

Almacenamiento de Datos en Sistemas Informáticos

Tipos de Almacenamiento

- **Medios físicos clasificados según:**
 - * Velocidad de acceso
 - * Coste de adquisición
 - * Fiabilidad

Principales Medios de Almacenamiento

- Memoria Flash

- * Utilizada en dispositivos electrónicos digitales
- * Capacidades desde 256 MB a varios GB
- * Bajo coste

- Discos Magnéticos

- * Principal medio de almacenamiento persistente
- * Guardan bases de datos completas
- * Requieren transferir datos desde disco a memoria

- Discos Ópticos

- * CD-ROM y DVD-ROM
- * Almacenamiento de datos pregrabados
- * Versiones grabables como CD-R y DVD-R

- Almacenamiento en Cinta

- * Usado para copias de seguridad y archivado
- * Acceso secuencial
- * Más lento que otros medios

Consideraciones Técnicas

- Jerarquía de Almacenamiento

- * Niveles superiores: rápidos pero costosos
- * Niveles inferiores: más económicos pero más lentos

- Volatilidad

- * Almacenamientos volátiles pierden datos sin energía eléctrica
- * Sistemas por debajo de memoria principal son no volátiles

Tecnologías Avanzadas

- RAID (Redundant Array of Independent Disks)

- * Mejora velocidad y fiabilidad
- * Diferentes niveles de implementación
- * Protección contra fallos de disco

Tendencias

- Aumento continuo de capacidad de almacenamiento
- Reducción de costes
- Mayor velocidad de transferencia de datos

Sql es un lenguaje de 4ta generación, solo se indica cual es el resultado que se requiere. SQL distinto a plsql

Yo quiero que todas aquellas combinaciones de el nombre de alumno con una materia con un valor de calificación para un particular de la condición de alumno se supone que por detrás un lenguaje un lenguaje que especifica un resultado software interpreta esa consulta identificar es el procedimiento procedimiento

Como puedo especificar el procedimiento que se debe seguir? la consulta debe estar correctamente escrita. Análisis sintáctico, si esta correctamente escrita desde el punto de vista del lenguaje.

Select * o select y un conjunto de identificadores y luego una palabra reservada y luego un identificador, la consulta podría quedar ahí o continuar continuar identificadoUr hasta que termina entonces yo tácticamente la consulta también puedo validar consulta qué significa validar

Semántica es decir algo que se entienda.

Una vez verificada la semántica

Esta consulta se debe evaluar. Como se evalúa?

Y luego computar, entonces el resultado se puede producir.

Como se implementa un join? Es un producto cartesiano. Como se evalúa? Doble flor, bucle anidado

Como hago una proyección?

Copio datos que necesito proyectar en filas nuevas, y ese resultado lo guardo o retorno. Guardo para volver a procesar si sigo evaluando, o retorno porque es la última expresión y es el resultado

Que tengo que tener en cuenta para implementar operaciones que resuelvan una consulta? Como están almacenada la información. Ya conozco el algoritmo para implementar, pero que mas necesito saber? Como esta implementado el objeto llamado tabla y el objeto llamado fila.

En todo momento operamos o con tablas o con filas en una base de datos.

Lógicamente hablando: Dentro del diseño lógico de una base de datos particular puedo tener una tabla alumno, tiene columnas cada una con tipo de dato asociado o definido.

Físicamente hablando: la base de datos esta en un directorio del disco /var/lib/academicos/alumnos.dbf

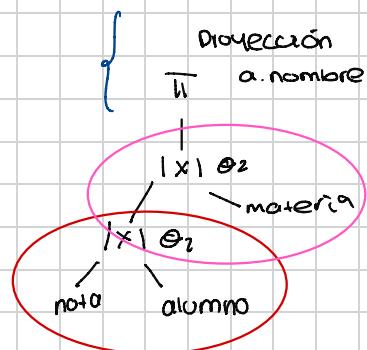
Lo primero que tenemos que saber que lógicamente son estructuras, pero físicamente están mapeados en un almacenamiento en forma de archivo, puede ser texto plano o binario. Hay algunos que mapean en un único archivo enorme, pero ese mismo archivo enorme se subdividen en distintos objetos. Datafile o varios datafiles, cual es la diferencia? Se traduce en optimizaciones que se pueden o no aplicar, también mayor o menor prestación para indexar los datos.

Por que la base de datos puede estar en formas diferentes? Es específico de tecnologías o enfoques aplicados para cada base de datos

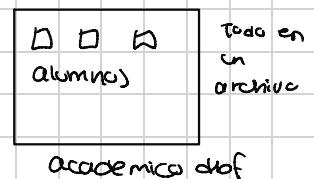
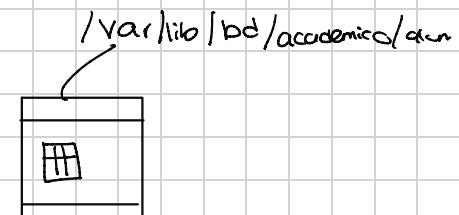
Que son los objetos que se crean en la BD? Tablas, procedimientos, indices, paquetes de desarrollo, objetos binarios con el archivo grande uno implementa algoritmos que no necesariamente estén implementado en sistemas operativos, esto con Oracle

Postgres es lo contrario, mapea como una carpeta y esos objetos son mapeados como archivos. Hay archivos para cada partición, el mismo no necesita manejo adicional del modelo de datos, se implementa un modelo de datos específicos del sistema operativo instalado.

