Resumen de las clases 6 y 9 de junio

Avance general de la materia

En estas dos clases se vio:

– Un repaso, con ejercicios, de la materia de índices.

– Una introducción a la arquitectura de procesamiento de consultas de una base de datos.

– Un acercamiento al problema de programación dinámica y de porqué su solución con heurísticas.

– La metodología de optimización heurística de árboles de consulta.

– Una breve introducción al diseño de algoritmos en el tema de evaluación de operadores relacionales.

Ejercicios de índices y almacenamiento

1. Tiene un archivo agrupado (clustered) con 4 clústers de 5 elementos de capacidad. Cada clúster tiene sólo 2 elementos. En el archivo, se tienen los números del 1 al N.

   (a) Haga un diagrama del archivo. ¿Cuánto vale N?

   (b) ¿Cuánto tarda en buscar el 3?

   (c) Borre el 4. ¿Cuánto tarda?

   (d) Borre el 3 y reordene. ¿Cuánto tarda?

   (e) Agregue: 3, 4, 5, 6, 9, 10, 10, 0.

2. ¿Cuándo el archivo agrupado es menos eficiente que el archivo ordenado?

3. ¿Por qué hashing es malo con rangos?

4. Sea $g : \text{palabras} \to N$, un ordenador lexicográfico de 0 a infinito. Analice la calidad de las siguientes funciones como funciones de hashing:

   (a) $g(w)$

   (b) $\lfloor 10/g(w) \rfloor$

   (c) $\lceil \log g(w) \rceil$

   (d) $\lfloor \cos g(w) \rfloor$

   (e) $g(w) \bmod m$

   (f) $g(w) * m' \bmod m$ (m' primo relativo a m)

5. Construya un árbol B+, cuyos nodos aceptan 3 claves y 4 punteros, con los siguientes valores: 1, 2, 3, 4, 6, 50, 13, 20, 21, 20, 29, 28, 27, 40.

6. Construya una tabla de hashing cuya función de hashing sea $N \bmod 6$ con los valores: 1, 2, 3, 4, 6, 50, 13, 20, 21, 20, 29, 28, 27, 40.

7. Construya una tabla de hashing cuya funcion de hashing sea $3N \bmod 7$ con los siguientes valores: 1, 2, 3, 4, 6, 50, 13, 20, 21, 20, 29, 28, 27, 40.

8. Para las tres últimas estructuras, ¿cuál es el costo promedio de búsqueda por igualdad? ¿Cuál es la varianza?

La evaluación de consultas

Consideraciones generales

– A priori, la evaluación de consultas SQL puede ser costosa. Es interesante hacerla eficiente.

– La solución óptima al problema de evaluación consiste en un problema de programación dinámica, que busca minimizar el número de accesos a disco.

– La información para alimentar el problema de optimización proviene del catálogo, que tiene estadísticas diversas de los archivos creados en la base de datos.

– Las vías de solución son generadas a través de las propiedades del álgebra relacional (explotando su condición de álgebra).

– Evidentemente, es necesario leer el SQL, convertirlo a álgebra relacional, y optimizar la consulta considerando el álgebra relacional.

– Y además hay que considerar la evaluación inteligente de los operadores relacionales. Por ejemplo, usar los índices adecuados en los *join*, como en *range search* (si es usado).

– Arquitectura:

```
SQL --> [Analizador léxico]   //Verifica SQL
          +--> [Parser]   //Convierte a álgebra relacional
                 +--> [Optimizador]  //Optimiza!!! (en serio???)
                        +--> [Evaluador]  //Evalúa!!! (yaaaa!!!)
                               +--> Respuesta
```

El problema de la optimización

– La programación dinámica es un problema muy caro. Las propiedades del álgebra relacional permiten generar muchas expresiones distintas que son equivalentes en términos de resultados.

– El problema anterior se puede aligerar si sólo se consideran planes con LEFT-DEEP JOIN. La idea es reducir el número de opciones posibles. Sin embargo, el número de maneras diferentes de hacer de realizar las cruzas de tablas es del orden de $N!$ (en el número de tablas), o lo que es lo mismo, el número de estrategias de cruzas crece exponencialmente con el número de tablas. Sigue siendo bastante.

– Además, la información que provee el catálogo es limitada, por lo que la optimización no puede ser más que una estimación.

– Los problemas anteriores hacen que el problema de programación dinámica se reduzca en dos: en la optimización de un árbol de consulta, y en la mejor elección de los algoritmos de evaluación de los operadores relacionales. Ambos hacen el *plan de consulta*, que es entregado al evaluador.

