Capítulo 4

Procesamiento de consultas

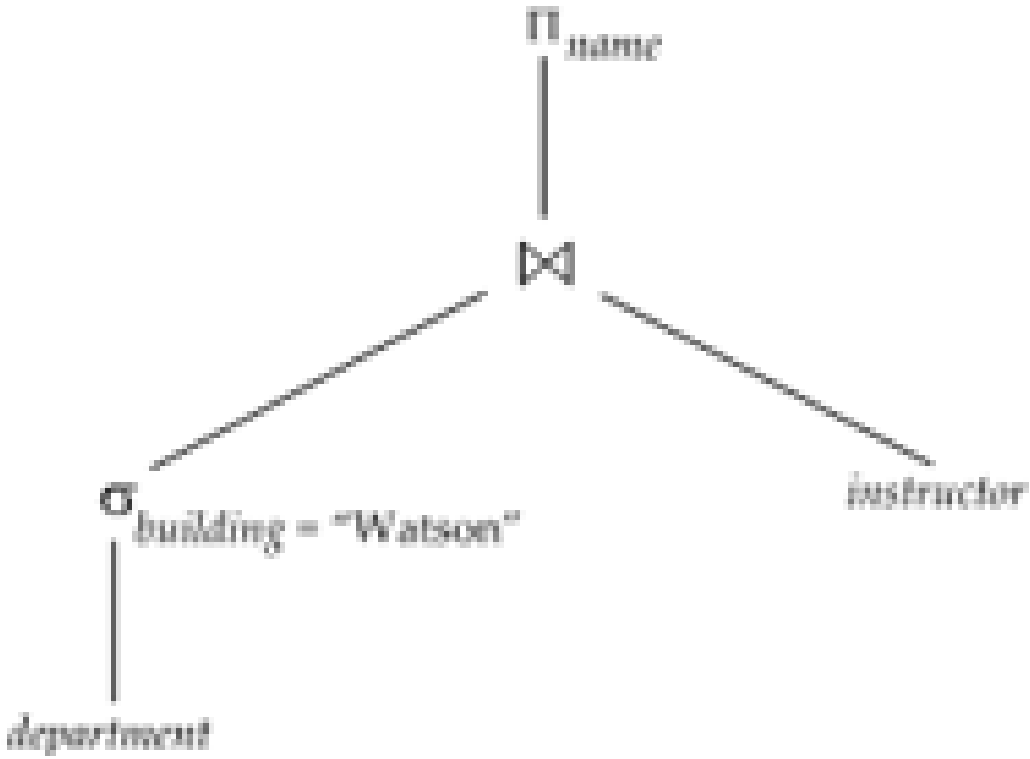
Visión general

- Para procesar una consulta, una sentencia de SQL se puede traducir a una expresión del álgebra de tablas.
- Luego se puede evaluar la consulta del álgebra de tablas.
- El **árbol binario de ejecución** de una consulta es el árbol binario de una expresión de consulta.
  - Los nodos hoja son tablas de la base de datos
  - Los nodos internos son operadores del álgebra de tablas
- Para una consulta del álgebra de tablas puedo definir varias expresiones equivalentes y cada una tiene su árbol binario de ejecución.

Árbol binario de ejecución

- El árbol binario de ejecución asociado a la consulta

$$\Pi_{name}(((\sigma_{building='watson'}(department)) \bowtie instructor)$$



Visión general

- Un **operador lógico** del álgebra de tablas considera el algoritmo ineficiente (usado para calcularlo) definido en el capítulo del álgebra de tablas.
- **Operadores físicos** son algoritmos específicos para operadores del álgebra de tablas.
  - En general tienden a ser más eficientes que los algoritmos lógicos porque hacen uso de informaciones adicionales, como índices, tamaños de búfer en memoria, técnicas avanzadas de algorítmica, etc.
  - Un operador del álgebra de tablas va a tener unos cuantos operadores físicos.
- La evaluación de un árbol binario de ejecución va a estar en términos de operadores físicos.

Visión general

- Una expresión de álgebra de tablas puede tener varias expresiones equivalentes.
- **Ejemplo**: $\sigma_{salary<75000}(\prod_{salary}(instructor))$ es equivalente a $\prod_{salary}(\sigma_{salary<75000}(instructor))$
- Dada una consulta *C* del álgebra de tablas un **plan de evaluación** consiste de un árbol binario de ejecución para una expresión equivalente a *C* y operadores físicos para ese árbol de ejecución.
- La **máquina de ejecución de consultas** toma el plan de evaluación de consulta, ejecuta ese plan y retorna las respuestas de la consulta.

