postdata9
www.wuolah.com/student/postdata9

tema2.pdf

Tema 2, Optimización

3° Administración de Bases de Datos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

TEMA 2:

OPTIMIZACIÓN DE CONSULTAS


CUNEF
Lidera tu futuro.
Define tu éxito.
POSTGRADO EN
DATA SCIENCE
PARA FINANZAS
SÚMATE AL ÉXITO
Excelencia, futuro, éxito.
www.cunef.edu

Los SGBD organizan la información dentro del disco. ¿Por qué habría que organizarla? Porque una vez almacenada la información, hay que obtenerla de la forma más eficiente posible. Una de las formas de acelerar el proceso de consulta es previendo en qué forma se van a consultar. La idea es intentar optimizar el SGBD, para conseguir que el sistema en su conjunto sea lo más eficiente posible.

Trabajaremos sobre ***lenguajes declarativos*** o de cuarta generación, como SQL, en los que se declara el objetivo de lo que queremos obtener, pero no le decimos al sistema cómo tiene que resolver el resultado, el sistema nos devuelve el resultado sin explicar cómo lo ha conseguido. (Nosotros estamos acostumbrados a los lenguajes procedimentales o de tercera generación en los que para resolver un problema hay que indicar cuáles son los pasos a seguir.) Estos lenguajes son de propósito muy específico, lenguajes para trabajar con problemas específicos como puede ser el tratamiento de la información, y son abstractos, ya que son cercanos al planteamiento teórico del problema.

Estos lenguajes presentan un problema ya que nosotros le expresamos la consulta que queremos resolver, y el sistema empezará a resolverlo como vea. Hay que tener en cuenta que el usuario puede no ser un experto o realizar consultas de forma muy ineficiente usando operadores que pueden llegar a ser ineficientes. Cualquier SGBD prevé estas situaciones, por ello, analiza antes de ejecutar la consulta. Mediante el análisis se consigue versiones optimizadas de la consulta, que van a concluir al mismo resultado pero con distinto tiempo para resolverlas. La diferencia entre estas versiones es el coeficiente, la constante p, que depende de la expresión de la consulta, de la carga del sistema. (Equiparando con la eficiencia de algoritmos, estaríamos hablando del orden de eficiencia. Dentro de un mismo orden, lo que importa es el coeficiente, la constante.)

¿Cómo se mide dicha eficiencia? Hay que saber cómo de buena o mala es dicha constante p, cuál de esas versiones es la más razonable para ejecutar. El sistema tiene una serie de heurísticas fijas que permiten evaluar distintas alternativas a una consulta, no busca todas las soluciones posibles ya que podría permanecer buscando soluciones sin dar una definitiva, sino que asume el factor riesgo y elige una. La constante se estima haciendo uso de una serie de estadísticas que hay dentro del sistema como el número de bloques, de registros por bloque. Estas estimaciones se hacen a priori, no sobre los propios datos. **El sistema planifica, decide y ejecuta.**

Para ello, vamos a ver cuál es la forma de obtener versiones alternativas a una consulta mediante lo que se llama la ***conversión de árboles de expresión***. Los árboles de expresión son representaciones únicas y minimizadas de una consulta, una vez tenemos estas versiones simplificadas (***plan lógico***), hay que ver cuál es mejor, por lo que hay que obtener la constante p multiplicadora para elegir la mejor versión de la solución que se supone que tarda menos tiempo en ejecutar. (Se supone porque puede que a la hora de planificar sea mejor, pero cuando se ejecute puede que tarde más que las otras versiones. Esto nunca se sabrá ya que no se ejecuta ninguna de las otras versiones, una vez se elige una, es la que se ejecuta.) Al calcular esa constante p, darle un valor específico para comparar un plan con otro, es lo que llamamos ***plan físico***.

Para optimizar el sistema, hay que reducir el tiempo de evaluación o de respuesta. Esto se basa en la reducción del coste de acceso a almacenamiento secundario y al coste computacional. El coste de acceso a almacenamiento secundario consiste en minimizar el acceso al disco, y con el coste computacional, nos referimos a que el sistema debe realizar el menor número de operaciones posibles, o en muchos de los casos, incluye operaciones que no están en la expresión de la consulta para poder reducir el número de datos y/o de bloques que se va a manejar. De esto se deriva que la consulta que yo realizo no tiene el mismo tamaño que la que consume el SGBD, ya que la del sistema es menor.

Tomemos como ejemplo de todo lo explicado los siguientes datos con su consulta:

Nos encontramos ante una base de datos de una Universidad, en la que tenemos las siguientes entidades: Alumno, Matricula y Asignatura, y nuestra consulta es saber el nombre de aquellos alumnos no becarios que tengan matrícula de honor.

