Introducción al procesamiento y optimización de consultas

Michael L. Rupley, Jr.
Universidad de Indiana en South Bend
mrupleyj@iusb.edu

Atributo	Longitud	¿Clave?
vehicle_id	15	Sí
Hacer	20	No
Modelo	20	No
año	4	No

Abstracto

Todos los sistemas de bases de datos deben ser capaces de responder a las solicitudes de información del usuario, es decir, procesar consultas. Obtener la información deseada de un sistema de base de datos de una manera predecible y fiable es el arte científico del *procesamiento*deconsultas. Obtener estos resultados de forma oportuna se ocupa de la técnica de *optimización*de consultas. Este documento presentará al lector los conceptos básicos de procesamiento de consultas y optimización de consultas en el dominio de base de datos relacional. También se presentará la forma en que una base de datos procesa una consulta, así como algunos de los algoritmos y conjuntos de reglas utilizados para producir consultas más eficientes. En la última sección, discutiré el plan de implementación para ampliar las capacidades de mi programa Mini-Database Engine para incluir algunas de las técnicas y algoritmos de optimización de consultas que se tratan en este documento.

1. INTRODUCCION

El procesamiento y optimización de consultas es una parte fundamental, si no crítica, de cualquier DBMS. Para ser utilizado eficazmente, los resultados de las consultas deben estar disponibles en el período de tiempo necesario para el usuario que lo envía, ya sea una persona, una máquina de montaje robótica o incluso otro DBMS distinto e independiente. La forma en que un DBMS procesa las consultas y los métodos que utiliza para optimizar su rendimiento son temas que se tratarán en este documento.

En algunas secciones de este documento, se ilustrarán varios conceptos con referencia a una base de datos de ejemplo de automóviles y conductores. Cada coche y conductor son únicos, y cada coche puede tener 0 o más conductores, pero sólo un propietario. Un conductor puede poseer y conducir varios coches. Hay 3 relaciones: *coches, conductores* y *car_driver* con los siguientes atributos:

La relación de *vehículos* :

owned_by	10	No

La relación *de los conductores* :

Atributo	Longitud	¿Clave?
driver_id	10	Sí
First_name	20	No
last_name	20	No
Edad	2	No

La relación *car_del conductor:* :

Atributo	Longitud	¿Clave?
cd_car_name	15	Sí*
*_id deconductorcd_*	10	Sí*

2. ¿QUÉ ES UNA CONSULTA?

Una consulta de base de datos es el vehículo para indicar a un DBMS que actualice o recupere datos específicos hacia/desde el medio almacenado físicamente. La actualización y recuperación real de datos se realiza a través de varias operaciones de "bajo nivel". Ejemplos de tales operaciones para un DBMS relacional pueden ser operaciones de álgebra relacional como proyecto, unión, selección, producto cartesiano, etc. Mientras que el DBMS está diseñado para procesar estas operaciones de bajo nivel de manera eficiente, puede ser bastante la carga para un usuario enviar solicitudes al DBMS en estos formatos. Considere la siguiente solicitud:

"Dame las identificaciones de vehículos de todo Chevrolet Camaros construido en el año 1977."

Si bien esto es fácilmente comprensible por un ser humano, un
El DBMS se debe presentar con un formato que pueda entender, tal como esta declaración SQL:

seleccionar
vehicle_id **de** *los*

vehículos en los
que *el año* 1977

Tenga en cuenta que el DBMS todavía tendrá que traducir aún más esta instrucción SQL para que las funciones/métodos dentro del programa DBMS no solo puedan procesar la solicitud, sino hacerlo de manera oportuna.

3. EL PROCESADOR DE CONSULTAS

Hay tres fases [12] que una consulta pasa durante el procesamiento de esa consulta por parte del DBMS:

1. Análisis y traducción
2. Optimización
3. Evaluación

La mayoría de las consultas enviadas a un DBMS están en un lenguaje de alto nivel, como SQL. Durante la fase de análisis y traducción, la forma legible de la consulta se traduce en formularios utilizables por el DBMS. Pueden estar en forma de expresión de álgebra relacional, árbol de consulta y gráfico de consulta. Considere la siguiente consulta SQL:

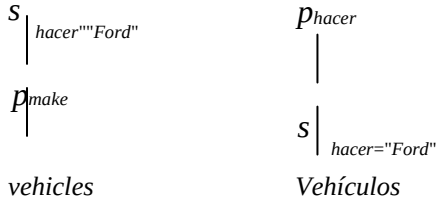
seleccionar *hacer* **de**
los vehículos **donde**
hacer "Ford"

Esto se puede traducir en cualquiera de las siguientes expresiones de álgebra relacional:

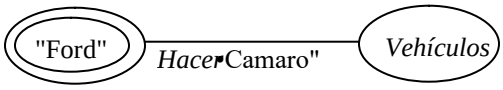
$$s_{make="Ford"}(p_{make}(vehicles))$$

$$p_{make}(s_{makeá"Ford"}(vehículos))\ "$$

Que también se puede representar como cualquiera de los siguientes árboles de consulta:



Y se representa como un gráfico de consulta:



Después de analizar y traducir en una expresión de álgebra relacional, la consulta se transforma en un formulario, normalmente un árbol de consulta o gráfico, que puede controlar el motor de optimización. A continuación, el motor de optimización realiza varios análisis de los datos de consulta, generando una serie de planes de evaluación válidos. A partir de ahí, determina el plan de evaluación más adecuado para ejecutar.