Álgebra relacional: la optimización heurística

– Es importante conocer las propiedades algebraicas que presentan los operadores relacionales:

  – Selección: $\sigma_{c \wedge d} X = \sigma_c(\sigma_d X) = \sigma_d(\sigma_c X)$

  – Proyección: $\pi_A X = \pi_{AB} X = \pi_{ABC} X = \pi_{ABCD...} X$

  – Producto cartesiano: $(A \times B) \times C = A \times (B \times C)$

  – Prod. cartesiano, por fines *prácticos*: $A \times B = B \times A$

  – Sigma-producto: $A \times \sigma_{c(B)} B = \sigma_{c(B)}(A \times B)$ (ojo con la validez)

– Una expresión de álgebra relacional se puede ver como un árbol de evaluación. En este árbol, la composición representa una "rama" o vínculo, y los operadores representan los nodos.

– Para convertir de SQL a álgebra relacional es muy simple: el SELECT se convierte en un proyección, el FROM se convierte en productos cartesianos y el WHERE se convierte en una selección.

– Consultas más complejas (con SELECT DISTINCT, ORDER BY, GROUP BY, HAVING) agregan operaciones en la raíz del árbol. En demás, la estructura del árbol se mantiene.

– Heurística:

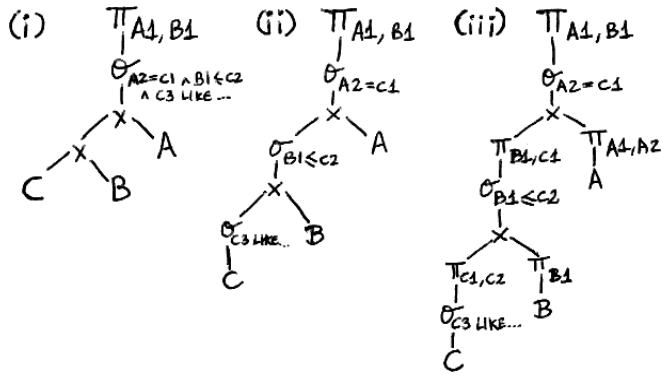
  – Llevar las selecciones lo más cerca posible de las hojas. IDEA: Minimizar el número de tuplas intermedias.

  – Proyectar frecuentemente, luego de las selecciones. IDEA: Minimizar el tamaño de las tuplas intermedias.

– Ojo que la heurística sólo considera planes con LEFT-DEEP.

– Ejemplo: para la consulta,

```
SELECT A.A1, B.B1 FROM A, B, C
WHERE A.A2=C.C1 AND B.B1<=C.C2 AND C.C3 LIKE 'COCODRILO%';
```



Evaluación de operadores relacionales

– No se vio mucho de esto, pero se trata de proponer y elegir algoritmos para la evaluación de operadores relacionales.

– Estos algoritmos deben ser adecuados a su contexto: tamaño de la entrada, presencia de índices, selectividad, etc.

– Los principios de diseño de los algoritmos se pueden resumir en tres: resolver por iteración (en general, fuerza bruta), ordenación (preparación de los datos durante la consulta) e indización (**o uso de índices existentes, creo que no dije esto en la auxiliar**).

– Ejemplo: JOIN con condición de igualdad en un atributo x, entre las tablas A y B.

  – Nested loop join/Join de bucle anidado: doble *for*, uno por las tuplas de A, otro por las tuplas de B, y en el resultado se presentan los pares de tuplas tales que a[x]=b[x]. Costo asintótico cuadrático. Mejoras: usar bloques en vez de tuplas para realizar menos I/Os.

  – Sort-merge join: primero se ordena A según x (costo: $n \log n$), luego B (otro costo: $n \log n$), y se realiza una pasada lineal entre A y B comparando valores iguales (costo: $n+n$). Costo asintótico $n \log n$.

  – Hash join: en vez de ordernar A y B, se construye una tabla de hashing para cada tabla. El costo de esto es lineal, idealmente. Y luego se realiza el join al comparar datos de las tablas de hashing, cosa lineal en condiciones ideales. El costo asintótico en condiciones ideales es lineal, pero en el peor caso es cuadrático.

  – Patrón de indización: ¿Cómo usar índices existentes para realizar los join? Nested loop join puede reducir su tiempo si se combina con árboles B+ o hashing. Sort-merge join se puede apurar si hay un árbol B+ en x, con lo que se evita tener que ordenar nuevamente. Hash join se puede apurar si la tabla de hashing ya existe.