- Base de datos:
  - Alumno (DNI, nombre, fec_nac, ciudad, direccion, tlfno, beca)
  - Asignatura (codigo, nombre, creditos, carácter, curso)
  - Matricula (codigo, DNI, calificacion)
- Consulta:

```
SELECT alumno.nombre FROM alumno, matricula
WHERE alumno.DNI = matricula.DNI AND beca = 'N'
AND calificacion LIKE 'SOBRESALIENTE HONOR%';
```

¿Cuál es el operador de dicha consulta? Un operador relacional, una reunión natural. Un join implica: producto cartesiano, selección y proyección.

Desde un punto de vista teórico, vamos a ver qué coste computacional y de espacio tendría el ejecutar dicha consulta, vamos a obtener distintas versiones que lleguen a un mismo resultado.

- Hipótesis:

Relación	NTuplas	Bfr	Cond1	Cond2	Cond1 Y Cond2
Alumno	500	5	100		15
Matricula	5000	10		120	

Tenemos 500 alumnos y 5000 matrículas. El factor de bloqueo para la tabla alumno es de 5 y para la tabla de matrícula es 10, esto supone que un bloque de la tabla alumno se compone de 5 registros, y un bloque de la tabla matrícula se compone de 10. Nos especifican que son 100 alumnos los que cumplen la primera condición de no becados, y 120 alumnos los que cumplen la segunda condición de Matrícula de honor, sin embargo en la última columna vemos que sólo son 15 los que cumplen ambas condiciones. Esta es la hipótesis de la que partimos para evaluar distintas respuestas.

1er plan

Este primer plan consiste en leer directamente la consulta que se ha planteado, la que aplica el producto cartesiano. El producto cartesiano une ambas tablas, es el producto de los elementos de cada tabla. En este ejemplo, hablaríamos de 2 500 000 registros (500 de Alumnos y 5000 de Matrícula). Estos registros se almacenarían en el almacenamiento temporal (tablespace) para poder seguir trabajando con ellas.

- Producto cartesiano:
  - Registros: 500 * 5000
  - Bloques de E/S: 500/5 * 5000/10 = 50000 bloques
  - Tuplas resultantes: 2500000
  - Tendrá que escribir a disco (Bfr = 3) luego habrá 833334 bloques

¿Cómo realizaríamos un producto cartesiano internamente? Con un bucle for anidado, pero nuestro problema aparece cuando los registros están en distinto bloque, ya que habría que realizar demasiadas lecturas de un mismo bloque. Una forma más eficiente sería multiplicar todo de los 2 primeros bloques, y luego con el tercer bloque, con lo que ya he recorrido uno de los bloques y no tengo que volver a cargarlo.

El tener 2 500 000 registros no implica que haya que leer los bloques 2 500 000 veces, no son tantas las operaciones de lectura y escritura sino que sólo voy a hacer 50000 operaciones de lectura, ya que voy a aprovechar cada bloque todo lo posible. ¿Hay alguna relación entre el factor de bloqueo del producto cartesiano y el factor de bloqueo de las dos tablas que están operando? Tiene que ser menor, ya que en un producto cartesiano los registros son más largos luego si entran más largos entran menos, por lo que el factor de bloqueo del producto cartesiano tiene que ser 5 o menos. Según la estimación, el resultado tendríamos que almacenarlos en 833 334 bloques.

Para la selección, leemos todos los bloques, es decir, cargamos los 833 334 bloques para después escribir sólo 15 registros en el resultado, que ocupan 5 bloques. ¿Una selección aumenta o reduce el número de atributos? Ni aumenta ni reduce, se mantiene el mismo número de atributos ya que una selección sólo disminuye el número de tuplas, no de atributos, por tanto, el factor de bloqueo de una selección permanece igual.

- Selección:
  - Leer todos los registros, 833334 bloques
  - Escribir 15 registros de resultado (5 bloques)

En la parte de la proyección, se eliminan aquellos elementos que estén repetidos. Vuelvo a leer los 5 bloques resultantes de la selección y para cada registro tengo que eliminar el campo repetido, aquellos DNI iguales. El factor de bloqueo de una proyección disminuye, ya que sólo me interesa el nombre de los alumnos, no la tupla entera. Los 15 nombres resultantes pueden ser almacenados en un bloque.

- Proyección
  - Leer los 5 bloques de la operación anterior
  - Sólo nombre de 15 registros caben en un bloque.

Como resultado de este plan tenemos que hemos leído 50 000 bloques para escribir después 833 334 bloques. A partir de aquí, volvemos a leer dichos bloques y escribimos 5 bloques, para seguidamente volver a leerlos y quedarnos sólo con 1 bloque.