Algebra Relacional



```
for ( fn en tabla nota ) {  
    for ( fm en tabla alumno ) {  
        if ( [fn, fm] ⇒ Ø1 ) {  
            R = R + [fn, fm]  
        }  
    }  
}  
return R;
```



La base de datos las vemos como tablas e información de manera abstracta con sql, lo que ocurre es que se traduce en una expresión sql que se traduce y esta se hace sobre archivos.

Niveles de almacenamiento disponible en un computador: cache, ram, unidades flash, unidades magnéticas, unidades ópticas.

El costo de implementar es mayor en la medida que el medio de almacenamiento es especializado.

Comprar una licencia que funciona en la ram o cache es mas caro que una que funciona en almacenamiento secundario normal. La prestación de capacidad también varía, en cache es mucho menor que en una magnética.

De que depende que unidad uso para implementar mi base de datos? Velocidad de acceso (depende de la necesidad y del ??), volumen de datos (depende del propósito)

Uno de propósito general no tiene sentido poner en un especializado, pero si necesito recuperar rápidamente porque necesito atender solicitudes lo antes posible uso los más rápidos

Por ejemplo si tiene que responder lo antes posible como spotify va a ser en la RAM o llamadas

Un sistema de navegación que se conduce solo, hay que tratar de tener un tiempo de respuesta apropiado para el propósito.

Al momento de diseñar una base de datos hay que entender su propósito, donde voy a desplegar/installar o depolar la base de datos.

Explicación interna de como funcionan los discos magnéticos en la 4ta edición

Una unidad de disco magnético es un aparato electromecánico, componentes electrónicos y mecánicos, los electrónicos son unidad de control que implementan un firm ware que implementa hacia el computador un protocolo de comunicación (CPU y memoria) por ejemplo SATA es un protocolo de comunicación. La parte electrónica es como se comunica CON el disco, la mecánica es COMO se guarda la información peine es la parte ?? De un disco magnético que tiene montado sobre si mismo cabezales que sirve para la lectura y escritura, responsables de escribir en un disco y de leer donde los discos están superpuestos uno sobre otros sobre un eje.

Por ejemplo: el tiempo de acceso en una unidad de disco es igual a tiempo de búsqueda (posicionarse sobre la pista) + tiempo de latencia rotacional

Elijiendo bien las tecnologías y el medio donde almacenar, encontraremos buena configuración de hys para aproximar un mejor rendimiento posible.

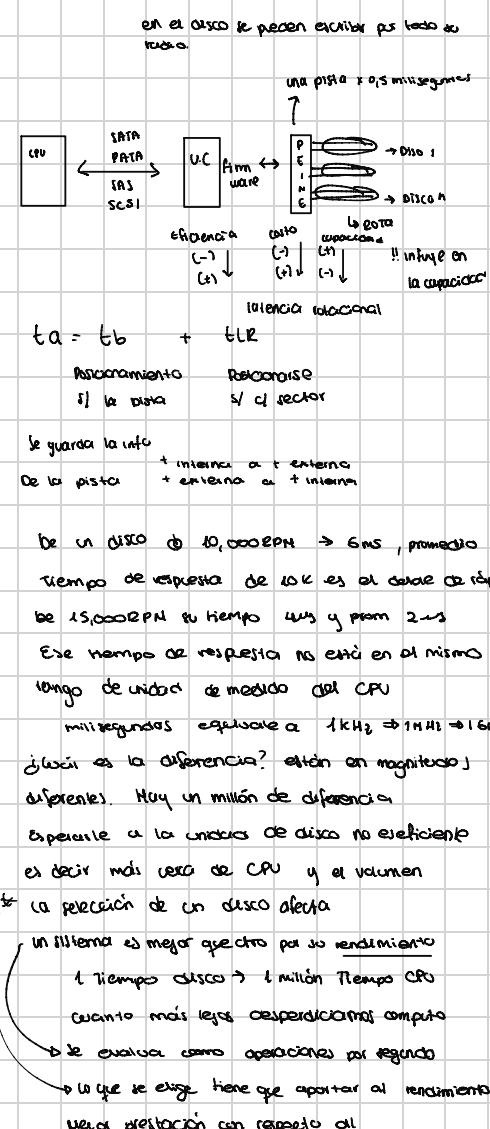
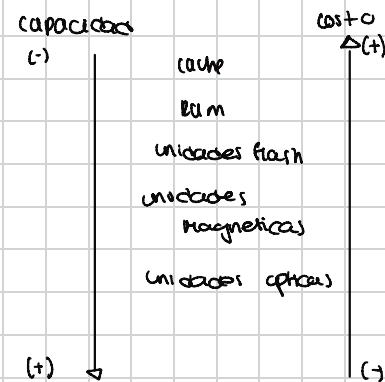
También tengo que elegir un servidor, luego tengo que tener en cuenta como implementar los objetos

Al diseñar la base de datos hay que entender que es un diseño lógico el create table, y eso se debe mapear en un archivo.

