como tuplas en la relación origen

$$T_Q = T_R$$

- Longitud de las tuplas: Suma de las longitudes de los atributos a proyectar.
  - Si se conocen las longitudes:

$$L_Q = \sum_{ai\ en\ l} \lceil (L_{ai}) \rceil$$

  - Si no se conocen las longitudes, se asumirá que todos ocupan lo mismo en la relación original.

$$L_Q = \lceil L_R * (\#\ atributos\ de\ l\ /\ \#\ atributos\ deR) \rceil$$

- Cantidad de Bloques: calculemos factor de bloqueo y luego los bloques ocupados

$$FB_Q = \lfloor LB / L_Q \rfloor$$

$$B_Q = \lceil T_Q / FB_Q \rceil$$

- Costo del Output (CO) : Costo de persistir el resultado, es la cantidad de bloques que ocupa

$$CO = B_Q$$

Algoritmos para Selección (σ)

- ¿Qué métodos se les ocurre que se pueden usar si se quiere resolver una operación de selección sobre una relación que está en una única tabla dedicada?

Comencemos por archivo sin índices ni orden…

¿ y si estuviese ordenado?

¿y si tuviese índices B^+?

¿Y si tuviese índices hash?

Algoritmos para Selección (σ)

- Pasando en limpio algunos de los algoritmos
- Algoritmos
  - Búsqueda lineal en archivo
  - Búsqueda binaria en archivo ordenado
  - Búsqueda en archivo con índice árbol B^+ clustered
  - Búsqueda en archivo con índice árbol B^+ unclustered
  - Búsqueda archivo con índice hash
- Costo del Output

Algoritmos para Selección (σ)

- Operación: $\sigma_p(R)$
- Algoritmo **Búsqueda lineal en archivo**
- Precondición: Ninguna
- Descripción: se recorren todos los bloques de la relación input (filescan), y a medida que se recuperan las tuplas se verifica la condición p
- Costo del Input (CI) : B_R

Algoritmos para Selección (σ)

- Operación: **$\sigma_p (R)$**
- Algoritmo **Búsqueda binaria en archivo ordenado**
- Precondición:
  - Archivo ordenado según clave k
  - La condición p coincide con el ordenamiento (en el sentido del index matching)
  - La subexpresión coincidente de p es por igualdad de clave o por rango
- Descripción: búsqueda binaria hasta encontrar la primer tupla, luego filescan mientras las tuplas cumplan condición coincidente. Para cada tupla se evalua si cumple la parte no coincidente de p
- Costo del Input (CI) : costo de búsqueda binaria + costo del recorrido

$$CI = \log_2(B_R) + \lceil (sel(p_k) * T_R) / FB_R \rceil$$

- Característica del resultado: la relación output queda ordenada por la misma clave k quela relación R

Algoritmos para Selección (σ)

- Operación: $\sigma_p(R)$
- Algoritmo **Búsqueda en archivo con índice árbol B^+ clustered**
- Precondición:
  - El archivo tiene un índice árbol B^+ clustered según clave k
  - La condición p coincide con la clave del índice (en el sentido del index matching)
  - La subexpresión coincidente de p es por igualdad de clave o por rango
- Descripción: búsqueda por índice hasta encontrar la primer tupla, luego filescan del archivo de datos mientras las tuplas cumplan condición coincidente. Para cada tupla se evalua si cumple la parte no coincidente de p
- Costo del Input (CI) : costo de búsqueda en árbol + costo del recorrido

$$CI = X + 1 + \lceil ((sel(p_k) * T_R) - 1) / FB_R \rceil$$

Si fuese condición igualdad por clave candidata, $CI = X + 1$

- Característica del resultado: la relación output queda ordenada por la misma clave k que el índice utilizado

Algoritmos para Selección (σ)