Visión general

- Los diferentes planes de evaluación para una consulta dada pueden tener diferentes costos.
- Es responsabilidad del SGBD construir un plan de evaluación que minimiza el costo de evaluación de consultas.
- Una vez que un plan de evaluación es elegido, la consulta es evaluada con este plan.
- **Optimización de consultas**: entre todos aquellos planes de evaluación equivalentes encontrar aquel con menor costo.
  - El costo es estimado usando información estadística de la base de datos.
  - **P.ej:** número de registros en cada tabla, tamaño de los registros, etc.
- Con el fin de optimizar una consulta un optimizador de consultas debe conocer el costo de cada operación.

Visión general

- En este capítulo estudiamos cómo ejecutar un plan de evaluación; no como elegirlo.
- **Vamos a estudiar:**
  - Cómo medir costos de consultas
  - Algoritmos para evaluar operadores del álgebra de tablas.
  - Técnicas de procesamiento de consultas
- En el próximo capítulo estudiamos cómo optimizar consultas.
- La meta aquí es dada una expresión del álgebra de tablas, elegir bien los operadores físicos y dado el árbol binario de ejecución de esa expresión, poder estimar el **costo de procesamiento**.

Evaluación del árbol binario de ejecución

- El resultado de evaluar un operador de un nodo interno del árbol binario de ejecución que no es la raíz del árbol se llama **resultado intermedio**.
- Para la evaluación del árbol binario de ejecución hay dos enfoques:
  - **Materialización**: los resultados intermedios se guardan en disco en tablas temporales a las cuales tiene acceso el sistema gestor de BD (SGBD).
    - No hay índices sobre este tipo de tablas.
  - **Encausamiento**: a medida que se van generando los resultados intermedios se van pasando al siguiente operador.
    - Los resultados intermedios no se guardan en disco.
- **Comparando ambos enfoques**: Materialización usa mucho menos memoria y bastante más espacio en disco y encausamiento usa mucho menos espacio en disco, y bastante más memoria.

Medidas de costo de consulta

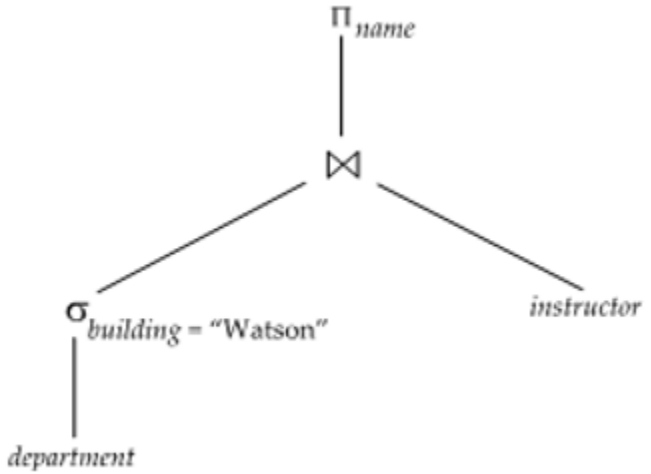
- Se contará el número de **transferencia de bloques** de disco.
  - Por simplicidad de las cuentas vamos a asumir que todas las transferencias de bloques tienen el mismo costo.
  - No distinguiremos entre las transferencias de bloques de lectura y las de escritura (a pesar que se demora más en escribir un bloque que leerlo de disco).
- Además contaremos la cantidad de **accesos a bloques**: tiempo que le lleva a la cabeza lectora posicionare en un bloque deseado.
- El tiempo de acceso a bloque suele ser de alrededor de 4 mseg y el tiempo de transferencia de bloques suele ser de 0,1 mseg, asumiendo un tamaño de bloque de 4 KiB y una tasa de transferencia de 40 MB por segundo.

Medidas de costo de consulta

- Los costos de todos los algoritmos físicos dependen del **tamaño del búfer** en memoria principal.
  - En el mejor caso todos los datos pueden ser leídos en búfer y el disco no necesita ser accedido de nuevo.
  - En el peor caso asumimos que el búfer puede sostener solo unos pocos bloques de datos – aproximadamente un bloque por tabla.
- Cuando presentamos estimaciones de costos, asumimos generalmente el peor caso.
- Vamos a ignorar los costos de CPU por simplicidad.