Primer plan:

50000 + 833334 * 2 + 5 + 5 + 1 = 1716679 bloques

Como resultado, hemos estimado que utilizaremos 1 716 679 bloques, operaciones de lectura y escritura a disco, para resolver la consulta. Esto es muy costoso.

2o plan

Nuestra tabla Alumnos tiene un factor de bloqueo de 5 y 500 tuplas, por lo que se almacena en 100 bloques. Esta tabla Alumnos se encuentra ordenada por la clave de DNI – toda tabla que tiene clave está ordenada. Nuestra tabla Matrícula se compone de 5000 registros con un factor de bloqueo de 10, por lo que tiene 500 bloques.

La tabla Matrícula, por su parte, está ordenada primero por el código de asignatura y después por el DNI. Esto es un índice secundario y para usarlo hay que utilizarlo en el mismo orden en el que aparece, y el orden en el que se encuentra no nos es útil.

- Hipótesis:

Relación	NTuplas	Bfr	Cond1	Cond2	Cond1 Y Cond2
Alumno	500	5	100		15
Matricula	5000	10		120	

Por ello, podemos gastar un poco de tiempo ordenando dicha tabla según el DNI, ordenaríamos los registros entre sí, no los bloques. ¿Cuánto tardaría esta ordenación? Tardaría $n \cdot \log_2(n)$, en lo que la n es el número de bloques. (Este orden es el que tiene el algoritmo Quicksort que es el más rápido hasta hoy día.) Para ordenar la tabla, necesitaríamos $500 \cdot \log_2(500) = 4483$ operaciones de lectura y escritura de bloques.

- Ordenar matrícula:
  - 5000 registros en bloques de 10, 500 bloques
  - 500 * $\log_2(500)$ = 4483 bloques E/S

Tenemos ahora dos tablas ordenadas por el DNI, que las vamos a unir realizando un merge. Miro el primer registro de ambas tablas, ¿son iguales? Las junto al resultado. Después miro el primer registro de una tabla y el segundo de otra tabla, si son iguales, los uno al resultado, si no son iguales, avanzo el más pequeño de ambos. Así es como se realiza una reunión natural de la forma más simple posible.

Con el algoritmo Merge nos aseguramos que recorremos un elemento una única vez, sólo tenemos que leer un bloque una vez. Para realizar esta unión de ambas tablas, realizamos 600 operaciones de lectura, 500 bloques de Matrícula y 100 de Alumnos.

- Mezclar alumno y matricula por DNI:
  - 500 bloques + 100 bloques = 600 bloques
  - Se escriben 5000 registros de reunión (tamaño mayor y Bfr = 3), o sea, 1667 bloques
  - En total, 1667 + 600 bloques = 2267 bloques de E/S

A la hora de escribir, escribimos los registros de la reunión, aquellos que tengan el mismo DNI. Con la reunión, nos queda un factor de bloqueo de 3. Si tengo 5000 registros que tienen clave externa a los otros 500 registros, ¿cuántos registros tiene esa reunión natural? 5000, ya que lo que se está haciendo es extender las matrículas con la información de su alumno. Esto nos quedaría que escribimos 5000 registros con un factor de bloqueo de 3, por lo que estaríamos utilizando 1667 bloques.

Esta operación de mezcla se ha resuelto en las primeras 600 operaciones de lectura + 1667 para la mezcla, con un total de 2267 bloques de entrada y salida.

Una vez tengo ya la mezcla de ambas tablas, realizo la selección de aquellos registros que cumplan las condiciones. Tengo que volver a leer los 1667 bloques que he escrito anteriormente para quitarme aquellos que tengan beca y no tengan matrícula de honor.

- Selección de tuplas:
  - Leer 5000 registros, o sea 1667 bloques
  - Escribir 15 registros de resultado, o sea, 5 bloques
  - En total, 1667 + 5 = 1672 bloques

Finalmente, nos quedaremos con 15 registros y un factor de bloqueo de 3, el mismo que antes. La selección nos ha costado 1672 bloques, 1667 de lectura + 5 bloques de escritura.

Ya tenemos aquellos registros que cumplen ambas condiciones, lo que nos falta es quedarnos sólo con aquellos atributos que nos interesan, que en este caso son los nombres. Para ello, realizamos una proyección. Leemos los 5 bloques anteriores y escribimos el resultado en 1 bloque, como en el plan anterior.

- Proyección:
  - Leer los 5 bloques de la operación anterior.
  - Escribir 15 registros de sólo nombre, que caben en un bloque

Este segundo plan tiene un coste de:

Segundo plan:

4483 + 600 + 1667 * 2 + 5 + 5 + 1 = 8428 bloques

4483 bloques para ordenar la matrícula, 2267 bloques para mezclar alumno y matrícula por DNI, 1672 bloques para la selección de las tuplas y 6 bloques para realizar la proyección. Este plan ha necesitado 8428 operaciones de lectura y escritura para poder resolver la consulta. Como vemos, es mucho mejor que la anterior.