- Operación: **$\sigma_p(R)$**
- Algoritmo **Búsqueda en archivo con índice árbol B^+ unclustered**
- Precondición:
  - El archivo tiene un índice árbol B^+ unclustered según clave k
  - La condición p coincide con la clave del índice (en el sentido del index matching)
  - La subexpresión coincidente de p es por igualdad de clave o por rango
- Descripción: búsqueda por índice hasta encontrar la primer tupla, luego indexscan mientras las entradas de índice cumplan condición coincidente. Por cada entrada, se tiene que acceder a la tupla, y al hacelo también se evalua si cumple la parte no coincidente de p
- Costo del Input (CI) : costo de búsqueda en árbol + costo de recorrer hojas + costo de acceder a tuplas

$$CI = X + (1 + \lceil((sel(p_k) * T_R) - 1) / FB_I\rceil) + \lceil sel(p_k) * T_R \rceil$$

Si fuese condición igualdad por clave candidata, **$CI = X + 1$**

- Característica del resultado: la relación output queda ordenada por la misma clave k que el índice utilizado

Algoritmos para Selección (σ)

- Operación: **$\sigma_p (R)$**
- Algoritmo **Búsqueda en archivo con índice Hash**
- Precondición:
  - El archivo tiene un índice basado en hashing según clave k
  - La condición p coincide con la clave del índice (en el sentido del index matching)
  - La subexpresión coincidente de p es por igualdad de clave
- Descripción: búsqueda por índice recorriendo los buckets correspondientes a la clave. Por cada entrada, se tiene que acceder a la tupla, y al hacerlo también se evalua si cumple la parte no coincidente de p
- Costo del Input (CI) : costo de recorrer los buckets + costo de acceder a tuplas

$$CI = MBxB_I + \lceil sel(p_k) * T_R \rceil$$

Si fuese condición igualdad por clave candidata, $CI = MBxB_I + 1$

Algoritmos para Selección (σ)

- **Costo de Output - Operación: $\sigma_p (R)$**
- Cantidad de tuplas: Es lo visto al hablar de Estimación de T' en la clase pasada.
  - El caso general es

$$T_Q = \lceil sel(p_k) * T_R \rceil$$

- Longitud de las tuplas: Es la misma que la relación input

$$L_Q = L_R$$

- Cantidad de Bloques: el factor de bloqueo es similar al de la relación input, lo que cambia es la cantidad de tuplas

$$FB_Q = FB_R$$

$$B_Q = \lceil T_Q / FB_Q \rceil$$

- Costo del Output (CO) : Costo de persistir el resultado, es la cantidad de bloques que ocupa

$$CO = B_Q$$

Algoritmos para Juntas (⋈)

- Operación costosa, involucra cruzar dos tablas, pueden ser muy grandes!!
- Para simplificar, veremos el caso de la junta por igualdad (equijoin)
- ¿Qué métodos se les ocurre que se pueden usar si se quiere resolver una operación de equijoin entre dos relaciones, cada una de simple sobre una relación que está en una única tabla dedicada?

Comencemos por archivo sin índices ni orden…

¿ y si estuviese ordenado?

¿y si tuviese índices B^+?

¿Y si tuviese índices hash?

¿Cuándo convendría un caso u otro?

Algoritmos para Juntas (⋈)

- Pasando en limpio (caso equijoin)
- Algoritmos
  - Block Nested Loop Join
  - Index Nested Loop Join
  - Sort Merge Join
- Costo del Output

Algoritmos para Juntas (⋈)

- Operación: **$R \bowtie_p S$**
- Algoritmo **Block Nested Loop Join**
- Precondición:
  - Se tienen B bloques de memoria disponibles
  - B-2 se usarán para el almacenamiento de R, 1 para ir leyendo S y otro para la salida.
- Descripción:

```
Para cada segmento de B-2 bloques de R
        Para cada bloque de S
                Para toda tupla r del segmento de R y tupla s del bloque de S
                        Si r_i == s_j ( o más generalmente, vale p(s) )
                                Agregar <r,s> al resultado
                        Fin si
                Fin para
        Fin para
Fin para
```

- Costo del Input (CI) : por cada segmento de B-2 bloques de R, se recorre S entero

$$CI = B_R + B_S * \lceil B_R / (B-2) \rceil$$

Algoritmos para Juntas (⋈)

- Operación: **$R \bowtie_p S$**
- Algoritmo **Index Nested Loop Join**
- Precondición:
  - P es una expresión de equijoin
  - El archivo tiene un índice según una clave de búsqueda k
  - El predicado p coincide con la clave de búsqueda k (en el sentido de index matching)
- Descripción:

```
Para cada tupla r de R
        Para cada tupla s de S | r_i = s_j (obtenida según el índice de S)
                Si no hay condicion adicional o (hay y s la cumple )
                        Agregar <r,s> al resultado
                Fin Si
        Fin para
Fin para
```

