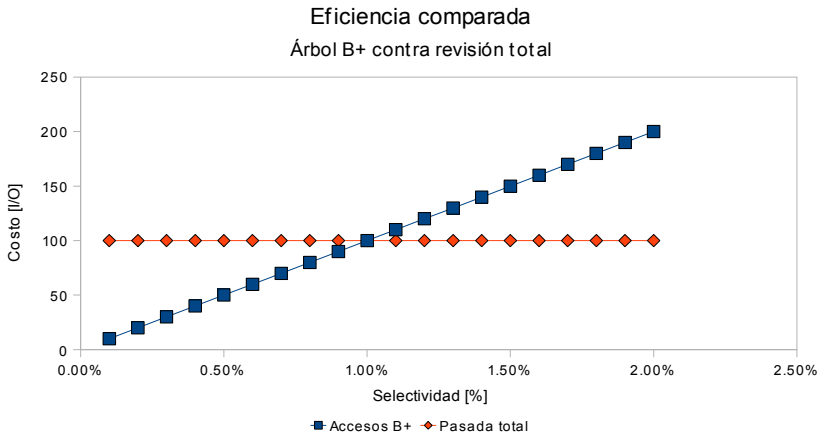
Auxiliar de Optimización

Problema 1

¿Cuándo es útil usar índices? Estudie el caso de búsqueda por intervalos o rangos (*range search*) por un atributo cuando hay 10000 filas (tuplas) en una tabla (relación), cada página guarda 100 filas, usa un árbol B+ de profundidad 4 y tiene un archivo desordenado. Base su respuesta en la selectividad de la búsqueda.

Resp. Lo primero es notar que cada fila recuperada con un índice cuesta un acceso a disco ([I/O]) por sobre los usados por el índice, ya que el archivo está desordenado.
Segundo, obviaremos el costo de recorrer el árbol B+ ya que, si bien la profundidad 4 significa usar al menos 4 [I/O] para usar el árbol, no usaremos muchas más en una búsqueda por intervalos.
Tercero, en una revisión total del archivo, el costo no sale de leer el archivo fila a fila, sino página a página. Como hay 10000 filas y 100 filas/página, sólo hay 100 páginas en el archivo. Luego, el costo de la revisión total es 100 [I/O], independiente de la selectividad.
Con todo lo anterior, notamos que el índice B+ es buena opción sólo cuando elegimos menos del 1% de las filas. La comparativa general, la dibujamos en el siguiente gráfico:



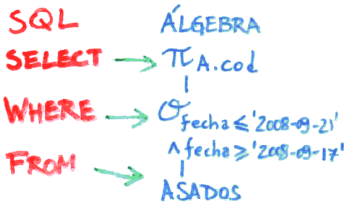
Problema 2

La primera consulta de la auxiliar pasada fue:

```
SELECT lugar
FROM ASADO
WHERE '2008-09-17'<=fecha AND fecha<='2008-09-21' ;
```

Nuestro problema consiste en optimizarla.

Resp. La optimización sigue dos etapas: una optimización heurística, que juega con equivalencias de álgebra relacional, y la optimización algorítmica, que trata de cómo se evaluarán los operadores de álgebra relacional. (Seguimos el estilo del antiguo optimizador *System R*.)
Traducimos SQL a álgebra, notando que: SELECT es una proyección, FROM es una cruza de relaciones y WHERE es una selección. Luego escribimos la consulta SQL en álgebra relacional, sin que quede mucho por optimizar.



Segundo, elegimos los operadores óptimos, según la información disponible (catálogo). Vemos si *fecha* tiene índice o no, y estimamos la selectividad de la consulta. Tal como en el problema anterior:

- Hacemos búsqueda por rangos si tenemos un B+ en *fecha*, y la selectividad es suficientemente pequeña.
- Hacemos búsqueda total en otro caso.

Si el archivo estuviera ordenado por fecha, casi siempre convendría usar un índice. ¿Por qué?

Problema 3

La segunda consulta de la auxiliar pasada fue:

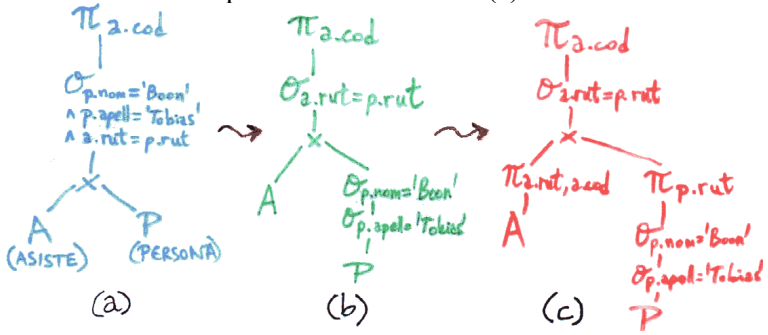
```
SELECT A.cod
FROM ASISTE A, PERSONA P
WHERE A.rut=P.rut AND P.nombre="Boon" AND P.apellido="Tobias" ;
```

Optimizarla en álgebra relacional y estudiar los algoritmos de join natural.