3er plan

¿Y si primero seleccionamos? Podemos primero seleccionar aquellos alumnos que no son becarios, lo que nos reduciría la población de 500 a 100, y seleccionar aquellas matrículas que tengan matrícula de honor, con lo que nos quedaríamos con 120 registros de los 5000 que teníamos. Vamos a ver cómo mejora la operación con este plan.

Primero aplicamos la selección. Leo la tabla de alumnos y me quedo con aquellas que no sean becarios y los guardo. Con esto he leído 100 bloques y he escrito 20. La selección de la tabla de Alumnos tiene un coste de 120 operaciones. Con esta operación he conseguido reducir la población considerablemente.

- Alumnos no becados:
  - Se leen 500 registros, o sea, 100 bloques
  - Se escriben 100 registros, o sea, 20 bloques
  - En total, 100 + 20 = 120 bloques

Realizo la misma operación anterior pero con la tabla de Matrículas. He realizado 500 operaciones de lectura, en la que leo la tabla, y 12 operaciones de escritura, en la que me quedo con aquellas tuplas que cumplan la condición. En total, 512 bloques.

- Seleccionar matrículas con esa calificación:
  - Se leen 5000 registros, o sea, 500 bloques
  - Se escriben 120 registros, o sea, 12 bloques
  - En total, 500 + 12 = 512 bloques

De estas dos tablas, me interesa saber aquellas matrículas que correspondan a alumnos no becados y con matrícula de honor. Para esto realizamos una mezcla, pero nos encontramos con el mismo problema que en el plan anterior: la tabla de matrícula no está ordenada como nos interesa, hay que reordenarla según el DNI. Esta reordenación tendrá un coste de $n \cdot \log_2(n)$, $12 \cdot \log_2(12) = 43$ bloques.

- Ordenar las matrículas seleccionadas:
  - 12 bloques * log2 (12) = 43 bloques

Ahora que la tabla de las matrículas está ordenada puedo realizar el merge que realizamos en el plan anterior. Leo los 20 bloques de alumnos y los 12 bloques de matriculas ordenadas por DNI, y el resultado son 32 bloques los que leo. De esos 32 bloques, escribo solo 5.

- Reunir alumnos seleccionados (100) y matrículas seleccionadas (120):
  - Se leen 20 + 12 = 32 bloques
  - Se escriben 5 bloques

Después, realizo una proyección para quedarme sólo con los nombres. Leo esos 5 bloques y escribo 1.

- Proyección:
  - Leer los 5 bloques escritos en la operación anterior.
  - 15 registros de sólo nombre en 1 único bloque

Este tercer plan ha tenido un coste total de 717 bloques, que se descompone en 120 bloques de la selección de la tabla de alumnos, 512 para la selección de la tabla de matrículas, 43 para la ordenación de las matrículas, 32 + 5 para la mezcla de ambas tablas y su resultado, y para la proyección 6, en total 718 bloques para este tercer plan.

Tercer plan:

120 + 512 + 43 + 32 + 5 + 5 = 717 bloques

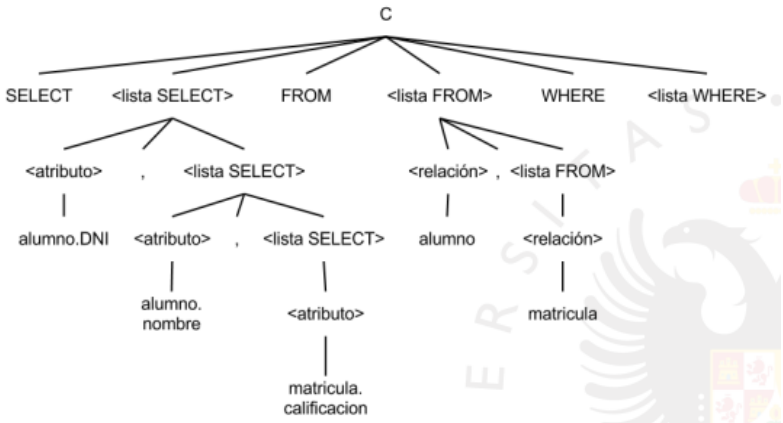
Le falta 1 bloque de escritura para la proyección

4o plan

¿Y si hay un índice sobre la calificación? Cuando se aplica esta operación sobre las matrículas que se queda con las matrículas de honor, tarda muy poco, ya que no tiene que acceder a todas las tuplas de la relación, porque sabe exactamente qué tuplas tienen dicha calificación.

