BASE DE DATOS Procesamiento y optimizacion de consultas

La idea es ver como funciona un SGBD internamente con los datos a partir del modelo relacional. Como resuelve una consulta en un nivel interno. SOLO NOS VAMOS A INTERESAR POR MOVIMIENTOS DE DISCO

Esquema de procesamiento

- SQL Query
- Parser & translator es el que toma la consulta y la convierte en una expresion algebraica
- Relational Algebra Expression es la forma que toma la query, pero no tiene que ser exactamente algebra relacional
- Optimizer es donde el gestor intenta mejorar la eficiencia de la expresion algebraica de acuerdo a la estadisticas que posee el gestor (catalog)- Por ejemplo: Hacer la seleccion antes o despues de X operacion
- Execution Plan es donde ya esta el codigo o algoritmo a ejecutar
- Evaluation Engine
- Query Result

En el caso de PostgreSQL, esta el pg_catalog que es el que contiene toda la informacion de la database. Cada database tiene su pg_catalog
El pg_catalog no se actualiza continuamente. No se actualiza cada vez que se agrega una tupla
Con el comando ANALYZE le digo al gestor que actualice las estadisticas de la database, es decir, que actualice el pg_catalog

Profundizando en el optimizer, vemos que:

- El N(r) son la cantidad de tuplas de la relacion 'r' (en Postgres es el atributo reltuples)
- B(r) es la cantidad de bloques que tiene la relacion 'r' en disco.
- V(a,r) es la cantidad de tuplas distintas del atributo 'a' en 'r'. Se consigue con COUNT (DISTINCT a)
- F(r) es el factor de bloque que determina la cantidad de tuplas de R que entran en un bloque --> F(r)= N(r) / B(r)

B(r) o los bloques de la relacion son los tamaños de bloques de disco
El gestor puede replicar los bloques de disco en memoria para trabajar

Es aqui donde el gestor implementa el uso de **INDICES**

> Un INDICE es una estructura que permite acceder mas rapido a los datos. Puede ser un hash, un arbol, un bitmap, etc..

Nos centraremos en los indices de tipo arbol, mas precisamente veremos que los B-trees o arbol B+ LOS INDICES ESTAN IMPLEMENTADOS EN DISCO, ESTO ES IMPORTANTE. Las hojas de estos arboles guardaran un puntero al bloque de datos.

Los indices, al igual que los datos de la BDD, los puedo replicar en memoria para manejarlos mas rapidos. Por ejemplo, replico la raiz del arbol en memoria para acceder mas rapido a las hojas, replicarlas tambien en memoria y asi acceder mas rapido a los bloques

Con respecto a los indices, tambien se guarda informacion de ellos:

- Height(I(A,R)) es la altura del indice de busqueda I por el atributo A de la relacion R. Si fuera un arbol y nos dice que tiene altura 4, quiere decir que tiene 4 niveles de nodos hijos
- Lenght(I(A,R)) es la cantidad de bloques que ocupan las hojas del indice I
  Nos importara sobre todo la altura del indice

La optimizacion de la consulta se inicia con una expresion en algebra relacional, de esto se obtiene un PLAN DE CONSULTA y luego se materializa para obtener un PLAN DE EJECUCION En el PLAN DE EJECUCION se indican estructuras de datos a usar, indices, algoritmos, etc..

En cuanto al plan de ejecucion, se tiene que evaluar los costos, entre los que se ven el costo de acceso a disco (el mas costoso de todos) costo de procesamiento, costo de uso de memoria, costo de uso de red, etc.. Solo estudiaremos COSTOS DE ACCESO A DISCO El costo nos servira para poder comparar distintos planes de ejecucion y asi elegir uno, por eso es que elegimos el costo de acceso a disco para comparar

Aca cumplen un papel fundamental los indices, que son estructuras de busqueda almacenadas y actualizadas por el SGBD para agilizar la busqueda de registros Pueden implementarse con Arboles o Tablas de hash

Para elegir un indice conviene que tenga una V(A,R) alta

- Indice primario: Es cuando el indice esta construido sobre la clave que ordena un archivo de datos.
- Indice secundario: Es cuando se construye sobre campos que no son los campos de ordenamiento Es decir, no se pone el indice sobre una clave ni tampoco estan ordenadas las tuplas por ese atributo que es indice, por lo que estan desparramadas
- Indice de clustering: Es cuando se construye sobre el campo de ordenamiento del archivo fisico pero este no es clave Es decir, se crea el indice sobre un atributo que esta ordenado pero no es clave. Un cluster es cuando se agrupan las tuplas de acuerdo a un atributo.

Solo puede haber un solo indice primario o de clustering

La mayoria de los gestores tiene su sentencia para crear indices: CREATE [UNIQUE] INDEX nombreIndice ON tabla (a1,...,aN) Los indices sirven para la busqueda y la optimizan, pero a la hora de insertar datos es muy costoso trabajar con indices

---

COSTO DE LOS OPERADORES

SELECCION

Cuanto cuesta hacer una seleccion con una condicion.
Aqui entra el concepto de ***FILE SCAN*** donde se recorre el archivo en busca de los registros que cumplen la condicion Puede ser por:

- Busqueda lineal: Aqui el costo es B(r)

Tambien se puede usar el **INDEX SCAN** (por indice) en donde la busqueda es en base al indice de busqueda

- Busqueda con indice primario: Es decir, se usa el indice basado en la clave. Aqui el costo es

```
Height(I(A,R)) + 1
```

porque tiene que recorrer el arbol para encontrar el bloque y luego se busca el registro