Materialización

- **Ejemplo**: Supongamos que tenemos el árbol binario de ejecución y usamos materialización:



1. Computamos y almacenamos en disco primero $\sigma_{\text{building = 'Watson'}}$ (department)
2. Computamos y almacenamos en disco la reunión natural del resultado intermedio anterior con *instructor*.
3. Computamos la proyección según *nombre*.

Materialización

- Con **materialización**:
  - Cambiar nodos lógicos por físicos en el árbol binario de ejecución.
  - Si hay más de un operador físico posible, elegir el menos costoso.
  - Evaluamos un operador físico por vez comenzando en el nivel más bajo.
  - Usamos resultados intermedios en tablas temporales para evaluar los operadores físicos del siguiente nivel.

Materialización

- **Problema**: Hay que estimar el tamaño de los resultados intermedios en cantidad de bloques a escribir a disco para los operadores de selección y reunión (selectiva y natural).
- **Solución**:
  1. Usar una función de probabilidad llamada **factor de selectividad** para calcular la cantidad de registros del resultado intermedio.
  2. A partir de cantidad de registros, calcular la cantidad de bloques del resultado intermedio.

Materialización

- Si el operador de selección o reunión usa predicado P, y el input del operador es i, denotamos al factor de selectividad mediante: fs(P, i).
- **Para selección:**

  cantidad de registros resultado intermedio = |r| * fs(P, r)
- **Para reunión:**
  - Cantidad de registros resultado intermedio = |r| * |s| * fs(P, r, s)
- Una forma de calcular el factor de selectividad es asumir uniformidad e independencia:
  - **Uniformidad**: todos los valores de un atributo son igualmente probables
  - **Independencia**: condiciones sobre diferentes atributos son independientes

Materialización

- **Ejemplo**: atributo sexo de persona: todos los valores de sexo son igualmente probables (la proporción de hombres es igual a la proporción de mujeres).
  - fs(sexo = ‘F’, persona) = 1/2
- **Ejemplo**: Atributos sexo y edad en persona, P = edad = 40 y sexo = ‘F’. La edad es independiente del sexo.
  - fs(edad = 40 and sexo = ‘F’, persona) = fs(edad = 40, persona) * fs(sexo = ‘F’, persona)
- Pero no es obligatorio usar uniformidad o independencia para calcular el factor de selectividad.

Materialización

- **Problema**: ¿Cómo calcular el número de bloques si tengo en el resultado N registros de tamaño R cada uno y B es el tamaño del bloque?
- **Solución**:

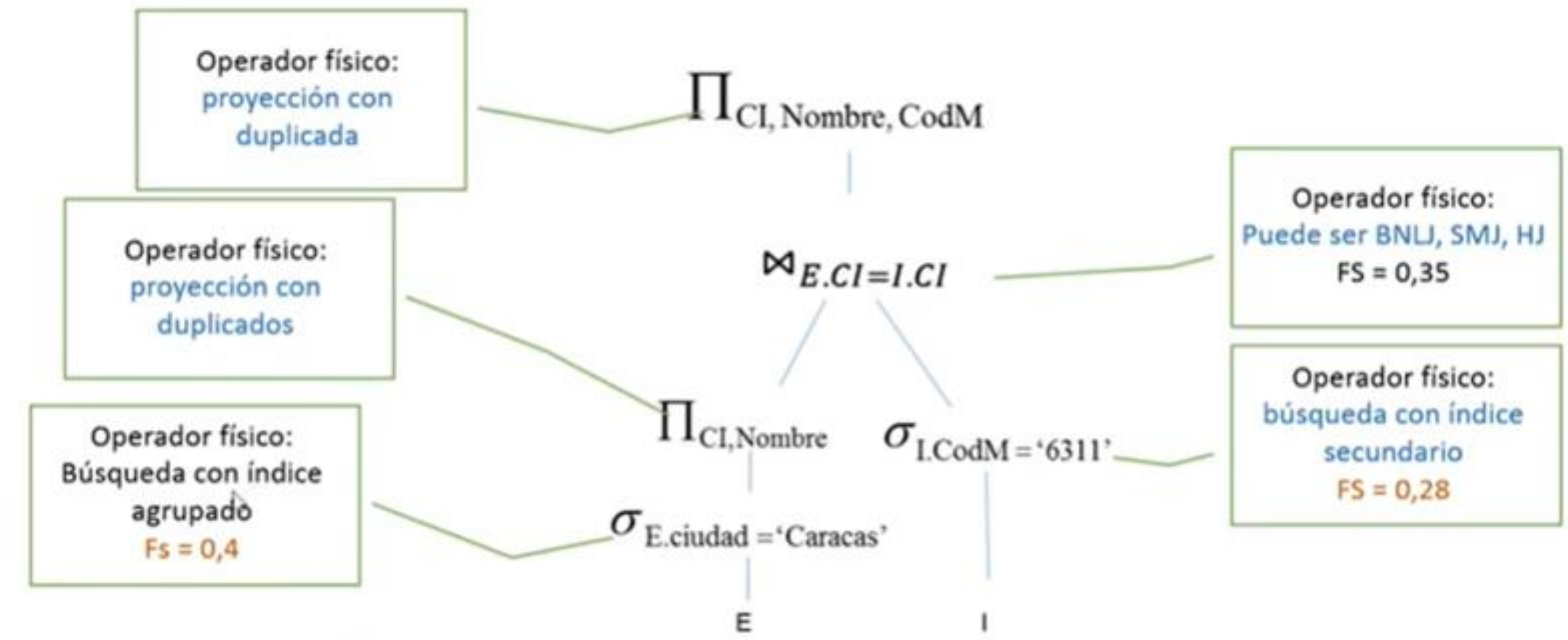
  NumBloques = ⌈(N x R) / B⌉ .

Materialización

- Ahora vemos cómo procesar y estimar el costo de una consulta con materialización.
- **Fase 1: decidir el plan de ejecución**
  - Armar el árbol binario de ejecución
  - Calcular el factor de selectividad para selecciones y reuniones (selectivas y naturales).
  - Decidir operadores físicos.
    - Solo se usan índices si la tabla de la BD lo amerita.