Cuando se usan lenguajes estructurados para la consulta es muy fácil el uso de optimizadores. Como hemos visto al principio, podemos creer que la consulta que estamos haciendo la mejor consulta, pero el sistema realiza igualmente una optimización. Los optimizadores tienen una ventaja y es que usan información privilegiada acerca de las tablas y del sistema, como el tamaño de las relaciones, el número de tuplas, el factor de bloqueo, número de bloques. Para conseguir un mejor resultado de la consulta, hay sistemas que reestructuran los datos, otros que preprocesan la consulta, esto es, obtienen una consulta distinta, otros usan distintos optimizadores dependiendo de la estructura de datos que están almacenados. Con esto, podemos evaluar muchas más posibilidades.

La ventaja de los lenguajes de 4ª generación es que el análisis semántico (el significado de la consulta) es innecesario, ya que al ser el lenguaje de un propósito tan restringido no hace falta mirar más allá de la existencia de los objetos implicados en la consulta. A la representación sintáctica de la consulta (ver cómo están relacionadas las palabras) se le llama **árbol sintáctico**.



Este árbol sintáctico no tiene nada que ver con las bases de datos sino más bien con los lenguajes, ya que puedo tener estructuras que buscan lo mismo pero ser sintácticamente distintas. Por ello, necesitamos una representación no ambigua de la consulta que no esté relacionado con el lenguaje; a esta representación unívoca de una consulta que está representada en un árbol sintáctico se le llama **plan lógico**.

La consulta que yo realizo es procesada por el **procesador de consultas** que realiza transformaciones sobre ella y garantiza que dichas transformaciones devuelvan el mismo resultado que la consulta original. De un plan lógico genero otros tantos, de los cuales no se conoce su coste, por lo que no podemos saber si es mejor o peor que otro, por ello el sistema evalúa la constante p multiplicadora de cada plan lógico.

Esta constante es calculada por el sistema, aplicándole a cada versión los mismos cálculos, esto es, mismo número de bloques, mismas bondades, para obtener el resultado de una consulta, es decir, el número total de bloques para cada versión. Cuando tiene dicha constante, el SGBD puede decidir cuál plan va a ejecutar. Con esto, los planes lógicos pasan a ser **planes físicos**, planes ponderados que ya tienen su propio coste.

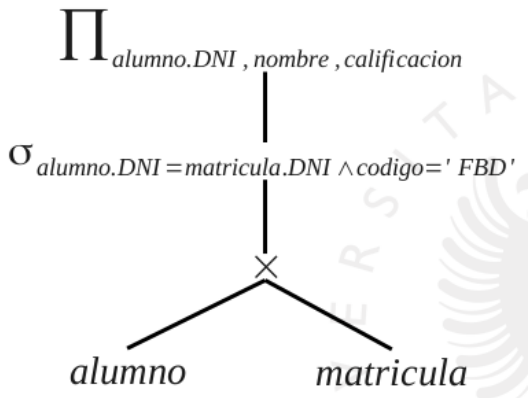
Cada plan lógico tiene varios planes físicos. Esto tiene sentido, ya que hay varias formas de resolver un producto cartesiano, al igual que hay otros operadores con distintas resoluciones. Podríamos usar siempre la mejor forma para cada operador, pero cada forma de un operador necesita distintas cantidades de recurso y el SGBD debe disponer de esos recursos para que el operador sea eficiente. Por tanto, elegiremos aquella resolución que necesite los recursos de los que dispone el sistema, además de tener en cuenta su eficiencia.

Producto cartesiano + proyección + selección = join

Cuando tenemos nuestros planes físicos, el selector de planes físicos escoge el más razonable, para después pasar el plan al **motor de ejecución**, para ejecutar el plan y devolverlo al usuario. Este es el proceso que se suele seguir dentro de la optimización.

La primera herramienta que aparece en todo este esquema es el árbol sintáctico que se construye a partir de la consulta mediante un análisis sintáctico. Este árbol es un árbol heterogéneo, es decir, que almacena información en las hojas mientras que en los nodos intermedios almacena operadores para llegar a dicha hoja. Los enlaces entre los nodos representan el orden en el que se aplican los operadores.

A partir del árbol sintáctico, se genera el **árbol algebraico** o **árbol de expresión algebraica**, que se corresponde con el plan lógico. De este árbol algebraico se pueden encontrar un número limitado de expresiones equivalentes, no hay un número infinito de versiones. A partir de esas transformaciones, el sistema las usa para eliminar operaciones que no tienen ningún uso y así conseguir una versión mejor de la consulta.



Estos árboles siempre se leen de abajo a arriba en postorden, es decir, primero se aplican los hijos y seguido el padre. En el ejemplo anterior, primero realizaríamos el join de alumno x matrícula, después la selección y finalmente la proyección.