Como se mapea una tabla en archivos?

Un archivo es un espacio físico que inicia con un sector 1 y termina en un sector n, normalmente empieza en un bloque 1 y termina en un bloque n, normalmente tienen cierto tamaño que son definido por el sistema de archivo.

Todo depende del formato del sistema de archivo. El mínimo espacio que puede ocupar es 4k para que exista en el medio de almacenamiento como unidad única es ocupando un Xk dependiendo de como esté formateándose el sistema de archivos.



Oracle permite definir que esa tabla se vaya en un bloque de 2k, se pueden definir particularidades físicas, cosas específicas de oracle block size 2k... Cuando hay que mapear una tabla en un archivo existen 4 formas conocidas,

1. **Mapeamiento basado en heap:** el registro, la fila que se quiere agregar se agrega en cualquier parte que haya espacio. La fila que se agrega se agrega en cualquier parte que haya espacio para ese bloque. Normalmente donde no hay clave primaria se usa eso
2. **Mediante organización secuencial:** los registros se guardan uno detrás de otro precautelando la clave primaria, significa que cuando se van insertando se van ordenando para mantener la clave primaria, se pueden reorganizar, tras cada inserción para resguardar eso los bloques se usan hasta un 70% de tal manera que queden un 30% y así pueda guardar en ese bloque si viene otro registro, no se usan nunca al 100%
3. **Hash:** la mayor cantidad de operaciones o mas frecuentes van a ser selecciones en un atributo único, entonces puedo decir que quiero guardar en hash según la clave id, hacer una función hash que se mapea con un registro con números del 1 al n?? Cuando guardo una fila le decís que se tiene que guardar según el resultado de la función hash, al hacer select se puede usar.
4. **En agrupación:** guardar según una correspondencia que haya en esa tabla, si quiero guardar 3 tablas en un archivo, para eso en un bloque bi que tengo la tabla alumno. Capaz es mejor hacer una lectura en forma secuencial y me quedo con una fila que me interese, hago eso porque de esa forma no tengo que combinar la información (eso implica leer diferentes archivos, que implica trabajar en diferentes sectores del disco y pagar latencia rotacional)

Vista: resultado precalculado de una consulta, usado para abstraer una consulta, la encapsula bajo un alias

Es una tabla virtual a menos que sea materializada

Que se calcula la consulta y se guarda en un archivo,

Como lo guardaría en ese caso? Me conviene guardar en forma lineal.

El siguiente paso es como mapear un registro (una fila o una Tupla) en un bloque?

1. **Registro de longitud fija:** evaluó su tamaño máximo y reservó tantos lugares como el bloque permita. Y voy mapeando. Guardan el máximo tamaño, pensando que guarda eficazmente, orden óptimo pero usa más recursos que debería
2. **Registros de longitud variable:** hago que cada fila ocupe un registro según la información que guarda el registro. Pero si crece hay un problema entonces hay que llevar a un bloque de desbordamiento hasta que pueda reorganizar la tabla (leer toda la tabla del archivo original y reordenar todos los bloques) es mas eficiente, el uso de los recursos es más eficiente. Se usan estructura de pagina (bloques) c/ ranuras.

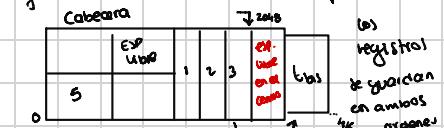
			o 500
íd	nom	ape	

¿Qué pasa con el registro?

long máxima:

long mínima:

Cómo guarda registros que tiene tamaño q se cambia tanto en un bloque de disco



el anterior en posición 5 me dice donde está la tupla del bloque i que se insertó en ?? posición

el bloque se consume de atrás para adelante

la mayoría utiliza esta estructura para guardar en los bloques

Capítulo 12

Como resolver la consulta? Un for con una condicional, búsqueda secuencial o lineal. En el medio hay álgebra relacional, donde si esta el algoritmo.

La tabla cuando se guarda en el disco se guarda por solo una columna, si quiero guardar por otro debo llevar a memoria, duplicar, reordenar (reescribo?)

Si quiero usar búsqueda binaria entonces debo buscar por la columna por la cual esta ordenado mi archivo.

- La consulta es en función al where.
- Los archivos se procesan de a bloques.

Que puedo hacer para mejorar? Indexar la tabla. Como la consulta demora mucho necesito crear index alumno_apellido_idx no Alumno(apellido)

Que es un indice? Un indice es un objeto de la BD que puede ayudar a optimizar el tiempo de recuperación de la información. El indice es un archivo ordenado en el que los registros del indice tienen dos componentes. Uno de los componentes es un valor clave de búsqueda y el otro es un puntero.