Materialización

- **Ejemplo de fase 1:**

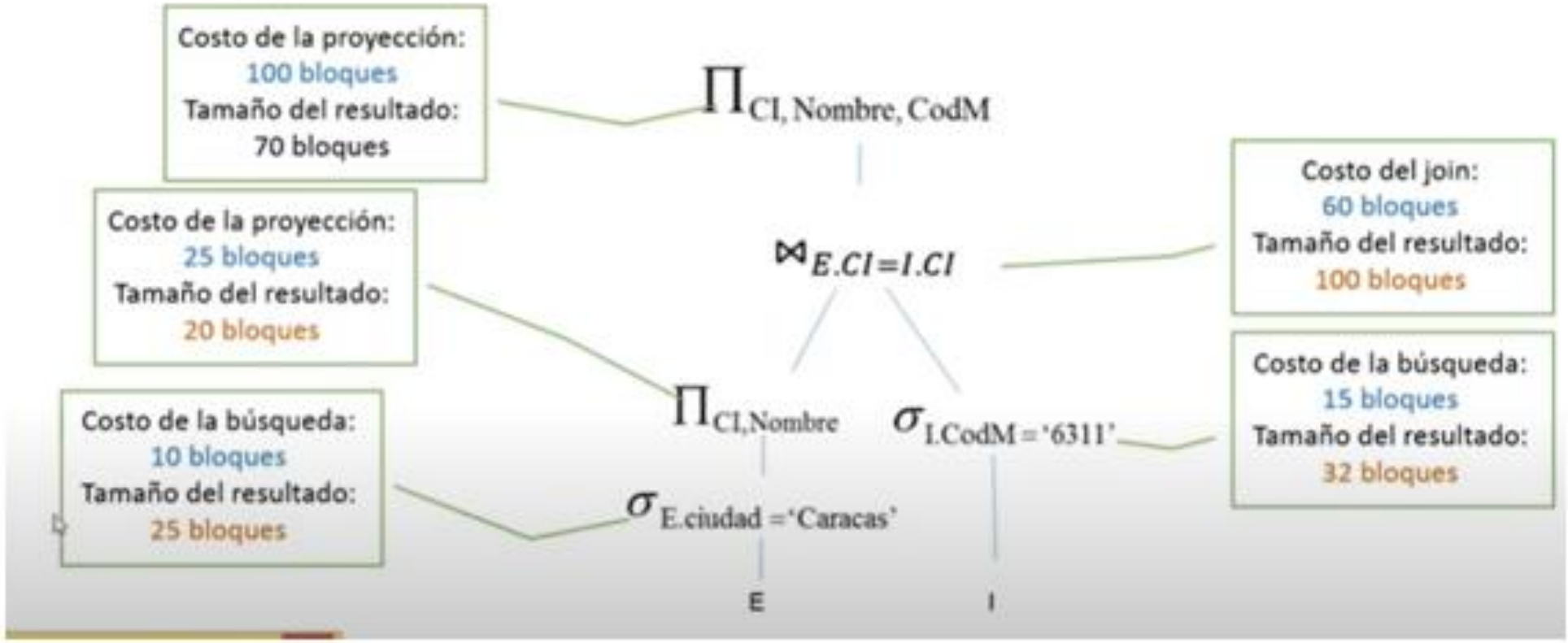


Materialización

- **Fase 2: estimar el costo de ejecutar el plan de evaluación**
  1. Calcular el tamaño en bloques de las tablas de la BD
  2. Calcular el tamaño de los resultados intermedios en bloques
  3. Calcular el costo de los operadores físicos
  4. Sumar los costos totales
- Aplicar el método de dos fases.

Materialización

- **Ejemplo de Fase 2:**



Materialización

- Juntando todo:

  - Materialización
    - Es la más usada por disponer de mayor cantidad de espacio en disco
    - $Costo\ total = \sum costo(operaciones) + \sum costo(materialización)$
    - Costo total = (10+20+15+60+100) + (25+20+32+100)
    - Costo total = 382 (accesos a disco)
    - Esto se multiplica por la velocidad de transferencia y se tiene el tiempo

Materialización

- Para poder aplicar materialización en detalle nos falta:
  - Reglas para el cálculo del factor de selectividad.
  - Operadores físicos con sus costos.
- A continuación vamos a hacer un ejemplo detallado de la aplicación de materialización.

Caso de estudio

- Sea la consulta: $\Pi_{id}\ (\sigma_{edad > 40}\ (persona))$, donde:
  - Persona ocupa 20 bloques de 10 registros cada uno.
  - Las personas tienen edades de 1 a 100
  - Asumir que entran 50 id por bloque.
  - Entran 20 edades por nodo en el índice primario en edad.

Árbol binario de ejecución

Π_{id}

|

σ(edad > 40)

|

persona

Factor de selectividad

- Usamos la propiedad:
- $fs(A \geq c, r) = (\max(A, r) - c) / (\max(A, r) - \min(A, r))$
- Así obtenemos:

$fs(edad > 40, persona)$

$= \max(edad, persona) - 41 / (\max(edad, persona) - \min(edad, persona))$

$= (100 - 41) / (100 - 1) = 59/99 = 0{,}595$

Operadores físicos

- **Proyección**:
  - Requiere recorrer todos los registros y realizar una proyección en cada uno.
  - Se recorren todos los bloques de la tabla.
  - Estimación de costo = b_r transferencias de bloques + 1 acceso a bloque
    - b_r denota el número de bloques conteniendo registros de la tabla *r*
- **Selección:**
  - La tabla está ordenada en *A*.
  - para $\sigma_{A \geq V}(r)$ usar el índice para encontrar el primer registro $\geq v$ y escanar la tabla secuencialmente desde allí.
    - Costo: hi + b transferencias de bloques, hi accesos de bloques.
    - b el número de bloques conteniendo registros con A $\geq v$.

Tamaño en bloques de las tablas

- Según el enunciado persona tiene 20 bloques.
- Cada bloque tiene 10 registros.

Tamaño de los resultados intermedios

- $r = \sigma_{edad > 40}(persona)$
- $|r| = |persona| * fs(edad > 40, persona) = 200 * 0{,}595 = 119$
- $B_r = 119 / 10 = 12$ bloques

Costo de los operadores físicos

- **Proyección**: costo es cantidad de transferencia de bloques de resultado intermedio: 12 transferencias de bloques.
- **Selección**: costo hi + b transferencias de bloques.
  - b es el número de bloques contiendo registros con edad > 40. Vimos que b = 12.
  - hi altura del árbol B+
  - Tenemos 100 edades y asumimos que entran 20 entradas por nodo del árbol B+. Entonces:
  - hi = $\lceil \log_{\lceil n/2 \rceil}(K) \rceil = \lceil \log_{\lceil 20/2 \rceil}(100) \rceil = \lceil \log_{10}(100) \rceil = 2$
  - Costo = 12 + 2 = 14 transferencias de disco.