- Busqueda con indice de clustering: Cuando Ai no es clave, pero se tiene un indice de ordenamiento (clustering) por el
  El costo es

```
Height(I(A,R)) + [n(R)/V(Ai,R)*F(R)]
```

Esta formula es igual a

```
Height(I(A,R)) + [B(R)/V(Ai,R)]
```

- Busqueda con indice secundario: El costo es

```
Height(I(A,R)) + [n(R)/V(Ai,R)]
```

Ahora esta el caso en que la seleccion tenga varias condiciones en conjuncion (AND) - Si uno de los atributos tiene un indice asociado, se aplica primero esta condicion y luego se ve las tuplas que cumplen las demas condiciones - Si hay un indice compuesto que involucra mas de un atributo de mas de una condicion, se trabaja con este primero - Si hay indices simples para varios atributos, se trabaja por separado y luego se intersecan resultados

Si tengo una disyuncion (OR) y un atributo no tiene indice, debo buscar por fuerza bruta (FILE SCAN)

PROYECCION (X)

Tomaremos el caso desde dos puntos de vista

> Con X siendo superclave

Aqui no es necesario eliminar duplicados --> costo = B(R)

> Con X no superclave Se deben eliminar los duplicados, entonces hay dos opciones

1. Ordenar la tabla con los duplicados en memoria,
   De lo contrario, usando un sort externo -->

```
costo = 2*B(R)*[logM-1(B(R))] - B(R)           con M = bloques en
memoria
```

2. Con hash haciendolo en memoria -->

```
costo = B(R).
```

Con hash externo

```
B(R)+2 * B(proyeccionConDuplicados(R))
```

Si la consulta no incluye un DISTINCT, el costo siempre es B(R).
Si se hace en pipeline con otra operacion ademas de la PROYECCION, el B(R) = 0

Pipeline: Cuando una consulta esta anidada con otra (por ejemplo, una proyeccion luego de una seleccion) se puede eliminar el costo de la segunda Esto es porque a medida que procesa las tuplas la primera, se los puedo pasar a la segunda sin necesidad de mandarlos a disco previamente

JUNTA

Tiene varios metodos para calcularla:

> loops anidados por bloque

Este metodo consiste en tomar todas las tuplas de dos relaciones R y S y compararlas entre si
El costo cuando tenga infinita memoria es B(R) + B(S)
Como minimo tengo que tener 2 bloques de memoria (M=2) para cargar br+bs
El costo si no tuviese memoria infinita es B(R) + B(R)*B(S)
Esto vale tambien para el producto cartesiano

> unico loop

Si el atributo de junta tiene un indice para una de las tablas de las relaciones, entonces recorro en la otra relacion una por una las tuplas y las comparo buscando con el indice de la relacion que tiene indice
Si el indice es primario --> costo: B(S) + n(S) * (Height(A,R) + 1)
Si el indice es de clustering --> costo: B(S) + n(S) * (Height(A,R) + n(R)/ V(A,R) * F(R))
Si el indice es secundario --> costo: B(S) + n(S) * (Height(A,R) + n(R)/ V(A,R))

> sort-merge

Consiste en ordenar los archivos de cada tabla por los atributos de junta
Si entran en memoria, el costo de acceso a disco es solo B(R) + B(S)
Si los archivos no entran en memoria, se deben ordenar las dos relaciones y hacer luego un merge
El costo es --> B(R) + B(S) + 2·B(R)·[logM−1 (B(R))] + 2·B(S)·[logM−1 (B(S))]

> junta hash (variante GRACE)

La idea es poder usar un hash cuando no me entran los archivos de datos en disco y no tengo un indice.
La variante Grace usa el disco para las operaciones.
k es la cantidad de particiones. h(r*A) -> [0, k-1]

Propone entonces partir la tabla R y S en k tablas en disco.
Costo del particionado:

```
2*(B(R) + B(S))
```

Costo total:

```
3 * (B(R) + B(S))
```

ESTIMACION DE LA CARDINALIDAD

Como parte del costo es necesario estimar el tamaño de las tablas intermedias antes de calcularlas La estimacion tiene que ser precisa, facil de calcular y no dependa de la forma de calcularla Lo veremos para

- PROYECCION Siempre va a ser una fraccion de B(R) Suponiendo que cada tupla ocupa 24 bytes y quiero proyectar el DNI que ocupa 4 bytes Entonces B(R) / (4/24) En cuanto a la cantidad de tuplas, sigo teniendo las mismas. Es decir, cambia el factor de bloque (B(R)) pero no cambia el numero de tuplas (n(R))
- SELECCION La seleccion reduce el numero de tuplas aunque respeta su tamaño La estimacion entonces es n(R) / V(Ai,R)
- JUNTA

-----HEURISTICAS DE OPTIMIZACION---------- Realizar lo antes posible las selecciones Reemplazar productos cartesianos por juntas siempre que sea posible Proyectar para descartar los atributos no utilizados Si hay varias juntas, realizar primero la mas restrictiva

---

PRACTICA

Costo de seleccion : b(r)

Indices Arbol B+

- Nodos internos: Tienen un valor y un puntero que indica un nodo hoja con los indices.
- Nodos hoja: Asi, cada nodo hoja, tiene una seria de valores que siguen una regla (ej: los padrones entre 61000 y 62000). Estos, a su vez, tienen el puntero a nodo de datos que contienen los valores de la base de datos.

Los arboles pueden tener varios niveles.

Costo de busqueda por igualdad:

- Altura de indice
- Acceso a los datos de las filas devueltas (estimacion n(R) / V(Ai,R))

Indices cluster: no cluster:

```
simples: Formados por un solo atributo
compuestos: Cuando esta formado por dos o mas atributos
```