Como mi where es en función del apellido necesito recuperar las filas que tengan el apellido González. Si hacemos búsqueda lineal tenemos $O(n)$.

Búsqueda binaria no se puede porque no esta ordenado.

El archivo idx tendrá un valor de búsqueda "González" y un puntero a un cajón de punteros a la/s fila/s con apellido González. Ahora el coste $O(n)$ será la cantidad de bloques del indice, que son los valores únicos de los apellidos.

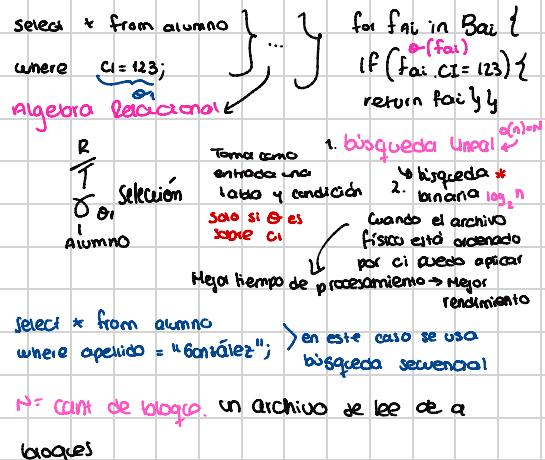
El indice es obligatoriamente un archivo secuencial ordenado, por lo que tiene que estar ordenado por apellido, para realizar una búsqueda binaria, por lo que el O de acceso del indice será \log_2 de la cantidad de bloques del indice

Es necesario que tabla alumno este ordenado? No, solo el archivo del indice, tiene que ser completo, si hay una no indexada no se va a encontrar y o se supone que no existe o se hace búsqueda lineal (full scan), el otro es binario search scan, y el que vemos ahora index scan.

Si quiero buscar eficientemente por nombre u otro, debo indexar por esa columna. Es correcto indexar todas las columnas? La búsqueda va a ser eficiente, pero el problema es al insertar el dato, hay que actualizar todos los indices donde esa nueva fila tiene impacto, la inserción con complejidad I ahora tiene complejidad $??$, se mejoraría búsqueda para empeorar actualizaciones y/o inserción. Que pasa con la modificación? Costo n , necesito modificar el archivo de tabla mas el de todos los indices.

Se crean los indices necesarios, todo depende de la aplicación que se está diseñando. Lo que se debe evaluar es entre todas las consultas ver los where para de todos esos determinar las que normalmente se hace búsqueda, ademas de ver todas las consultas que normalmente se ejecutan sobre la BD, también debo evaluar la frecuencia con la que se ejecutan esas consultas, las mas frecuentemente son las que necesitan un indice, las que no son tan frecuentes no es necesario sobrecargar las inserciones o actualizaciones, es un tradeoff

Hola



Eso se traduce en una operación de reunión entre venta y cliente, dada por una condición Tita y produce un resultado.

Que aspectos tiene que tener en cuenta el motor? El motor de base de datos necesita tener en cuenta recursos de CPU, + recursos de CPU + memoria.

Del CPU se recibe un cierto tiempo, y el proceso es dueño de la CPU durante ese tiempo. Puede usar toda la CPU durante un espacio de tiempo. **Cuanta memoria recibe?** Virtualmente un proceso es dueño de toda la memoria, pero en realidad puede usar tanta memoria como la memoria libre. Se ejecutan con la memoria disponible hasta un tope configurado. Me llegan 100 consultas, 20 lanza como proceso concurrente, y 80 quedan en colas. El proceso transmite la memoria, libera y luego muestra. Cada motor tiene variables de configuración para determinar la cantidad máximas de hilos o consultas en paralelo, cantidad máxima de memoria para hilos, cantidad máxima de consultas del servidor principal, cantidad de archivos. Según eso se afina el comportamiento. Que tipo de gestión de memoria debe hacer.

2 tipos de gestión de la memoria que puede hacer el sistema operativo

MRU: se basa en tratar de usar el bloque más reciente utilizado.

LRU: el bloque menos recientemente utilizado.

Si uso MRU, el tiempo $O(n) = N_{fV} * B_{rC} + B_{rV}$, por cada fila de venta levantando los 10 bloques, y en algún momento se procesan todos los bloques de venta $(20.000 * 10 + 20) = 200.020$ bloques procesados.

Si queremos editar el archivo de configuración, en el editor de texto

`$> vi dms.conf`

buscamos `mom_mgmt=MRU`

`$> sobrecarga dbms restart`

La memoria del proceso va a estar vacía. Comenzamos por B_{rV} , se toma el primer bloque y mientras tenga espacio en la memoria va levantando B_{rC} hasta que se quede sin espacio, y el B_{rC} es el MRU, el que se acaba de terminar y continua con el ciclo 10, se cambia ese bloque por B_{rC} , se procesa la fila con respecto a todos los bloques clientes, el bloque externo pasa a la segunda fila y se reinicia el `for` externo, empieza por el B_{rC} , el cual ya está en la memoria.