Las transformaciones que se suelen hacer sobre una consulta son:

- ***Conmutatividad del producto, reunión, unión e intersección***: la conmutatividad implica que a·b = b·a. ¿Qué supone la conmutatividad aquí? Cuando tenemos que hacer la reunión natural de dos tablas, una de 100 bloques y otra de de 10 bloques, podemos hacer la reunión de la de 100 con la de 10 ó la de 10 con la de 100. Esto significa que una de las dos relaciones debe ordenarse, y es preferible ordenar la de 10 que la de 100. Aunque el resultado sea el mismo, varía en el número de operaciones que necesita para resolverlo.

  Cuando se trata de una reunión natural de 3 tablas: a x b x c, podemos hacer (a x b) x c, ó a x (b x c), pero no podemos hacer (a x c) x b, daría error porque el orden de la reunión natural implica que a y b tienen cosas en común, y b y c igual, mientras que a y c no tienen algo en común y no se puede realizar una reunión natural de dos tablas que no tengan algo en común.

- ***Asociatividad del producto, reunión y unión:*** tiene que ver con la conmutatividad, ya que este es un operador binario y cuando es necesario 3 tablas, primero asocias las 2 primeras o 2 últimas para luego juntarla con la restante.

- ***Movimiento de selección:*** intenta eliminar el mayor número de registros que no nos son necesarios lo antes posible.

- ***Movimiento de proyección:*** intenta eliminar el mayor número de atributos que no tengan nada que ver con la consulta lo antes posible.

- ***Producto cartesiano:*** hay que quitarse esta heurística lo antes posible.

¿Cuántas transformaciones podemos realizar? Para una consulta que involucra a n relaciones, las posibilidades de combinación entre ellas se ajustan más o menos a la fórmula:

$$\frac{(2 \cdot n-2)!}{(n-1)!}$$

Cuantas más relaciones tenga, más crece el factorial y, como resultado, el número de alternativas. Debido a esto, el optimizador genera un número de planes amplio pero limitado; genera un plan según alguna heurística.

Los primeros optimizadores no exploraban varias posibilidades, sino que aplicaban una heurística y el resultado era la solución que se ejecutaba, no planteaban si realmente era o no eficiente. Actualmente, los optimizadores suelen aplicar una solución fija, generan un plan inicial que no es el que le hemos dado sino aplicándole una heurística, y a partir de aquí, proponen modificaciones para estudiar el comportamiento de cada una de ellas, de entre las cuales el selector elegirá una.

El problema de los planes obtenidos es que no podemos ver cuanto cuesta ejecutándolo, porque si lo ejecutas ya tienes la solución. Por ello, el sistema accede a sus propias estadísticas sobre cada una de las relaciones para evaluar cuál es la mejor posible, pero sin alterar los datos.

Un sistema que mantenga actualizadas dichas estadísticas tomarán decisiones más ajustadas, más reales. Esta información que está almacenada depende de unos datos muy básicos, y el sistema puede mantener resultados intermedios, pero estos datos intermedios dependen de unos previos, han sido calculados en base a otros, por lo que el sistema tienen que mantener un montón de información adicional aparte de la estadística como relaciones, dependencias.

Para evaluar un plan físico, ponerle valor a un plan lógico, hace falta una serie de datos como pueden ser:

Para una relación:

- **N(R):** número de tuplas de R,
- **L(R):** longitud en bytes de la tupla de R,
- **Bfr(R):** el factor de bloqueo de R,
- **B(R):** número de bloques de la relación R.

Para un atributo:

- **V(R, atr):** la variabilidad es el número de valores distintos de un atributo en una tabla. Por ejemplo, ¿cuál es la variabilidad del atributo calificación? Calificación puede tomar los valores: matrícula, sobresaliente, notable, suficiente y suspenso, por lo tanto, tiene variabilidad 5. La variabilidad es discriminante, ya que a mayor variabilidad, menor tuplas en el resultado.
- **nI(R, atr):** es el índice de un atributo sobre una relación para no tener que ordenar en caso que haya índice. Muestra cuántos niveles tiene un índice multinivel o un B-árbol.

Para el sistema:

- **T(B):** el tamaño del bloque en bytes, lo necesita para poder calculas las operaciones intermedias.

Con estos parámetros podemos estimar cuánto ocupa cada operador.

Coste de un producto cartesiano

El producto cartesiano multiplica un elemento de una tabla por todos los elementos de la segunda tabla.

- N(X) se calculará entonces como el producto del número de tuplas de la primera tabla por el número de tuplas de la segunda tabla.

$$N(X)=N(R)\cdot N(S)$$

Si cada uno de un conjunto se combina con todos los del otro conjunto, por cada elemento del primer conjunto se tienen todos los del segundo. Esto es, el primero por n_2, el segundo por n_2, el tercero por n_2… . Si el n_2 lo sacas como factor común me queda una suma de todas las N, una suma de n_1 veces 1 multiplicado por n_2, que acaba siendo $n_1 \cdot n_2$.