Una vez seleccionado el plan de evaluación, se pasa al motor de ejecución de consultas del DMBS [12] (también denominado procesador de base de datos en tiempo de ejecución [5]), donde se ejecuta el plan y se devuelven los resultados.

3.1 Analizar y traducir la consulta

El primer paso para procesar una consulta enviada a un DBMS es convertir la consulta en un formulario que pueda usar el motor de procesamiento de consultas. Los lenguajes de consulta de alto nivel, como SQL, representan una consulta como una cadena o secuencia de caracteres. Ciertas secuencias de caracteres representan varios tipos de tokens como palabras clave, operadores, operandos, cadenas literales, etc. Al igual que todos los lenguajes, hay reglas (sintaxis y gramática) que rigen cómo se pueden combinar los tokens en instrucciones comprensibles (es decir, válidas).

El trabajo principal del analizador es extraer los tokens de la cadena sin procesar de caracteres y traducirlos en los elementos de datos internos correspondientes (es decir, operaciones de álgebra relacional y operandos) y estructuras (es decir, árbol de consultas, gráfico de consulta).

El último trabajo del analizador es comprobar la validez y la sintaxis de la cadena de consulta original.

3.2 Optimización de la consulta

En esta etapa, el procesador de consultas aplica reglas a las estructuras de datos internas de la consulta para transformar estas estructuras en representaciones equivalentes, pero más eficientes. Las reglas pueden basarse en modelos matemáticos de la expresión de álgebra relacional y el árbol (heurística), tras estimaciones de costos de diferentes algoritmos aplicados a las operaciones o en la semántica dentro de la consulta y las relaciones que implica. Seleccionar las reglas adecuadas para aplicar, cuándo aplicarlas y cómo

se aplican es la función del motor de optimización de consultas.

3.3 Evaluación de la consulta

El último paso en el procesamiento de una consulta es la fase de evaluación. Se selecciona el mejor candidato del plan de evaluación generado por el motor de optimización y, a continuación, se ejecuta. Tenga en cuenta que puede haber varios métodos para ejecutar una consulta. Además de procesar una consulta de una forma secuencial simple, algunas de las operaciones individuales de una consulta se pueden procesar en paralelo, ya sea como procesos independientes o como canalizaciones interdependientes de procesos o subprocesos. Independientemente del método elegido, los resultados reales deben ser los mismos.

4. MÉTRICAS DE CONSULTA: COSTO

El tiempo de ejecución de una consulta depende de los recursos necesarios para realizar las operaciones necesarias: accesos a discos, ciclos de CPU, RAM y, en el caso de sistemas paralelos y distribuidos, comunicación de subprocesos y procesos (que no se tendrán en cuenta en este documento). Dado que la transferencia de datos a/desde discos es sustancialmente más lenta que las transferencias basadas en memoria, los accesos al disco suelen representar una abrumadora mayoría del costo total, especialmente para bases de datos muy grandes que no se pueden cargar previamente en la memoria. Con los equipos de hoy en día, el costo de la CPU también puede ser insignificante en comparación con el acceso al disco para muchas operaciones.

El costo de acceso a un disco se mide generalmente en términos del número de bloques transferidos desde y hacia un disco, que será la unidad de medida mencionada en el resto de este documento.

5. EL PAPEL DE LOS ÍNDICES

La utilización de índices puede reducir drásticamente el tiempo de ejecución de varias operaciones, como seleccionar y unir. Revisemos algunos de los tipos de estructuras de archivos de índice y los roles que desempeñan en la reducción del tiempo de ejecución y la sobrecarga:

Densa Index: El archivo de datos se ordena mediante la clave de búsqueda y cada valor de clave de búsqueda tiene un registro de índice independiente. Esta estructura requiere una sola búsqueda para encontrar la primera aparición de un conjunto de registros contiguos con el valor de búsqueda deseado.

*Índice*disperso: el archivo de datos se ordena por la clave de búsqueda de índice y sólo algunos de los valores de clave de búsqueda tienen correspondientes. El puntero de archivo de datos de cada registro de índice apunta al primer registro de archivo de datos con el valor de clave de búsqueda. Aunque esta estructura puede ser menos eficaz (en términos de número de accesos a disco) que un índice denso para encontrar los registros deseados, requiere menos espacio de almacenamiento y menos sobrecarga durante las operaciones de inserción y eliminación.

Primary Index: el archivo de datos está ordenado por el atributo que también es la clave de búsqueda en el archivo de índice. Los índices primarios pueden ser densos o escasos. Esto también se conoce como un archivo secuencial de índice [5]. Para escanear los registros de una relación en orden secuencial por un valor clave, esta es una de las estructuras más rápidas y eficientes: localizar un registro tiene un costo de 1 búsqueda, y la composición contigua de los registros en orden ordenado minimiza el número de bloques que se deben leer. Sin embargo, después de un gran número de inserciones y eliminaciones, el rendimiento puede degradarse bastante rápidamente y la única manera de restaurar el rendimiento es realizar una reorganización.

*Segundo*índice: el archivo de datos está ordenado por un atributo que es diferente de la clave de búsqueda en el archivo *de* índice. Los índices secundarios deben ser densos.

*Indice de varios*niveles: una estructura de índice que consta de 2 o más niveles de registros donde los registros de un nivel superior apuntan a los registros de índice asociados del nivel siguiente. Los registros de índice del nivel inferior contienen los punteros a los registros de archivos de datos. Los índices de varios niveles se pueden utilizar, por ejemplo, para reducir el número de lecturas de bloques de disco necesarias durante una búsqueda binaria.