De donde traigo los bloques? Del archivo de la tabla cliente, el archivo de la tabla venta tiene 20 bloques, por ej.

El resultado se transmite fuera de la base de datos al aplicativo?????

$O(n) \text{MRU} = n_{fV} * (B_{rC} - M) + B_{rV} = 2000 * (10 - 10 + 1) + 20 = 2.020$ bloques procesados

Es uno porque la es lo que no entra en la memoria, es la cantidad que bloques que no me entra

Ambos con complejidad cuadrática, pero uno de los algoritmos tiene cierta ventaja, pero uno hace una estrategia más inteligente. Si cada bloque toma 10 milisegundos, cual algoritmo de uso de memoria es mejor? Y en todos los casos se prefiere aquel algoritmo que procese la menor cantidad de bloques. Por qué?

1. Porque es el que menos tarda en ejecutarse y por eso se maximiza el rendimiento
2. Pero la operación es mas pesada con respecto al CPU, el tiempo de CP es el que menos se desperdicia. Las operaciones de entrada y salida desperdician tiempo de CPU

```
Select v.nro, v.fecha, v.monto, c.nombre
from venta
join cliente c on c.id=v.cliente
for fvr in Bv1    Producto consumo
for frc in Bc1
if ([fv1,fc1]==0) then b=b+[fv1,fc1]
je toma la primera fila y se
compara con una fila cliente
```

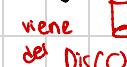
$$B_{rV} = 20$$

$$B_{rC} = 10$$

M = 10 bloques

B _{rV}	B _{rC}		

¿Qué bloque de carga viene primero?



Mientras haya lugar en la memoria se carga un siguiente bloque

Y si la memoria se llena, ya procese q y me queda bloque 10 de tercera cliente y debo liberar la memoria ¿Cuál?

Menos reciente: ya leí todo y no leeré de vuelta

Más reciente: el último

Si uso LRU devuelvo el bloque 1 y cargo el 10 y termina mi join. Encuentré en el último bloque.

Si encontraba carros pasaba a otro Bv1

cuando paso a la segunda fila recibo BC1 y se cambia BC2 por BC1, y así hasta que el menos reciente sea BC10 por BC9 y ese puntero va corriendo a lo largo de la memoria

La métrica dominante: cantidad de operaciones de entrada y salida en relación a las consultas que procesa, a + consultas - entrada y salida el mejor

Buffer manager: el encargado de escribir

Catálogo del sistema: base de datos guarda datos de si misma. La información

Lógica se guardan en tablas propias de la base de datos. TAREA!!

LEER ESQUEMA DE LA BASE DE DATOS

Redundancia por propósito de recuperamiento de información.

Técnica defecto es RAID significa redundant array of disk

RAID 0 no tiene redundancia, agrupa discos y maneja como una sola unidad, divide la información en n partes guardado en discos, discos usado en paralelo y tiempo en n. La perdida de un disco es una catástrofe.

RAID 1: se parea una unidad con otra, el principal y el espejo. Todo lo que entra en el principal se duplica en el espejo, varios pares y se divide en n pares. Permite que si hay n discos hay un rendimiento hiperescalar de $n/2$ porque la mitad es para dividir y la mitad para espejar, mitad escalar mitad redundancia

Cada par redundante es independiente puedo tener fallas en diferentes pares y entrar los de respaldo, ademas la reparación es sin interferencia, el par entra como respaldo, el original se arregla, al mismo tiempo que restauro puedo seguir usando el respaldo con perdida de rendimiento pero puedo parallelizar.

RAID 5: a partir de 3 o mas discos, se guarda información en el disco 1 y se guarda eso o respalda en el i-1, en el disco 2 se respalda en el disco 1, disco 0 en el disco n, fallos de hasta un disco y teniendo los n-1 con fina de paridad recuperar el disco perdido y hago el resto re de hasta un disco al mismo tiempo

RAID 6: redundancia de doble paridad casi igual a la de paridad 5 solo que en el sector de paridad se guarda doble paridad. En el i-1 e i-2, hasta fallos de 2 discos simultáneamente. Disco 0 se respalda en el n y n-1.

RAID 10: combinación de raid 0 y raid 1, si tengo 4 unidades de almacenamiento puedo agarrar dos e implementar un raid 0, toda la información que entra se divide en dos discos i/2, se duplica velocidad de escritura y lectura, se sacrifica redundancia. Los otros dos discos también pueden ser raid 0, toda la información 1 se divide entre dos, al nivel mas alto en vez de dividir, se duplica, entonces la información i entra en ambos pares, virtualmente se implementa un raid 1

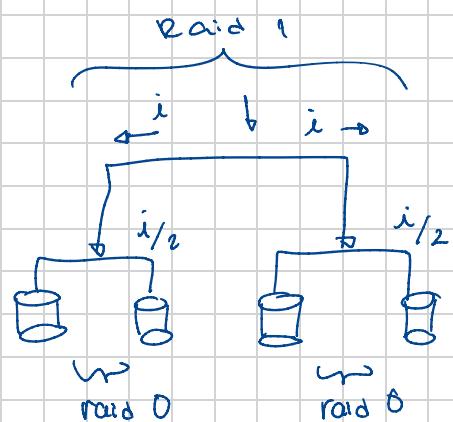
Un tipo de optimización lógica son los índices.