- L(X) también se ve modificado, ya que el producto cartesiano concatena todo el contenido de un registro de una tabla con el contenido de la segunda, por lo que la longitud es la suma de las longitudes de los registros. Si los registros son de longitud variable, se ahorra un poco ya que se quita la marca de registro.

$$L(X)=L(R)+L(S)$$

Coste de una reunión natural

¿Qué relación hay entre la longitud de la tupla de la reunión natural y la del producto cartesiano? La de la reunión natural debe ser más pequeña. Mientras que el producto cartesiano almacena varias veces el mismo campo, la reunión natural sólo se queda con una de ellas. La longitud del registro de la reunión natural es, por tanto, menor que la del producto cartesiano, pero no mucho.

- L(X) es, por tanto, la suma de la longitudes de ambas tablas menos el tamaño del campo que comparten.

$$L(X)=L(R)+L(S)-\text{size}(b)$$

¿Cómo estimamos el número de tuplas que resultan de la reunión natural de dos tablas? Como límite superior tenemos el producto del número de tuplas de ambas tablas. Como al realizar la reunión natural nos quedamos con aquellas tuplas que tienen valores en común, nos quedamos con aquella tabla que tenga mayor variabilidad del atributo que comparten.

- N(X) se calcula como el producto del número de tuplas de ambas tablas dividido entre la variabilidad del atributo en común que sea más grande.

$$N(X)=\frac{N(R)\cdot N(S)}{max\{V(R,b),V(S,b)\}}$$

Si a la reunión natural le asignamos relaciones más pequeñas, las tuplas serán más pequeñas. Por dicha razón, se intentará realizar tanto proyecciones como selecciones al principio para reducir tanto el número de tuplas como la longitud de estas.

Coste de una selección

El mayor problema con la selección es estimar el número de tuplas que resultan. Para estimarlo necesitamos saber cuántas tuplas hay de cada valor distinto, pero si ya sabemos esto no necesitaríamos hacer nada. Como no sabe cuántas tuplas hay de cada valor (tampoco sería eficiente mantener y actualizar dicha información), asume una distribución uniforme en la que le asigna a cada valor la misma probabilidad, para ello se basa en la variabilidad.

Como ejemplo de esto: tenemos una población de 5000 matrículas, ¿cuántas puedo asumir según una distribución uniforme que tengan matrícula? Como la variabilidad de la asignatura dijimos que era 5: MH-SB-NT-AP-SP, a cada uno de esos valores le asignamos 1/5 de la población total; a MH le asignamos 1/5 de la población, es decir, estimamos que 1000 tuplas son de MH. Y así con los demás valores.

Para estimar el número de tuplas se usa un multiplicador α que se multiplica por el número de tuplas. Esta α depende de la condición por la cual estemos seleccionando.

- **Condición de igualdad**. Si estamos buscando la condición en la que el atributo debe ser igual a un valor, la α toma el valor del resultado de dividir 1 entre la variabilidad, le asignamos a cada valor la misma probabilidad. Se asume el peor de los casos.
- **Condición de comparación.** Si el atributo debe ser menor, menor o igual, mayor, o mayor o igual, selecciono un tercio de la población original. Se asume el caso promedio.
- **Condición de desigualdad.** Si el atributo debe ser distinto, asumo siempre el peor de los casos, por lo que selecciono todas las tuplas.

$$N(X)=\alpha\cdot N(R)$$

$$\text{donde } \alpha=\begin{cases}\frac{1}{V(R,atr)} \text{ si } \sigma_{atr=valor}\\ \frac{1}{3} \text{ si } \sigma_{atr<valor}\\ 1 \text{ si } \sigma_{atr\neq valor}\end{cases}$$

Para condiciones compuestas como el AND y el OR, para el AND aplicamos las selecciones en cascada y multiplicamos los factores de selección, y con el OR lo sustituimos por una unión de selecciones y la estimación sería:

$$N(R)\cdot(1-(1-\frac{n_1}{N(R)})\cdot(1-\frac{n_2}{N(R)}))$$

donde n_1 es el número de tuplas que cumplen la primera condición y n_2 es el número de tuplas que cumplen la segunda condición. El realizar 1 menos n_1 entre el número de tuplas totales me estaría devolviendo la probabilidad que hay de que no ocurra eso; lo mismo para n_2. Estaríamos multiplicando la probabilidad de que no pase la primera condición, por la probabilidad de que no pase la segunda. Si a 1 le resto el resultado de dicha multiplicación, estaría devolviendo la probabilidad de que ocurra.