Clustering Index: Una estructura de índice de dos niveles donde los registros del primer nivel contienen el valor del campo de agrupación en clústeres en un campo y un

segundo campo que apunta a un bloque [deregistros de 2nd nivel] en el segundo nivel. Los registros del segundo nivel tienen un campo que apunta a un registro de archivo de datos real o a otrobloque de 2nd nivel.

B+*-tree Index*: índice de variosniveles con una estructura de árbol equilibrado. Encontrar un valor de clavede búsqueda en un árbol B+es proporcional a la altura del árbol: el número máximo de búsquedas requeridas es □álg(*height*)□). Si bien esto, en promedio, es más que un índice denso de un solo nivel que requiere una sola búsqueda, laestructura B+-tree tiene una clara ventaja en que no requiere reorganización, es autooptimizadora porque el árbol se mantiene equilibrado durante las inserciones y eliminaciones. Muchas aplicaciones de misión crítica requieren un alto rendimiento con un tiempo de actividad cercano al 100 %, lo que no se puede lograr con estructuras que requieran reorganización. Las hojas del árbol B+se utilizan para reorganizar el archivo de datos.

6. ALGORITMOS DE CONSULTA

En última instancia, las consultas se reducen a una serie de operaciones de análisis de archivos en las estructuras de archivos físicos subyacentes. Para cada operación relacional, puede haber varias rutas de acceso diferentes a los registros concretos necesarios. El motor de ejecución de consultas puede tener una multitud de algoritmos especializados diseñados para procesar combinaciones de rutas de acceso y operaciones relacionales específicas. Veremos algunos ejemplos de algoritmos para las operaciones de selección y combinación.

6.1 Algoritmos de selección

La operación Seleccionar debe buscar registros que cumplan los criterios de selección en los archivos de datos. Los siguientes son algunos ejemplos de algoritmos de selección simples (un atributo) [13]:

S1. Búsqueda lineal: cada registro del archivo se lee y se compara con los criterios de selección. El costo de ejecución para la búsqueda en un atributo no clave es b_r,donde b_r es el número de bloques en el archivo que representa la relación r. En un atributo clave, el costo medio es b_r / 2, con el peor caso de b_r.

S2. Búsqueda binaria en la clave principal: Una búsqueda binaria, en igualdad, realizada en un atributo de clave principal (archivo ordenado por la clave) tiene un costo en el peor de loscasos delg(b_r)□. Esto puede ser significativamente más eficaz que la búsqueda lineal, especialmente para un gran número de registros.

S3. Buscar mediante un índice principal en igualdad: con un índice B+-tree, una comparación de igualdad en un atributo de clave tendrá un costo en el peor de los casos del alto del árbol (en el archivo de índice) más uno para recuperar el registro del archivo de datos. Una comparación de igualdad en un atributo no clave será la misma, excepto que varios registros pueden cumplir la condición, en cuyo caso, agregamos el número de bloques que contienen los registros al costo. S4. Buscar utilizando un índice primario en la comparación: cuandose utilizan los operadores de comparación (<, ≤, >, ,, ,)para recuperar varios registros de un archivo ordenado por el atributo debúsqueda, se encuentra el primer registro que cumple la condición y se añaden los bloques totales antes de (<,)o después(>,)al costo de localizar el primerregistro.

S5. Buscar mediante un índice secundario en igualdad: recuperar un registro con una comparación de igualdad en un atributo de clave; o recuperar un conjunto de registros en un atributo sin clave. Para un único registro, el costo será igual al costo de localizar la clave de búsqueda en el archivo de índice más uno para recuperar el registro de datos. Para varios registros, el costo será igual al costo de localizar la clave de búsqueda en el archivo de índice más un acceso de bloque para cada recuperación de registros de datos, ya que el archivo de datos no se ordena en el atributo de búsqueda.

6.2 Algoritmos de unión

Al igual que la selección, la operación de combinación se puede implementar de varias maneras. En términos de accesos a discos, las operaciones de combinación pueden ser muy costosas, por lo que implementar y utilizar algoritmos de combinación eficientes es fundamental para minimizar el tiempo de ejecución de una consulta. Los siguientes son 4 tipos conocidos de algoritmos de combinación:

J1. Combinación de bucle anidado: este algoritmo consta de un bucle for interno anidado dentro de un bucle for externo. Para ilustrar este algoritmo, utilizaremos las siguientes notaciones:

r, s Relaciones r y s t_r Tuple (registro) en relación r t_s Tuple (registro) en relación s n_r Número de registros en relación r n_s Número de registros en relación s b_r Número de bloques con registros en relación r b_s Número de bloques con registros en relación s

Aquí hay una lista de pseudo-código de ejemplo para unirse a las dos relaciones r y s utilizando el bucle anidado-for [12]:

para cada tupla t_r en r
para cada tupla t_s en s
si la condición de combinación es verdadera para (t_r, t_r)
añadir t_r+t_s al resultado

Cada registro en la relación externa r se escanea una vez, y cada registro en la relación interna s se escanea n_r.veces, lo que resulta en n_r* n_s exploraciones de registro total. Si solo un bloque de cada relación puede caber en la memoria, el costo (número de accesos a bloques) es nr* b_s + b_r [12]. Si todos los bloques de ambas relaciones pueden caber en la memoria, el costo es b_r + b_s [12]. Si todos los bloques de la relación s (la relación interna) pueden caber en la memoria, entonces el coste es idéntico a ambas relaciones que encajan en la memoria: b_r + b_s [12]. Por lo tanto, si una de las relaciones puede caber completamente en la memoria, entonces es ventajoso para el optimizador de consultas seleccionar esa relación como la interna.