Archivos secuenciales ordenados, cada registro es un par clave puntero, secuencialmente ordenados en bloques, los bloques también secuencialmente ordenados en archivos bloques, lo importante es escoger el el...

Se pueden clasificar, pero si la tabla se ordena por la clave. El orden físico coincide con el orden de su tabla. Tiene ventaja de dejar de indexar todos los valores, ademas de ser índice primario es un índice denso, todos los valores diferentes de la tabla están indexados en el índice.

Es más rápido procesar el índice mas que la tabla, un primary key no se puede repetir, es más fácil buscar en el archivo chiquito si algo se repite que en mas grande, otra función que es mas apropiada en la mini tabla es más fácil hacer un count en la tabla chica, clave primaria es un índice denso, las filas son únicas, que sea denso significa que todos los únicos aparecen en la mini tabla, puedo verificar la duplicidad y contar, siempre van a coincidir la cantidad. Control de duplicado, búsqueda en rango, contar la cantidad.

Solo se puede tener una clave primaria, en una única columna se tienen esas ventajas, a menos que se indexen otras columnas, lastimosamente los índices no van a coincidir con los índices de la tabla, son llamados secundarios y **NECESARIAMENTE DENSO**, para todos los valores de saldo si o si deben estar indexados



Los indices secundarios son array de array, array de puntero que apunta a array de cajón de puntero, llamado cajón porque se puede guardar varios, son estructuras implementadas en discos, en mi bloque siempre entran muchos punteros, en bloque indice son cajones de registro.

Algunos cajones de puntero tiene un solo puntero, otros cajones tienen mas de un puntero que apuntan a 3 posiciones diferentes de la tabla porque aparece 3 veces.

Si se inserta una nueva fila, cuando se quiere indexar se ve si es que ya esta indexado, se sigue el camino al cajón y se agrega el puntero. **Por que siempre el archivo esta ordenado?** Para aplicar búsqueda binaria, para que la búsqueda sea optima.

Indices densos: primarios, ordenados. Los secundarios son necesariamente densos.

Indice disperso: el primario puede serlo. Si son dispersos el archivo indice es disperso, conviene cuando mis datos están ordenados en relación a la misma clave de búsqueda que se usa para construir el ???, y cuando los datos corresponde a bloques de discos distintos.

Indices muy grande: primer indice denso y los demás dispersos, cuando hay mas de un nivel se parece a un árbol

Una manera de implementar los indices es con arboles B+.

Archivos indexados: archivos ordenados, registros ordenados de acuerdo a un valor clave.

Por que árbol b: balanceado, cualquier camino de raíz a hoja tiene la misma longitud, siempre están completos. Todos los caminos de raíz a nodo hoja misma longitud, cada nodo interno que no es raíz tiene $n/2$ hijos (toda esa parte) siempre están medio lleno o medio vacío. Lleno de la mitad para arriba, lo bueno es que la utilización es mayor a 50 y encontrar el lugar apropiado para mantener la información, resulta estructura apropiada para indexación en motores de HD siempre que los nodos sean bloques.

Como implemento un árbol b+? El nodo de disco se divide como un bloque de longitud fija, cada registro es puntero clave. El ultimo es puntero. Si puedo llenar un nodo de esa forma se puede convertir en un nodo de árbol B. (Leer toda esa parte)

Los nodos internos tienen la misma forma nodo hoja, siempre tienen nodos internos hijos u otras hojas pero no se encadenan entre si.

Todos los valores mayores están a su derecha, los menores en izquierda.

Un árbol b es + cuando los valores indexados siempre están en hojas y los internos son para búsqueda o direccionamiento. Info útil en la hoja, en los internos información de búsqueda. El ultimo nivel se comporta como un indice de un solo nivel. A mas clave por nodo es mas bajito.

Se pueden crear indices en B, la info no se repite en las hojas, distribuida al nivel de todo el árbol, la carga útil. Los nodos internos tiene que tener doble puntero, uno para ir a donde esta en la tabla y el otro para seguir explorando, los nodos hojas si son iguales. Ventaja: es mas compacto, menos nodos para la misma cantidad de información. Utiliza menos espacio. Desventaja: requiere mas esfuerzo de mantenimiento, indice cambia más rápido y requiere mas esfuerzo.

Una tabla por has, créate table pero antes de terminar específico que quiero que sea hash, complemento el DDL con info adicional para que sea como hash.