El factor de bloque permanece igual, ya que la selección solo quita filas, no columnas. Sin embargo, altera el número de bloques que necesita el sistema, ya que reduce la tabla original a aquella con las tuplas que cumplen las condiciones.

Con la selección hemos reducido el conjunto de búsqueda es decir, a la tabla original le hemos eliminado aquellas filas que no nos son útiles, mientras que con la proyección nos hemos quedado con aquellos atributos que nos interesan, hemos eliminado las columnas que no nos sirven, por eso proyección y selección se realizan antes, pero esto lo realiza el sistema según vea, no depende de nosotros.

Coste de una proyección

Dada una relación R, como una proyección elimina columnas de una tabla, no filas, es decir, se queda con aquellos atributos que le son necesarios, los siguientes parámetros se ven afectados:

- L(X) se calcula como la sumatoria del tamaño de los atributos que componen la tupla resultante.

$$L(X) = \sum_{atr_i \in K} \text{size}(atr_i)$$

- Bfr(X) de la proyección se consigue restándole al tamaño del bloque la cabecera y dividiendo el resultado por el tamaño de los registros.

$$Bfr(X) = parte\ entera\left(\frac{T(B) - C}{L(X)}\right)$$

- B(X) se obtiene redondeando hacia arriba el número de registros (número de tuplas) entre el factor de bloqueo.

$$B(X) = redondeo\ hacia\ arriba\left(\frac{N(X)}{Bfr(X)}\right)$$

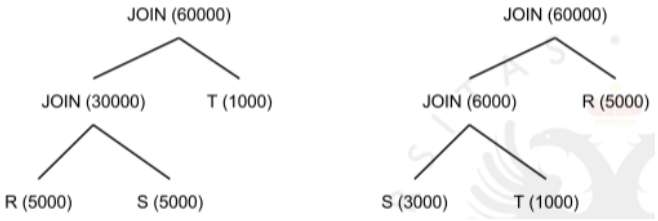
- N(X) se mantiene igual, ya que la proyección no altera el número de registros sino su longitud, esto es, elimina atributos de la relación.

Tenemos estas tres tablas y queremos hacer la reunión natural entre las 3, pero la reunión natural es una operación binaria, por lo que tendríamos que hacer primero una reunión natural entre dos de ellas, para después hacerlo con la tercera.

$$R \ JOIN \ S \ JOIN \ T$$

R(a,b)	S(b,c)	T(c,d)
N(R)=5000	N(S)=3000	N(T)=1000
V(R,a)=200		
V(R,b)=500	V(S,b)=400	
	V(S,c)=500	V(T,c)=100
		V(T,d)=20

Podemos empezar realizando el producto cartesiano de las dos primeras tablas, las cuales tienen 5000 y 3000 tuplas, que realizando el producto cartesiano nos quedaríamos con 15 000 000 de tuplas. Si reducimos, según la variabilidad (escogeríamos la que tiene mayor variabilidad que es la de R), nos quedaríamos con 3 000 registros. Una vez tenemos hecho el producto cartesiano de las dos primeras tablas, toca realizarlo con la tercera, que tiene 1000 registros. Del producto de ambas, resultarían 3 000 000 registros y dividiendo por la variabilidad (la mayor es la de S), llegaríamos a la solución con 6 000 registros. Este plan lógico se corresponde con el árbol de la izquierda.



Vamos a empezar, esta vez, primero realizando el producto cartesiano de S y T, para después realizarlo con R. El primer producto cartesiano de 3000 y 1000 nos dejaría con 3 000 000 de registros, que dividido entre la variabilidad mayor (la de S), resultaría 6 000 registros. Ahora realizamos el producto cartesiano con la tabla de R. Serían 6000 por 5000, que es 30 000 000 y vuelta a dividir entre la mayor variabilidad, nos quedaríamos con 6000. El árbol de la derecha se corresponde con este plan lógico.

Ambos parten y llegan a la misma solución, lo que importa es el número de bloques que ha necesitado para llegar al resultado. ¿Cuál es más eficiente? El de la derecha, porque usa menos bloques, lo que implica menos lectura y escritura de bloques, esto es, menos accesos al disco.

Pensamos en la siguiente consulta.

```
SELECT AL.DNI, nombre_al, calificacion
FROM alumnos AL, matricula MAT, asignaturas
AS
WHERE AL.DNI = MAT.DNI AND
   MAT.cod_asig = AS.cod_asig AND
   curso = 5 AND
   beca = 'S';
```

Si intentamos representarla, tenemos que presentar varios planes lógicos para cada una de las consultas, varios planes físicos para cada plan lógico, ademas de las varias implementaciones que podemos tener en cada caso. Si intentamos representar esto de alguna manera, al final lo que necesitamos es tomar una decisión basada en un número.