Aunque el peor caso para la combinación de bucle anidado es bastante costoso, tiene una ventaja en que no impone ninguna restricción en las rutas de acceso para cualquier relación, independientemente de la condición de combinación.

J2. Combinación de bucle anidado de índice: este algoritmo es el mismo que la unión de bucle anidado, excepto que se utiliza un archivo de índice en el atributo join de la relación interna*(s)*frente a un análisis de archivo de datos en s—cada búsqueda de índice en el bucle interno es esencialmente una selección de igualdad

en s utilizando uno de los algoritmos de selección (por ejemplo. S2, S3, S5). Deje *que c* sea el costo para la búsqueda, entonces el costo del peor de los casos para unirse s b_r $a r$ + n_r * c [12].

J3. Combinación de combinación de ordenación: Este algoritmo se puede utilizar para realizar uniones naturales y equi-uniones y requiere que cada relación (r y s) seordene por los atributos comunes entre ellos (R - S)[12]. Los detalles sobre cómo funciona este algoritmo se pueden encontrar en [5] y [12] y no se presentarán

Aquí. Sin embargo, es notable señalar que cada registro en r y s sólo se escanea una *vez,*produciendo así un peor y mejor costo de b_r + b_s [12]. Las variaciones del algoritmo Sort-Merge Join se utilizan, por ejemplo, cuando los archivos de datos están en orden no ordenado, pero existen índices secundarios para las dos relaciones.

J4. Combinación hash: Al igual que la combinación de combinación de ordenación, el algoritmo de combinación hash se puede utilizar para realizar combinaciones naturales y equi-unions [12]. La combinación hash utiliza dos estructuras de archivos de tabla hash (una para cada relación) para dividir los registros de cada relación en conjuntos que contienen valores hash idénticos en los atributos de combinación. Se analiza cada relación y se crea su tabla hash correspondiente en los valores de atributo join. Tenga en cuenta que pueden producirse colisiones, lo que da como resultado algunas de las particiones que contienen diferentes registros de conjuntos con valores de atributo de combinación coincidentes. Después de compilar las dos tablas hash, para cada partición coincidente de las tablas hash, se crea un índice hash inmemory de los registros de relación más pequeña (la relación de compilación) y se realiza una combinación de bucle anidado en los registros correspondientes en la otra relación, escribiendo el resultado de cada combinación.

Tenga en cuenta que lo anterior solo funciona si la cantidad necesaria de memoria está disponible para contener el índice hash y los registros numéricos en cualquier partición de la relación de compilación. Si no es así, se realiza un proceso conocido como *partición recursiva:* consulte [5] o [12] para obtener más información.

El costo de la combinación hash, sin particionamiento recursivo, es 3(b_r + b_s) +4n_h donde n_h es el número de particiones en la tabla hash [12]. El costo de la combinación hash con particionamiento recursivo es de $2(b_r + b_s), \log_{M,1}(b_s), 1, + b_r +$ b $+ s, s$

donde M es el número de bloques de memoria utilizados.

7. OPTIMIZACIÓN DE CONSULTAS

La función del motor de optimización de consultas de un DBMS es encontrar un plan de evaluación que reduzca el costo total de ejecución de una consulta. Hemos visto en las secciones anteriores que los costos para realizar operaciones particulares, como seleccionar y unir, pueden variar bastante drásticamente. Por ejemplo, considere 2 relaciones r y s, conlas siguientes características:

10.000 nn_r - Número de tuplas en r
1.000 s n_s - Número de tuplas en s
1.000 á b_r - Número de bloques con tuplas en r
100 á b_s - Número de bloques con tuplas en s

La selección de un único registro delg(b_r)□= r en un atributo no clave puede tener, un coste de 10(búsqueda binaria) o un coste de b_r /2 a 5.000 (búsqueda lineal). La unión de r y s puede tener un coste de n_r áb_s +b_r á1.001.000 (unión de bucle anidado)[13] o un coste de 3(b_r +b_s)+ 4n_h a 73.000(hashjoin donde n_h á 10,000)[13].

Tenga en cuenta que la diferencia de coste entre las 2 selecciones difiere en un factor de 500, y la 2 se une por un factor de 14. Claramente, la selección de métodos de menor costo puede resultar en un rendimiento sustancialmente mejor. Este proceso de selección de un mecanismo de menor costo se conoce como optimización basada en costos. Otras estrategias para reducir el tiempo de ejecución de las consultas incluyen la optimización basada en heurística y la optimización basada en la semántica.

En la optimización basada en heurística, las reglas matemáticas se aplican a los componentes de la consulta para generar un plan de evaluación que, teóricamente, dará lugar a un menor tiempo de ejecución. Normalmente, estos componentes son los elementos de datos dentro de una estructura de datos interna, como un árbol de consultas, que el analizador de consultas ha generado a partir de una representación de nivel superior de la consulta (es decir, SQL).

Los nodos internos de un árbol de consultas representan operaciones de álgebra relacional específicas que se realizarán en las relaciones que se les pasan desde los nodos secundarios directamente debajo. Las hojas del árbol son las relaciones. El árbol se evalúa de abajo hacia arriba, creando un plan de evaluación específico. En la sección 3, vimos que el árbol de consultas de una consulta se puede construir de varias maneras equivalentes. En muchos casos, habrá al menos uno de estos árboles equivalentes que produce un plan de ejecución más rápido y "optimizado". La Sección 7.2 ilustrará este concepto.