- Costo del Input (CI) : por cada bloques de R, para cada tupla se busca en el índice las entradas coincidentes, y por cada una se busca la tuplas en el archivo S

$CI = B_R + T_R$ * (costo de acceder mediante el índice a las tuplas de S que cumplen la condición)

Algoritmos para Juntas (⋈)

- Operación: **$R \bowtie_p S$**
- Algoritmo **Sort merge Join**
- Precondición: se disponen de B bloques de memoria, (B>=3)
- Descripción:
  - En caso de ser necesario, se ordenan las tuplas de cada una de las relaciones según una clave coincidente con p
  - Luego de eso, se van recuperando bloques de ambas relaciones ordenadas, mediante un mecanismo de “apareo”
  - Las tuplas que cumplen la condición p, se agregan al resultado
- Costo del Input (CI) : se tiene el costo de ordenar ambas relaciones y, en el caso general, se recorre una vez cada archivo ordenado durante el apareo

$$CI = B_R + B_S + (\lceil \log_{B-1} \lceil B_R/B \rceil \rceil +1)*2B_R + (\lceil \log_{B-1} \lceil B_S/B \rceil \rceil +1)*2B_S$$

El costo del ordenamiento corresponde al algoritmo Sort Merge Join, muy común en BBDD

Costos del procesamiento

Las diferentes implementaciones tratan de aprovechar

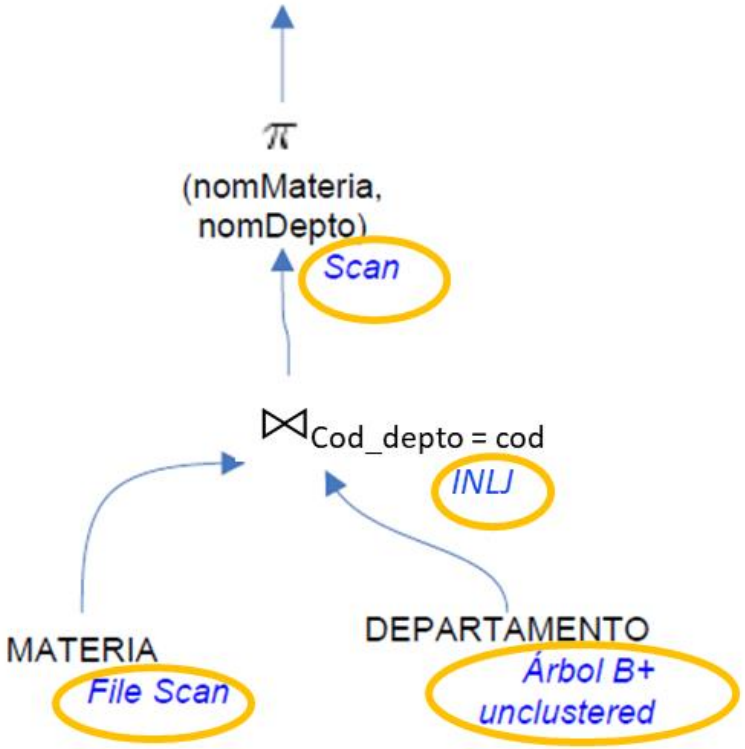
- Órdenes interesantes del input
- Existencia de índices “útiles” para la consulta
- Tamaños de los archivos
- Tamaños del buffer

Planes de Ejecución

Son **secuencias de operaciones físicas**.

El plan tiene que indicar el algoritmo concreto para resolver cada operación algebraica involucrada y el método de acceso concreto para acceder a los datos.

Costo de un Plan de Ejecución: es la Sumatoria de los costos de input y output de cada nodo del mismo.