Sumar los costos totales

- Costo total = costo de operaciones + costo de materialización
- = (12 + 14) + 12 = 24 + 14 = 38 transferencias de bloque.

Tamaño del resultado final

- La proyección se aplica a 119 registros
- La proyección me da 119 id
- Se asumió que entran 50 id por bloque
- Bloques de resultado final = Techo(119/50) = 3

Encauzamiento

- Materialización produce archivos temporales para los resultados intermedios.
  - Esto implica costo por transferencias de bloques hacia y desde el disco y de almacenamiento en disco.
- **Problema**: ¿Cómo mejorar la eficiencia de la evaluación de consultas reduciendo el número de archivos temporales que son producidos?
- **Solución (encauzamiento)**: se pueden combinar varias operaciones del AT en una tubería de operaciones en la cual los resultados de una operación son pasados para la siguiente operación de la tubería.
- **Ejemplo**: Sea $\Pi_{A, B}(r \bowtie s)$.
  - Con encauzamiento cuando la operación de reunión genera una tupla de su resultado, pasa esta tupla a la operación proyección para procesamiento y se crea resultado final directamente.

Encauzamiento

- **Beneficios de encauzamiento**:
  1. Elimina el costo de leer y escribir tablas temporales, reduciendo así el costo de evaluación de consultas.
  2. Puede comenzar generando resultados rápidamente, si el operador raíz de un plan de evaluación de consulta es combinado en una tubería con sus inputs.
     - Esto es útil si los resultados son mostrados al usuario a medida que son generados.
- Los requisitos de memoria son bajos porque los resultados de una operación no son almacenamos por mucho tiempo.
- Los inputs de operaciones no están disponibles todos a la vez para procesamiento.

Encauzamiento

- **Implementación del encauzamiento:**
  - Cada operación en la tubería puede implementarse como un **iterador** que provee las siguientes funciones en su interfaz: abrir(), siguiente(), y cerrar().
  - Un iterador produce la salida de una tupla por vez.
  - Varios iteradores pueden estar activos en un tiempo dado, pasando resultados hacia arriba en el árbol de ejecución.
  - El plan de consulta puede ejecutarse invocando los iteradores en un cierto orden.
  - Cuando describimos iteradores y sus métodos asumimos que hay una clase para cada operador físico implementado como un iterador y la clase define abrir(), siguiente() y cerrar() en las instancias de la clase.

Encauzamiento: interfaz de un iterador

- **Abrir():**
  - Inicializa el iterador alojando búferes para su entrada y salida e inicializando todas las estructuras de datos necesarias para el operador.
  - Además llama abrir() para todos los argumentos de la operación.

Encauzamiento: interfaz de un iterador

- **Siguiente():**
  - cada llamada a **siguiente()** retorna la próxima tupla de salida de la operación;
  - además ajusta las estructuras de datos para permitir que tuplas subsiguientes sean obtenidas.
  - Siguiente() ejecuta el código específico de la operación siendo realizada en los input.
  - Además llama siguiente() una o más veces en sus argumentos.
  - En **siguiente()** el **estado del iterador** es actualizado para mantener la pista de la cantidad de input procesado.
  - Cuando no se pueden retornar más tuplas se retorna un valor especial: **NotFound**.

Encauzamiento

- **Cerrar():**
  - termina la iteración luego que todas las tuplas que pueden ser generadas han sido generadas, o el número requerido de tuplas ha sido retornado.
  - Se llama cerrar() en todos los argumentos del operador.
- La implementación de una operación llama abrir() y siguiente() en sus inputs para obtener las tuplas de los input cuando son necesitadas.

Encauzamiento

- **Problema**: si tenemos un iterador para un operador que opera sobre tablas de la BD, ¿cómo puede hacer para acceder a tuplas de esas tablas? (recordar que siguiente llama siguiente de sus argumentos)
- **Solución**: definir iteradores que escanean las tablas de la BD y retornan tuplas de ellas.