Otra forma de optimizar una consulta es la optimización de consultas basadas en semánticas. En muchos casos, los datos dentro y entre las relaciones contienen "reglas" y patrones que se basan en situaciones del "mundo real" que el DBMS no "conoce". Por ejemplo, vehículos como el Delorean no se hicieron después de 1990, por lo que una consulta como "Recuperar todos los vehículos con hacer igual a Delorean y año > 2000" producirá cero registros. La inserción de estos tipos de reglas semánticas en un DBMS puede mejorar aún más el tiempo de ejecución de una consulta.

7.1 Estadísticas de resultados de expresión

Para estimar los diversos costos de las operaciones de consulta, el optimizador de consultas utiliza una cantidad bastante extensa de metadatos asociados con las relaciones y sus estructuras de archivos correspondientes. Estos datos se recopilan durante y después de varias operaciones de base de datos (como consultas) y se almacenan en el catálogo de DBMS. Estos datos incluyen [5, 12]:

$n_{r\,r}$ Número de registros (tuplas) en una relación r. Conocer el número de registros en una relación es una pieza crítica de datos utilizada en casi todas las estimaciones de costos de las operaciones. f_r Factor de bloqueo (número de registros por bloque) para la relación r. Estos datos se utilizan para calcular el factor de bloqueo y también son útiles para determinar el tamaño y el número adecuados de búferes de memoria. $b_{r\,r}$ Número de bloques en relación r's data-file. También un dato crítico y comúnmente utilizado, b_r se calcula el valor igual a n_r / b_r. l_r Longitud de un registro, en bytes, en relación r. El tamaño del registro es otro elemento de datos importante utilizado en muchas operaciones, especialmente cuando los valores difieren significativamente para

dos relaciones implicadas en una operación. Para los registros de longitud variable, el valor de longitud real utilizado (el promedio o el máximo) depende del tipo de operación que se va a realizar.

d_{Ar} Número de valores distintos del atributo *A* en la relación *r*. Este valor es importante para calcular el número de registros resultantes para una operación de proyección y para funciones agregadas como **sum**, **count** y **average**. *x* Número de niveles en un índice de varios niveles (B^+-tree, índice de clúster, etc.). Este elemento de datos se utiliza para estimar el número de accesos a bloques necesarios en varios algoritmos de búsqueda. Tenga en cuenta que para un

B^+-tree, *x* será igual a la altura del árbol.

s_{Una} cardinalidad de selección de un atributo. Este es un valor calculado igual a n_r / d_{Ar}. Cuando *A* es un

atributo clave, s_A a 1. La cardinalidad de selección permite al optimizador de consultas determinar el "número medio de registros que satisfarán una condición de selección de igualdad en ese atributo"[5].

El optimizador de consultas también depende de otros datos importantes, como el orden del archivo de datos, el tipo de estructuras de índice disponibles y los atributos implicados en estas estructuras de organización de archivos. Saber si existen determinadas estructuras de acceso permite que el optimizador de consultas seleccione los algoritmos adecuados que se utilizarán para determinadas operaciones.

7.2 Expresión y transformaciones de árboles

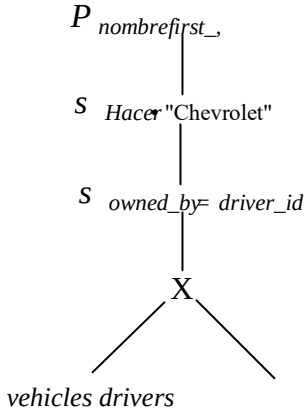
Después de analizar una consulta de alto nivel (es decir, una instrucción SQL) en una expresión de álgebra relacional equivalente, el optimizador de consultas puede realizar reglas heurísticas en la expresión y el árbol para transformar la expresión y el árbol en formularios equivalentes, pero optimizados. Por ejemplo, considere la siguiente consulta SQL:

seleccionar *first_name, last_name* **de** *los conductores, vehículos* **donde** *hacer* "Chevrolet" y *owned_by* driver_id *driver_id*

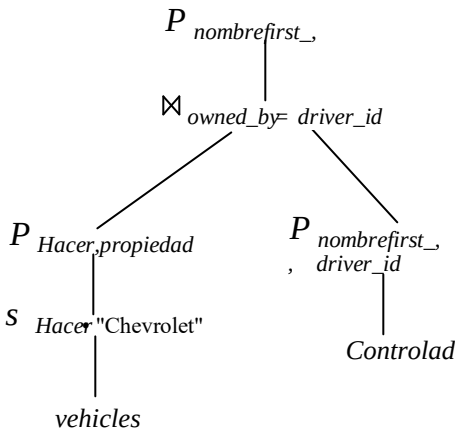
Una expresión de álgebra relacional correspondiente es:

$p_{first\_name,last\_name}((s_{make} á "Chevrolet" (s_{owned\_by} ? driver\_id$ (*Vehículos* X *conductores*)))

Y el árbol de consulta canónico correspondiente para la expresión de álgebra relacional:



Supongamos que las *relaciones* entre vehículos y *conductores* tienen 10.000 registros cada uno y el número de vehículos Chevrolet es de 5.000. Tenga en cuenta que el producto cartesiano que resulta en 10,000,000 registros se puede reducir en un 50% si la s_{make} "Chevrolet".operation se realiza primero. También podemos combinar las operaciones de producto *owned_by* $_{owned\_by\ driver\_id\ y}$ cartesiana s en una operación de unión más eficiente, así como eliminar las columnas innecesarias antes de realizar la costosa unión. El siguiente diagrama muestra esta mejor versión "optimizada" del árbol:



En el álgebra relacional, hay varias definiciones y teoremas que el optimizador de consultas puede usar

para transformar la consulta. Por ejemplo, la definición de relaciones equivalentes indica que el conjunto de atributos (dominio) de cada relación debe ser el mismo, porque son conjuntos, el orden no importa. Aquí hay una lista parcial de teoremas de álgebra relacional del libro de texto Elmasri/Navathe [5]:

1. Cascada de s : Una selección con condiciones conjuntivas en la lista de atributos equivale a una *cascada* de selecciones al seleccionar:

   $s_{A_1 \wedge A_2 \wedge \ldots} \ldots {}_{A A_n}(R)$ ás $_{A_1}(s_{A_2}(\ldots(s_{A_n}(R))\ldots))$

2. Commutativity of s : La operación de selección es conmutativa:$s_{A_1}(s_{A_2}(R))$ s $_{A_2}(s_{A_1}(R))$

3. Cascada de p : Una *cascada* de operaciones del proyecto es equivalente a la última operación del proyecto de la cascada:

   $p_{ALista_1}(p_{ALista_2}(\ldots(p_{ALista_n}(R))\ldots)) \bullet p_{ALista_1}(R)$

4. Desplazamiento s con p : Dada una lista de atributos p 's and s de $A_1,A_2,\ldots,A_n$,las operaciones p y s se pueden conmutar:

   $p_{A_1,A_2,\ldots,A_n}(s_c(R))$ ác $_c(p_{A_1,A_2,\ldots,A_n}(R))$

5. Comunatividad de ⋈X: Las operaciones de unión y de productos cartesianos son comunitivas: R⋈S s R y R X S ⋈ S X R

6. Desplazamiento s con ⋈ o X: Seleccione se puedeconmutar con unión (o producto cartesiano) de la siguiente manera:

   a. Si todos los atributos en la condición de la selección están en la relación R entonces s_c (R⋈S) á(s_c (R)) S ⋈ b. Dada la selección de la condición c compuesta de las condiciones c1 y c2, y c1 contiene solamente los atributos de R, y c2 contiene solamente los atributos de S, después s_c (R⋈S) á(s_{c1} (R))⋈(s_{c2} (S)) 7. La conmutatividad de las operaciones de conjunto (∪,∩,,− ,): Las operaciones de unión e intersección son conmutativas; pero la operación de diferencia no es:

   $$R \cup S \equiv S \cup R, \quad R \cap S \equiv S \cap R, \quad R - S \neq S - R$$

8. Asociatividad de ⋈ , X, y : Las cuatro operaciones son asociativas individualmente. Deje que sea any cualquiera de estos operadores, entonces::

   (R S) •T á≡R á (S á T)

9. (Desplazamiento θ ≡— θ *one* s con operaciones de set (ss ,s, $_c$ (R))) θ (s_c (S))

10. Desplazamiento p con : Las operaciones de proyecto y de unión se pueden conmutar: $p_{Lista\ A}$ (*R*s*S*) ≡,p_{Lista} A (R) , p Lista$_A$ (*R*),p Lista A (R)(p Lista A ($_{Lista\ A}$((*S*))

Con estos teoremas, se puede definir un algoritmo para transformar la expresión/árbol de consulta original creado por el analizador en una consulta más optimizada. Un ejemplo detallado de tal algoritmo se puede encontrar en el libro de texto de Elmasri/Navathe [5], algunos de los conceptos clave se pueden resumir de la siguiente manera:

1. Un objetivo principal es reducir el tamaño de las relaciones intermedias, tanto en términos de bytes por registro como en número de registros, lo antes posible para que las operaciones posteriores tengan menos datos para procesar y, por lo tanto, se ejecuten más rápido.
2. Las operaciones, como las selecciones coyuntar, deben desglosarse en su conjunto equivalente de unidades más pequeñas para permitir que las unidades individuales se muevan a posiciones "mejores" dentro del árbol de consultas.
3. Combine productos cartesianos con las selecciones correspondientes para crear combinaciones, utilizando algoritmos de combinación optimizados como la combinación de combinación de ordenación y la combinación hash puede ser órdenes de magnitud más eficientes.
4. Mueva las selecciones y los proyectos lo más abajo posible del árbol, ya que estas operaciones producirán relaciones intermedias más pequeñas que se pueden procesar más rápidamente mediante las operaciones anteriores.

7.3 Elección de planes de evaluación

El motor de optimización de consultas normalmente genera un conjunto de planes de evaluación de candidatos. Algunos, en la teoría heurística, producirán una ejecución más rápida y eficiente. Otros pueden, por resultados históricos anteriores, ser más eficientes que los modelos teóricos, esto puede muy bien ser el caso de las consultas dependientes de la naturaleza semántica

de los datos que se van a procesar. Otros pueden ser más eficientes debido a "agencias externas" como la congestión de la red, aplicaciones de la competencia en la misma CPU, etc. Por lo tanto, puede existir una gran cantidad de datos a partir de los cuales el motor de ejecución de consultas puede sondear el mejor plan de evaluación para ejecutar en un momento dado.