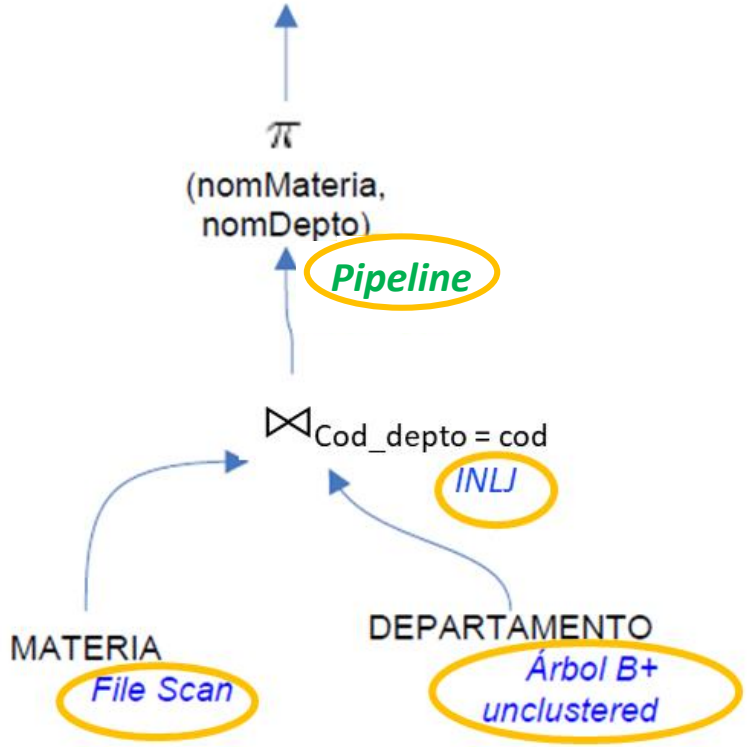
En este ejemplo, tenemos estos costos:

a) Costo del Input y Procesamiento del INLJ, accediendo a MATERIA mediante Filescan y a DEPARTAMENTO mediante un árbol B+ unclustered
b) Costo del output del INLJ
c) Costo del input y procesamiento de la proyección, accediendo al output previo mediante filescan
d) Costo del output de la proyección

Planes de Ejecución

- Un caso de interés es el tratamiento de los resultados intermedios entre nodos.
- Hay dos tratamientos:

- **Materialización**: los resultados intermedios son persistidos, lo cual involucra costo de escritura y de lectura (por el nodo siguiente)
- **Pipeline**: los resultados intermedios se le pasan "al vuelo" al nodo siguiente, el cual los va procesando directamente. Esto no involucra costos, ya que no hay escritura ni lectura.



- Un caso típico de pipeline es la selección o junta seguida de proyección.
- Nuestro ejemplo anterior podría modificarse para utilizar pipeline entre la junta y la proyección

- En este caso, los costos de los ítems b) y c) desaparecerían

Planes de Ejecución

- Órdenes interesantes:

Un resultado intermedio está en un orden interesante si:

- Alguna cláusula ORDER BY de un nivel superior del árbol coincide con ese orden
- Alguna cláusula GROUP BY de un nivel superior del árbol coincide con ese orden
- Alguna cláusula DISTINCT de un nivel superior está aplicada a atributos que coinciden con ese orden
- Algún JOIN de un nivel superior puede hacer uso de ese orden para disminuir sus costos

- Un resultado con un orden interesante puede ser de utilidad para minimizar costos globales aunque su elección no minimice el costo local (o hasta la operación de ese nodo)

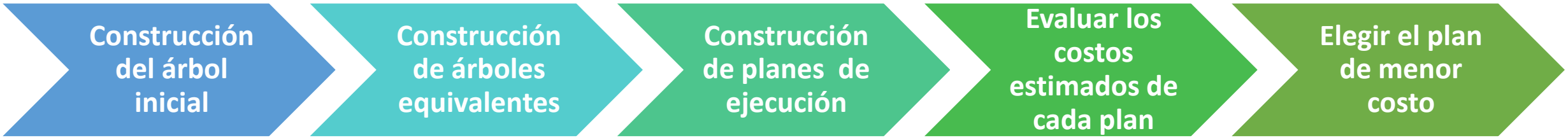
Proceso de optimización de consultas

- En este proceso el motor busca encontrar la mejor estrategia disponible para resolver una consulta
- El plan de ejecución resultante no necesariamente será “el” óptimo absoluto.
- Será el plan **más eficiente** que pueda encontrarse **en un tiempo razonable** a partir de la **información disponible** del esquema y del contenido de las relaciones involucradas.