Encauzamiento

- **Ejemplo**: Iterador para escaneo de tabla R

```
Open() {
    b := the first block of R;
    t := the first tuple of block b;
}

GetNext() {
    IF (t is past the last tuple on block b) {
        increment b to the next block;
        IF (there is no next block)
            RETURN NotFound;
        ELSE /* b is a new block */
            t := first tuple on block b;
    }  /* now we are ready to return t and increment */
    oldt := t;
    increment t to the next tuple of b;
    RETURN oldt;
}

Close() {
}
```

Encauzamiento

- **Ejemplo**: **escaneo ordenado** de tabla R. Se leen tuplas de R y se las retorna en forma ordenada.
  - No se puede retornar la primera tupla hasta que se ha examinado cada tupla de R. Asumimos que R entra en memoria principal.
  - Abrir() lee R en memoria principal y ordena las tuplas de R.
  - Siguiente() retorna cada tupla en el orden de ordenación.

Encauzamiento

- **Ejemplo**: concatenación R ++ S. Sean *R* y *S* iteradores que producen tuplas de R y S. *R* y *S* son escaneos de tablas.

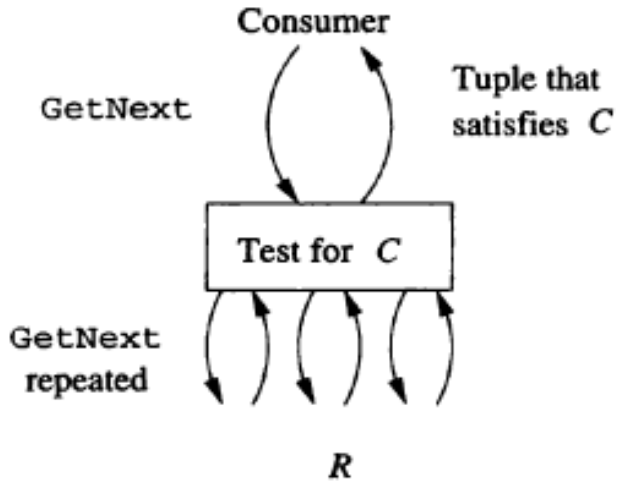
```
Open() {
    R.Open();
    CurRel := R;
}

GetNext() {
    IF (CurRel = R) {
        t := R.GetNext();
        IF (t <> NotFound) /* R is not exhausted */
            RETURN t;
        ELSE /* R is exhausted */ {
            S.Open();
            CurRel := S;
        }
    }
    /* here, we must read from S */
    RETURN S.GetNext();
    /* notice that if S is exhausted, S.GetNext()
       will return NotFound, which is the correct
       action for our GetNext as well */
}

Close() {
    R.Close();
    S.Close();
}
```

Encauzamiento

- **Ejemplo**: **iterador implementando selección usando búsqueda lineal:**
  - **Abrir()** comienza el escaneo del archivo R y el estado del iterador guarda el punto en el cual el archivo ha sido escaneado.
  - Cuando se llama **siguiente()** el escaneo del archivo continua después del punto previo.
  - Puede ser necesario llamar siguiente() varias veces en R hasta que una tupla que satisface la condición de la selección es encontrada.
  - Cuando la próxima tupla satisfaciendo la selección es encontrada por escanear el archivo R, la tupla es retornada luego de almacenar el punto donde fue encontrada en el estado del iterador.



Esta figura muestra como funciona el iterador de selección (rectángulo *test for C*) usando iterador para R.

Encauzamiento

- **Ejemplo: Iterador para proyección:**
  - Supongamos hay tabla de origen R con iterador.
  - Siguiente() del iterador de proyección llama siguiente() en iterador de R y proyecta la tupla obtenida apropiadamente y retorna el resultado al consumidor.
- **Ejemplo: Iterador implementando reunión por combinación (merge-sort join):**
  - Asumo como inputs tablas R y S con iteradores de escaneo ordenado.
  - **Abrir()** va a llamar R.abrir() y S.abrir() o sea, que si estos no están ordenados, entonces van a ordenarse.
  - Cuando se llama **siguiente()** sobre la reunión, va a retornar el siguiente par de tuplas que cazan.
    - La **información de estado** va a consistir de hasta donde cada tabla de input ha sido escaneada.