10. CONCLUSION

Uno de los requisitos funcionales más críticos de un DBMS es su capacidad para procesar consultas de manera oportuna. Esto es particularmente cierto para aplicaciones muy grandes y de misión crítica, como la predicción meteorológica, los sistemas bancarios y las aplicaciones aeronáuticas, que pueden contener millones e incluso billones de registros. La necesidad de resultados más rápidos y rápidos, "inmediatos" nunca cesa. Por lo tanto, una gran cantidad de investigación y recursos se dedican a crear motores de optimización de consultas más inteligentes y altamente eficientes. Algunas de las técnicas básicas de procesamiento y optimización de consultas se han presentado en este documento. Otros temas más avanzados son los temas de muchos trabajos y proyectos de investigación. Algunos ejemplos incluyen el procesamiento de consultas XML [3, 11], la contención de consultas [2], la utilización de vistas materializadas [13], consultas de secuencia [9, 10] y muchos otros.

11. AMPLIACIÓN DE MI MOTOR MINI-DB

Mi objetivo principal en la mejora de mi aplicación de motor de base de datos "mini" es acelerar el procesamiento de consultas. Antes de entrar en los detalles detrás del plan de implementación para lograr este objetivo, echemos un vistazo a qué estructuras de datos, estructuras de archivos y algoritmos están actualmente en su lugar. Sólo se discutirán los más significativos.

11.1 Implementación actual

Estructuras de archivos

- Registro-número-índice ordenado. Este índice es de "bajo nivel" y no es "visto" por los métodos de álgebra relacional (es decir, seleccionar, proyectar, etc.). Es propósito es a proporcionar un primary acceso primario mecanismo de los registros de archivos de datos. También es el "propietario" del estado de cada registro de archivo de datos (es decir, activo o eliminado).

- El índice secundario basado en hash, desordenado, de un solo nivel. Esta estructura de índice proporciona acceso de búsqueda única a los registros de archivos de datos. Dado que no está ordenado, solo se pueden utilizar comparaciones basadas en la igualdad para localizar registros.

- Archivo de datos secuencial y desordenado. Debido al hecho de que los índices actuales están desordenados, este archivo no será capaz de ser puesto en un formato ordenado físicamente, incluso después de una reorganización.

- Archivo de metadatos: Este archivo almacena toda la información, en texto con formato XML estándar, perteneciente a su relación correspondiente. Este incluye: nombrede la tabla; número de fields (atributos); lista de campos con su nombre, tamaño y tipo; identificador de campo de clave principal; identificadores de clave externa; lista de índices con el nombre del índice, el nombre del campo de índice y el tipo de índice (único/clave o agrupado/secundario). Además, el algoritmo de acceso para este archivo es lo suficientemente flexible como para manejar cualquier dato adicional (singular o anidado).

Estructuras de datos

- Cada una de las estructuras de archivos tiene una clase correspondiente con métodos que controlan el acceso al archivo (abrir, cerrar, renombrar, etc.), así como la lectura, actualización, inserción y eliminación de registros.

- Envolver alrededor de las clases de estructura de archivos es la clase DBRelation con métodos que controlan la creación, apertura y actualización de los archivos asociados. Esta clase también tiene métodos de alto nivel que se asociarían con operaciones de relación única como: insertar, eliminar, actualizar y buscar.

- Envolver alrededor de la clase DBRelation es el Mini_Rel_Algebra clase con métodos que realizan las siguientes operaciones de álgebra relacional: select, project, Cartesian product, union, intersection difference y join.

Algoritmos

- DBRelation.Search() Este método realiza la búsqueda real de registros en una relación determinada. Puede aceptar varios campos de búsqueda, condiciones y valores de búsqueda para realizar el equivalente de una selección coyuntar. Los valores de búsqueda pueden ser valores de cadena constantes o referencias al valor de un campo determinado. El búsqueda algoritmo puede funcionar de dos maneras:1. Búsqueda lineal, search donde cada registro en el archivo de datos de la relación es escaneado y comparado con la condición. 2.

  Búsqueda de índices, por igualdad, en un atributo de la lista de condiciones. La búsqueda de índice es esencialmente equivalente al algoritmo de búsqueda S5 detallado en la sección 6.1. Tenga en cuenta que si el registro se encuentra mediante el índice, las condiciones de búsqueda restantes, si las hay, se evalúan para el registro actual. Si no se encuentra el registro, la condición con el atributo indizado es false y no es necesario evaluar las condiciones restantes.

- Mini_Rel_Algebra.Join() La implementación actual de la operación de combinación se realiza ejecutando una operación selecta seguida de una operación de producto cartesiano.

11.2 Mejoras propuestas

Hay 6 mejoras principales de la velocidad de ejecución de consultas que planeo implementar:

1. Implemente un "generador de registros" para que se pueda rellenar un gran número (>100.000) de registros en la base de datos. Esto permitirá realizar comparaciones de rendimiento entre las implementaciones actuales y las nuevas.
2. Reemplace el algoritmo de combinación actual por el algoritmo de combinación hash J4 que se describe en la sección 6.2. Espero un aumento muy significativo en el rendimiento, y, para que se puedan hacer comparaciones de velocidad, mantendré el método de combinación antiguo (renombrarlo a OldJoin). La sintaxis de la nueva llamada Join() será la misma:

   Join(**string** *relationName1*, **string** *relationName2*, **string** *joinField1*, **string** *joinField2*)
3. Cree una nueva clase HJIndex para controlar la estructura de archivos de tabla hash que necesitará el nuevo algoritmo de combinación hash. Dado que la mayoría de los métodos de acceso existentes de la clase DBIndex actual (agregar, eliminar, modificar) probablemente no necesitará ningún cambio (incluso si lo hacen, los cambios serán menores), el HJIndex se heredará de la clase DBIndex.
4. Finalice la implementación del índice del clúster. Esta nueva estructura de índice permitirá recuperar varios registros utilizando los conceptos del algoritmo de búsqueda S5 en la sección 6.1.
5. Modifique el método DBRelation.Search() para utilizar el nuevo índice de clúster.
6. Cree una nueva clase DBQuery. Esto permitirá al usuario crear y ejecutar una consulta que consta de una secuencia de operaciones relacionales. Esta clase será una versión simplificada en que solo controlará una lista secuencial de operaciones. Si el tiempo lo permite, puedo crear una estructura de árbol de consulta "verdadera". El siguientes se implementarán variables y métodos instance de implementedinstancia:

   ArrayList opList : variable de instancia que contiene la lista secuencial de operaciones relacionales.