Resp. Primero, optimizamos en álgebra relacional. Notamos las equivalencias:

- $\sigma_{a \wedge b}(X) = \sigma_a(\sigma_b(X))$
- $\pi_{a \cap b}(X) = \pi_{a \cap b}(\pi_a(X))$

que obviamente permiten conmutatividad entre *a* y *b*, ya que $\wedge$ y $\cap$ son conmutativos.
Luego, optimizamos la consulta en álgebra relacional, traduciendo de SQL a álgebra (a), llevando los σ a las hojas (b) y proyectando antes de los productos cartesianos (c).



Respecto al join natural, hay bastantes maneras de realizarlo. Recordemos los tres principios de diseño de algoritmos usados en bases de datos:

- Iteración: fuerza bruta, recorrer todo a la antigua. Como diríamos en PHP, puros *foreach*.
- Ordenación: fuerza mental, preparar los datos en el vuelo y generar combinaciones con ingenio. Se llama ordenación porque al ordenamos podemos realizar varias operaciones de manera más eficiente. Pero el principio no se limita a ordenar.
- Indización: uso de índices preexistentes (hacerlos en el vuelo es seguir el principio de ordenación). La estrategia aquí es tener todo preparado para usar los algoritmos rápidamente.

Siguiendo estos principios, estudiaremos algunos algoritmos posibles para join natural.

NOTA: En lo que sigue, presentaremos algoritmos en pseudocódigo para realizar join natural. Los costos quedan como ejercicio y se pueden calcular de la siguiente manera:

- Cada bucle sobre elementos en disco, tiene costo en [I/O].
- Cada bucle sobre elementos en RAM, no tiene costo (no se compara a un [I/O]).
- Ordenar es proporcional a $n \log n$, pero con una constante considerable.

Para el estudiante inspirado, si desea hacer un evaluador de consultas casero, como SQLite tal vez, o que simplemente desea estudiar los costos de las consultas a través de simulaciones, los siguientes pseudocódigos pueden ser útiles como guía para la escritura de los programas reales.

Iteración.
La familia de los nested-loop join o reuniones con bucle anidado.

Versión simple, por tupla

```
Resp = Ø ;
foreach(tupla ta de A) {
        foreach(tupla tb de B) {
                if (ta[rut]=tb[rut])  Resp = Resp ∪ ta.tb ;
        }
}
```

Versión por páginas

```
Resp = Ø ;
foreach(pág pa de A) {
        foreach(pág pb de B) {
                foreach(tupla ta de pa) {
                        foreach(tupla tb de pb) {
                                if (ta[rut]=tb[rut])  Resp = Resp ∪ ta.tb ;
                        }
                }
        }
}
```

Ordenación.
Ahora interesa el ordenar o crear estructuras en el vuelo.

Ordenar, luego revisar

```
Resp = Ø ;
Ordenar(A) ;
Ordenar(B) ;
ia = 0;
ib = 0;
while( ia ≤ |A| ∧ ib ≤ |B| ) {        // mientras los índices ia e ib no sobrepasan los tamaños de las tablas
        ta = A.index(ia) ;            // ta es la ia-ésima fila de A, ahora ordenada
        tb = B.index(ib) ;
        if (ta[rut] = tb[rut]) Resp = Resp ∪ ta.tb ;
        if (ta[rut] < tb[rut]) ta++;
        if (ta[rut] > tb[rut]) tb++;
}
```

Hash-join: crear índice de hashing en el vuelo

```
Resp = Ø ;
Ha = Hash(A) ;          // creamos dos tablas de hashing de igual tamaño: Ha y Hb
Hb = Hash(B) ;
ih = 0;
while( ih ≤ |Ha| ) {
        foreach(tupla ta en Ha.index(ih)) {
                foreach(tupla tb en Hb.index((ih)) {
                        if (ta[rut] = tb[rut]) Resp = Resp ∪ ta.tb ;
                }
        }
}
```

Indización.

Indización o indexación. Aprovechamos las estructuras existentes.

Hash-join, con hashing existente

```
Resp = Ø ;
Let Ha, Hb ;            // dos tablas de hashing de igual tamaño, de A y B
ih = 0;
while( ih ≤ |Ha| ) {
        foreach(tupla ta en Ha.index(ih)) {
                foreach(tupla tb en Hb.index((ih)) {
                        if (ta[rut] = tb[rut]) Resp = Resp ∪ ta.tb ;
                }
        }
}
```

Preg. ¿Y si son de diferente tamaño? ¿Y si sólo una está indizada?

B+ join (sólo en palabras)

Tomamos las hojas de los árboles B+, y las recorremos tal como ordenación. Accedemos a las tablas sólo en caso de ser necesario.

Ejercicios propuestos poco triviales:

- Mezclar los principios de diseño de algoritmos, i.e. ordenación con fuerza bruta, ordenación con indización, etc.
- Hacer reuniones naturales de tres tablas. Esto es bien típico; la entidad A se relaciona con B a través de R, y nosotros queremos $A * R * B$. Diseñar *buenos* algoritmos para este caso (o sea, obviar fuerza bruta).
- Buscar datos multidimensionales. Por ejemplo, hacer join natural de información geográfica, o sea, coordenadas (x, y). Hint: puede usar R-Tree.