- El proceso de optimización se basa en algunas heurísticas que aprovechan ciertas propiedades de las operaciones algebraicas

Proceso de optimización de consultas

- En este proceso el motor busca encontrar la mejor estrategia disponible para resolver una consulta
- El plan de ejecución resultante no necesariamente será “el” óptimo absoluto.
- Será el plan **más eficiente** que pueda encontrarse **en un tiempo razonable** a partir de la **información disponible** del esquema y del contenido de las relaciones involucradas.
- Los pasos estándar de este proceso son:



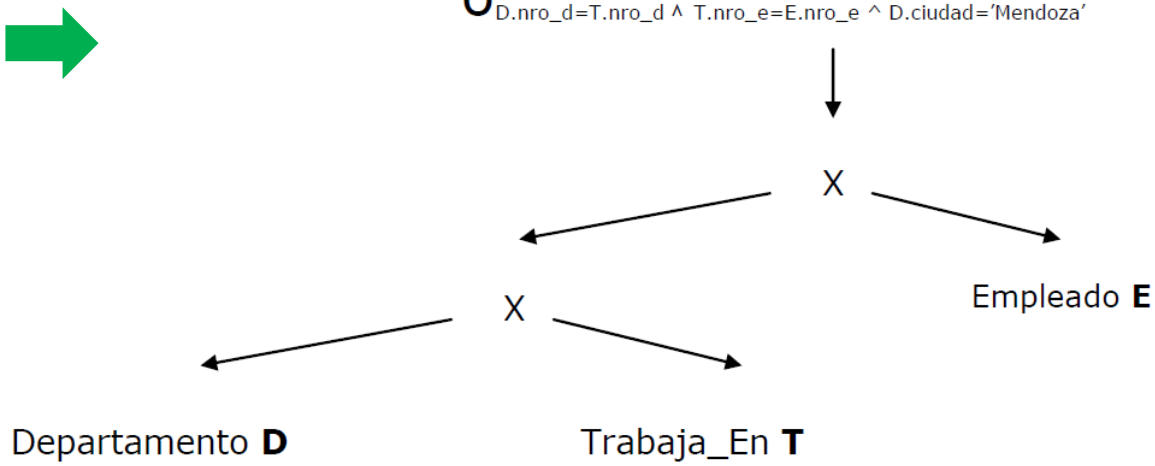
- Es un árbol canónico
- Se utilizan heurísticas
- Puede ser más de uno por cada árbol
- A partir de la información disponible

Este enfoque corre el riesgo de evaluar más de una vez las mismas operaciones en el contexto de diferentes planes de ejecución.
Existe una estrategia llamada Programación Dinámica, que trata de optimizar esto en base a una estrategia de varias pasadas.
No la veremos en la materia.

Proceso de Optimización de Consultas

- **Árbol Canónico**: es un árbol de consulta con las siguientes características:
  - El nodo raíz tiene la proyección de atributos de la consulta
  - El nodo hijo de la proyección tiene una selección con todas las condiciones necesarias tanto para selecciones individuales de relaciones como para juntas entre ellas
  - Como descendiente de la selección, hay un árbol sesgado a izquierda cuyos nodos son productos cartesianos que cubren todas las relaciones involucradas.

```
SELECT   nro-e, nombre-e
FROM     Departamento D, Trabaja-en T, Empleado E
WHERE    D.nro-d = T.nro-d  and
         T.nro-e  = E.nro-e  and
         D.ciudad = "Mendoza";
```



Proceso de Optimización de Consultas

- **Optimizaciones algebraicas**: buscan mejorar la performance independientemente de la organización física
- Propiedades

Cascada de σ

$\sigma_{C1 \text{ and } C2 \ldots\ldots \text{and } Cn}(R) \equiv \sigma_{C1}(\sigma_{C2}(\ldots\ldots\sigma_{Cn}(R)\ldots))$