La función hash tiene 10 valores de hash diferentes en el ejemplo, la clave es nombre sucursal? O sea si tienen mismo hash hay colisión

create table cuenta (

num_cuenta varchar[10] pk,

nombre_sucursal varchar[50] nr,

saldo integer nr,

constraint ...

) using hash (nombre_suc, f)

severicial

in memoria

Esperamos que la función hash sea uniforme y aleatoria, no se puede predecir en que grupo cae en donde cae, pero cuando todos los grupos tienen el mismo tamaño.

Como se crea un indice organizado por hash es un indice asociativo, no es ordenado. No se le dice la columna porque los indices ya tienen una clave con la que se construye, ya se hashea por la misma clave. Para la tabla si se necesita especificar.

Como es el archivo indice de carrera? Si se le dice que use hash entonces va a ser un archivo hash subdividido en bloques internos, cajones de dispersión, la BD elige en su librería una función que hashea de 1 al 5, que hay en cada cajón? (Si es tabla serán filas) si es indices hay registros indices (como son: clave, puntero)

Como se indexan? En que momento se indexan? Al insertar o al crear el indice, una búsqueda lineal.

Si no existe se agrega en el bloque correspondiente de su hash, el puntero, como el valor es repetible no se apunta directamente, se apunta a un cajón de punteros auxiliar donde se tienen muchos punteros, en el primer bloque se pone Bi. O en el primer lugar vacío si es que ya no se encuentra.

Archivo indice organizado por hash que puede indexar una tabla en cualquier tipo de organización, nos obliga que el archivo indice sea un hash, necesariamente es indice secundario si tiene cajón de punteros.

Normalmente el cajón de puntero se guardan ordenados.

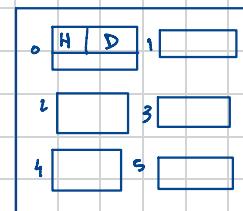
Ninguna fila se guarda como puntero, se guardan el bloque.

Como se optimiza todas las filas que me lleva a los alumnos con lIN? Se analiza semánticamente, y anotas que tener que usar index_scan, tomas valor clave, has es hash y buscas su entrada (Esto ya es en index_scan prácticamente), como es indice secundario y encuentro el cajón de puntero, me dice que bloques hay que explorar. Ya se que todas las filas que tengan esa carrera están en esos bloques, se explora la memoria que fue cargada por el buffer y consulto, consulto la cantidad indexada, en caso promedio no es ni 0 ni n, es $n/2$, depende de si mi función hash es aleatoria.

Indice ordenado son de propósito general, los indices hash solo busqueda en igualdad o aplicación en reportes

Create index idx_Alumno_Carrera
on Alumno(carrera) using hash

B-Trie ← como archivo hash



Problema de la asociación estática: si hay muchas colisiones estas pueden generar una lista desordenada, a la larga si hay muchas colisiones el tiempo se degenera a una búsqueda lineal.

Si elegimos una función has aleatoria y uniforme nunca va a pasar eso.

Uniforme todo el universo se dispersa equitativamente, aleatoria no importa el patrón, cuando proceso todos los cajones reciben equitativamente en teoría valores de dispersión.

El problema es el espacio, la asociación tiene que reservar todo el espacio de una. Un cajón puede ser mil o un bloque de disco, todo cajón que no es utilizado es un overheap, o es una sobrecarga, desperdicio. Hay cajones que pueden estar subutilizados.

Pasa esto con los índices ordenados? No, porque se ajusta, la estructura de datos se ajusta, una de árbol B se ajusta.

Si tengo mi tabla secuencialmente ordenada, y el índice has con. Cajones que indexa numero sucursal, por ejemplo en el cajón 3 se desbordó (ver imagen ejemplo).

Lo ideal es tener un esquema de asociación estática que reserva espacio en la medida que se necesita, no que sea a priori y se desperdicie, sería bueno que la cantidad aumente o disminuya según aumenta o no la cantidad de claves que estoy procesando. Si necesito ajustar el tamaño que usa el índice, significa que el numero de cajones debe ser ajustable, cantidad de cajones modificada a lo largo del tiempo.

Tabla has que se auto adapta en cantidad que opera a lo largo del tiempo a lo largo del espacio que necesita para ser implementada, se llama asociación extensible.

Cuando tengo un valor has no necesariamente tomo todo el valor, necesito el prefijo para direccionar, si tengo 4 cajones necesito un prefijo de 2 bits, si tengo dos cajones un bit, 0 para el primero y 1 para el segundo. Si tengo un solo cajones tengo 2 a la cero, cero bits. (Leer esa parte)

Que pasa cuando divido un cajón? Donde ocurre normalmente? En los arboles B, en realidad es una abstracción de los arboles, es en los arboles donde si tengo que aumentar espacio divido o combino los nodos.

Los valores has pueden tener 32 bits pero no los considero todos, solo los que necesito. La cantidad de cajones es mayor a 2 a la 1-1

Como son los registros índice? Clave puntero y a eso apunta a la tabla que esta siendo indexada, si la clave es única que hacen los punteros? Apunta directamente a la fila, si la clave se duplica se necesita un cajón auxiliar de punteros, dependiendo del tipo de índice se convierte primario y apunta a los datos, o apunta a un cajón.