Estimación de costos

– El optimizador, tomando algunos planes de consulta (árbol + algoritmos) y la información del catálogo, decide qué plan de consulta llevar a cabo eligiendo al plan que estima como el más rápido.

– Un plan es más rápido entre menos I/Os realiza. (I/O = acceso a disco.)

– La memoria primaria (RAM) no interesa. Sólo la secundaria (disco). Se supone que las tablas son tan grandes que no caben en la RAM, y el tiempo de acceso a memoria secundaria es significativamente mayor que el tiempo de acceso a la memoria primaria.

– Pregunta de diseño: ¿Cómo utilizar la RAM para acelerar los algoritmos de evaluación de operadores relacionales? La idea es que más RAM signifique menos tiempo.

– En un árbol de evaluación hay PIPELINING. Paso a paso, los resultados intermedios del árbol de evaluación son escritos en disco para dejar realizar otra operación independiente. Luego, la información es recuperada cuando se necesita. En resumen: **cada vez que se escribe algo, eso será leído más tarde**.

– La manera en que se lea la información indica la eficiencia de la evaluación. Por ejemplo, los I/Os de lectura de nested loop join son cuadráticos mientras que los I/Os de lectura de sort merge join subcuadráticos.

– Sin embargo, independiente de los costos de lectura y escritura por algoritmos, es necesario llevar la contabilidad de la selectividad de las operaciones, i.e. cuántas tuplas salen de cada operación, y qué tamaño tiene su esquema. Esto indicará cuánto escribirá un operador relacional como un join.

– Ojo que la salida de un join es independiente del algoritmo que se utiliza. El algoritmo incide en los costos de lectura de los PIPELINEs y en los costos de lectura y escritura intermedios.

– Por ejemplo, nested loop join no tiene costos intermedios, pero lee ineficientemente los PIPELINEs, En cambio, sort-merge join lee eficientemente los PIPELINEs, pero tiene [altos] costos intermedios.

– También hay que considerar que hay bloques de instrucciones que se ejecutan *simultáneamente*, o sea, en el vuelo. Por ejemplo, un trío cruza-selección-proyección se realiza como un solo join que escribe lo que la proyección le indica. Otro ejemplo, es que cuando se lee una tabla directamente de disco, la proyección sólo indica que atributos considerar. Reescribir la proyección de la tabla es redundante en muchos casos (aquí es necesario evaluar para decidir... véalo como ejercicio).

– Todos estos costos son considerados por el optimizador. Finalmente, la consulta que se crea más barata en tiempo será la que será ejecutada.

Ejercicios de optimización heurística

1. Tiene un árbol B+ sobre A.x, y sea "`select * from A where x>10`". A es una tabla desordenada. Si hay 10.000 tuplas y hay 20 por bloque, ¿cuántas tuplas debe devolver x>10 para usar el índice B+?

2. Realice el ejercicio anterior, pero comparando tablas ordenadas y tablas *agrupadas* (clustered). Proponga una jerarquía de métodos eficientes.

3. ¿Cuál es la ventaja del álgebra relacional en la optimización? ¿Cómo ocurre la optimización? ¿Qué se reduce?

4. Sea "`select p.id, count(*) from auto a, persona p where a.id=p.id group by p.id;`"

   (a) Obtenga el árbol de consulta.

   (b) Optimícelo heurísticamente.

   (c) ¿Qué índice usaría? ¿Qué impacto tiene?

   (d) Indique un algoritmo para evaluar el join.

   (e) Invente un algoritmo para evaluar *group by* y *count*.

   (f) Estime el costo de la consulta.

5. Sea "`select distinct p.apellido from personas p, universidad u, region r where r.nom='maule' and p.r_nom=r.nom and p.u_nom=u.nom;`"

   (a) Optimice: use dos árboles distintos y aplique las heurísticas a cada uno.

   (b) ¿Cómo evalúa *distinct*? (proponga un algoritmo)

   (c) Sin índices, elija algoritmos.

   (d) Estime costos. ¿Qué árbol es mejor?

6. Ha hecho dos consultas A y B. ¿Cómo evaluaría A *except* B? Proponga un par de algoritmos "eficientes". Note que no puede usar índices existentes.

7. Proponga y compare algoritmos para evaluar la división.

8. Proponga reglas heurísticas para reducir el número de árboles de consulta a considerar, i.e. que propongan órdenes para los productos cartesianos.