Conmutatividad de σ

$\sigma_{C1}(\sigma_{C2}(R)) \equiv \sigma_{C2}(\sigma_{C1}(R))$

Cascada de π

$\pi_{list1}(\pi_{list2}(R)) \equiv \pi_{list1 \cap list2}(R)$

Conmutatividad de σ con respecto a π

$\pi_{A1, A2, \ldots\ldots An}(\sigma_{C}(R)) \equiv \sigma_{C}(\pi_{A1, A2, \ldots\ldots, An}(R))$

Si C referencia solamente a atributos dentro de $A_1 \ldots A_n$

Comutatividad del Producto Cartesiano (o Junta)

$R \times S \equiv S \times R$

Proceso de Optimización de Consultas

- **Optimizaciones algebraicas**: buscan mejorar la performance independientemente de la organización física
- Propiedades (cont)

Conmutatividad del σ con respecto al Producto Cartesiano (o Junta)

$\sigma_C (R \times S) \equiv (\sigma_{Cr}(R)) \times (\sigma_{Cs}(S))$ donde $C = C_r \cup C_s$

Conmutatividad de la π con respecto al Producto Cartesiano (o Junta)

$\pi_L (R \times S) \equiv (\pi_{L1}(R)) \times (\pi_{L2}(S))$ donde $L = L_1 \cup L_2$

Conmutatividad de operaciones de conjuntos ($\cup$ e $\cap$)

$R \theta S \equiv S \theta R$

Si $\theta = \{\cup$ e $\cap\}$

Asociatividad del Producto Cartesiano, Junta, $\cup$ e $\cap$

$(R \theta S) \theta T \equiv R \theta (S \theta T)$

Si $\theta = \{x, |X|, \cup$ e $\cap\}$

Proceso de Optimización de Consultas

- Algunas heurísticas aplicables al generar árboles de consulta equivalentes:

Considerar sólo árboles sesgados a izquierda	Descomponer las selecciones conjuntivas	Llevar las selecciones y proyecciones lo más cerca posible a las hojas	Evitar en lo posible los productos cartesianos
Realizar primero los joins más selectivos, y utilizar como outer (externas) las relaciones más selectivas	En cada nodo, retener los planes menos costosos, sin perder de vista órdenes interesantes	Tener en cuenta los índices y órdenes interesantes	Utilizar pipeline siempre que sea posible

- Veremos ahora algunos ejemplos en base a un modelo simplificado, como para afianzar conceptos

Ejemplos de optimización de consultas

- Sean estas tablas

```
CREATE TABLE division (
    id_division SMALLINT PRIMARY KEY,
    nombre VARCHAR(50)
);
```

```
CREATE TABLE empleado (
    legajo INTEGER PRIMARY KEY,
    apellido VARCHAR(50),
    nombre VARCHAR(50),
    anio_ingreso SMALLINT,
    id_division SMALLINT,
  FOREIGN KEY (id_division) REFERENCES division (id_division)
);
```

tal que

- T_{Emp}: 100.000
- T_{Div}: 50

Existe un índice árbol B+ por cada clave primaria

- Se tiene la siguiente consulta,

```
SELECT E.apellido, E.nombre, D.nombre division
FROM EMPLEADO E
    LEFT JOIN DIVISION D
        ON E.id_division = D.id_division
WHERE E.anio_ingreso>=2020
;
```

- se pide construir un plan de ejecución optimizado sabiendo que

Para pensarlo, lo veremos en detalle en la próxima clase

Bloques disponibles en memoria B: 50
Tamaño de cada bloque (en bytes) L_B: 2048
Aproximadamente el 20% de los empleados ingresaron desde 2020 (inclusive)

Bibliografía del tema

- *Elmasri, Navathe (2016) Fundamentals of Database Systems, 7th Edition. Pearson. Cap. 18 y 19*
- *García Molina H., Ullman J. & Widom J. (2009) Database Systems: The Complete Book, (2da. Ed.), Pearson, Cap. 15 y 16*
- *Silberschatz A., Korth H. & Sudarshan S.(2020) Database System Concepts (7ma. Ed.), Mc.Graw Hill, Cap. 15 y 16*
- *Ramakrishnan r. & Gehrke J. (2003) Database System Concepts (3ra. Ed.), Mc.Graw Hill, Cap. 12 a 15*
- *Apunte "Procesamiento y Optimización de Consultas" , materia Base de Datos, FCEyN UBA*

Dudas

Muchas gracias!