   DBOperation opObj : Objeto que contiene una operación relacional. DBOperation será una estructura o clase que contiene todos los parámetros posibles implicados en los distintos tipos de operaciones relacionales.

   DBQuery(string queryName) El método constructor.

   bool AddOp(string opName, string param1, string param2, ...) Agrega un operación objeto a el Matriz opList. Se sobrecargará para manejar las diversas listas de parámetros de las diferentes operaciones relacionales. Por ejemplo, una operación de proyecto necesita tres parámetros: string opName, string relationName, string attributeList.

void Clear() Borra la consulta actual: opList.Clear()

string Execute() Ejecuta la consulta actual. La cadena devuelta será el nombre de la relación resultante.

string ToString() Devuelve una cadena de varias líneas de la lista actual de operaciones y sus parámetros.

11.3 Comparaciones de rendimiento de consultas

Si el tiempo lo permite, construiré una base de datos de prueba que consta de varios miles de registros. Esta base de datos de prueba se utilizará para cronopelar las velocidades de ejecución de las consultas "idénticas" en la versión existente y nueva de la aplicación Mini DBEngine. Los resultados de la prueba se compilarán en una tabla de comparación y se incluirán en el informe de la versión final de la aplicación.

Referencias

[1] Henk Ernst Blok, Djoerd Hiemstra y Sunil Choenni, Franciska de Jong, Henk M. Blanken y Peter M.G. Apers. Predicción de la compensación de calidad de costos para las consultas de recuperación de información: Facilitar el diseño de la base de datos y la optimización de consultas. *Actas de la décima conferencia internacional sobre gestión de la información y el conocimiento,*octubre de 2001, Páginas 207-214.

[2] D. Calvanese, G. De Giacomo, M. Lenzerini y M. Y. Vardi. Razonamiento en consultas de ruta de acceso regular. *ACM SIGMOD Record*, Vol. 32, No 4, diciembre de 2003.

[3] Andrew Eisenberg y Jim Melton. Avances en SQL/XML. *ACM SIGMOD Record*, Vol. 33, No 3, septiembre de 2004.

[4] Andrew Eisenberg y Jim Melton. Una mirada temprana a XQuery API for Java™ (XQJ). *ACM SIGMOD Record*, Vol. 33, No 2, junio de 2004.

[5] Ramez Elmasri y Shamkant B. Navathe. Fundamentos de los sistemas de bases de datos, segunda edición. Addison-Wesley Publishing Company, 1994.

[6] Donald Kossmann y Konrad Stocker. Programación dinámica iterativa: una nueva clase de algoritmos de optimización de consultas. *ACM Transactions on Database Systems*, Vol. 25, No. 1, marzo de 2000, Páginas 43-82.

[7] Chiang Lee, Chi-Sheng Shih y Yaw-Huei Chen. Un modelo Graph-theoritic para optimizar las consultas que implican métodos. *The VLDB Journal — The International Journal on Very Large Data Bases*,Vol. 9, número 4, abril de 2001, páginas 327-343.

[8] Hsiao-Fei Liu, Ya-Hui Chang y Kun-Mao Chao. Un algoritmo óptimo para las consultas
Estructuras de árboles y sus aplicaciones en Bioinformática. *ACM SIGMOD Record* Vol. 33, No. 2, junio de 2004.

[9] Reza Sadri, Carlo Zaniolo,Amir Zarkesh y Jafar Adibi. Expresar y optimizar
Consultas de secuencia en sistemas de base de datos. *ACM Transactions on Database Systems*,Vol. 29, Número 2, junio de 2004, Páginas 282-318.

[10] Reza Sadri, Carlo Zaniolo,Amir Zarkesh y Jafar Adibi. Optimización de consultas de secuencia en sistemas de base de datos. En *Actas del vigésimo simposio de ACM SIGMOD-SIGACTSIGART sobre los principios de los sistemas de bases de datos,*mayo de2001, Páginas 71-81.

[11] Thomas Schwentick. Contención de consultas XPath. *ACM SIGMOD Record*, Vol. 33, No. 1, marzo de 2004.

[12] Avi Silbershatz, Hank Korth y S. Sudarshan. *Conceptos del sistema de base de*datos, 4ath edición. McGraw-Hill, 2002.

[13] Dimitri Theodoratos y Wugang Xu. Construcción de espacios de búsqueda para la selección de vista materializada. *Actas del 7o taller internacional de ACM sobre*

almacenamiento de datos y OLAP, noviembre de 2004, páginas 112-121.

[14] Jingren Zhou y Kenneth A. Ross. Almacenamiento en búfer Operaciones de base de datos para instrucciones mejoradas Rendimiento de caché. *Actas de la 2004 Conferencia internacional ACM SIGMOD sobre gestión de datos , junio de*2004, Páginas 191-202.