Cuál es la consulta DDL (definición de datos, indica que crea, elimina o modifica los datos) que crea ese índice?

Cuando indexo, indexo valores únicos. El algoritmo de inserción hace full scan, actualmente no hay valores en la tabla por lo cual es necesario recorrer todo.

En este caso necesito cajón de punteros porque la clave de búsqueda se puede repetir.

Cuando organizo la tabla como si fuera hash, si importan los duplicados, porque usa la fila, y esa si es única, por mas de que dos filas caigan en un mismo cajón esas mismas son independientes entre si.

Un cajón lleno implica que el índice hash tiene que crecer. Se mira primero cuántos bits tiene el cajón que se llena y se suma uno, para que? Para dividirse, si a su prefijo le sumo un bit más divido.

```
create index <...>
on cuenta (nombre_suc)
use extendable hash
```

Cuando un cajón aumenta el numero de bits, su nuevo cajón tiene el mismo numero de bits y ademas de eso se aumenta los tamaños de bits de la tabla, si a cero le sumo 1, entonces queda 1, la tabla tiene que alcanzar el tamaño del prefijo. El cajón original tiene un gemelo/dividido que tambien tiene tamaño 1, en el original quedaron todos los que tenian prefijo 0 y en el otro se mudan todos los] que tienen prefijo 1, la tabla tambien crecio porque tenemos prefijo 0 y 1.

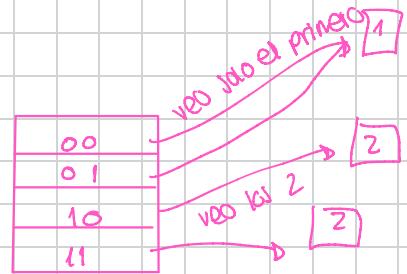
Al estar el cajón lleno, necesito dividir el cajón y +1 y relocalizar los elementos, los que tienen nuevo bit 1 se mudan, el cajón tambien pasa a ser bit 2, y si el cajón supera la tabla, la tabla se iguala al cajón. No necesariamente el primer cajón se ve superado si es que no esta lleno. Al dividirse la tabla se copian uno a uno los punteros ?? se duplica

Cuando se duplica el tercer perro que pasa? Esta lleno y todos tienen el mismo valor hash, puedo dividir y considerar un mismo hash, pero todos se van a quedar de un mismo lado, siempre van a caer en el mismo bit de prefijo

Cuando un cajón esta lleno y si un cajón lleno con el mismo valor hash es el único caso en el cual el cajón no se divide, se desborda, porque un cajón se desborda cuando no hay mas espacio para almacenar un nuevo ítem con misma clave de búsqueda que los anteriores, el único caso con problema, cajón lleno con todos los registros de mismo valor hash y lo que se inserta también, lo que ocurre es que se desborda: cuando el cajón se llena más allá de la capacidad y todos los elementos tienen el mismo valor hash.

4 punteros al mismo cajón, el cajón nunca tuvo que dividirse.

La implementación es mas compleja y en términos de codificación lógica de negocio, la lógica que hay que manejar. Si se implementara hash se debe entender si es con tablas o indices y si soporta organización hash dinámica o estática, todo depende del motor, son conceptos de cómo están arquitecturados, no necesariamente los implementan todos los motores.



INDICES BITMAPS

Mapear los registros a un numero, necesito que los valores puedan ser descompuestos en valores discretos (numero finito de valores diferentes, las filas puede tomar uno de los valores)

La forma mas simple es un array para cada valores diferentes. Array de bits que te dice si el valor esta presente o no en la fila. Si hay cuatro arrays tengo un mapa de bits para la columna nivel de ingreso

Cuál es el DDL de indice de vender?

Si se pide un héroe basado en genero, y se tiene indice de tipo bitmap se recupera el array del valor f y fijarme que entradas tienen 1 y recuperar esas filas. Siempre hago un index scan, el indice puede ser hash o asociativo o indice bitmap.

```
create index ...
on Emp(gender)
use Bitmap;
```

Y si ahora hacemos where gender = f and nivel_ingreso = 3, se hace un and entre array de bits de f AND array de bits del nivel 3, lo que resulta de ese array OOOOI, los 1 son los genero f y nivel de ingreso 3. Permite encontrar rápidamente las filas que cumplen con un VALOR DISCRETO y cruzar dos o mas condiciones, encontrar filas con dos o mas condiciones con operaciones de array de bits.

Los indices son estructuras auxiliares pensadas para optimizar el costo de una búsqueda de información, para perder el menor tiempo accediendo a la unidad de almacenamiento, o accediendo eficientemente para recuperar eficientemente. Por que necesitamos que sea eficiente? Para optimizar tiempo y desperdiando menos tiempo de CPU y mejoramos rendimiento (transacciones realizadas por unidad de tiempo)

Recibo una consulta que debe ser analizada (sintáctica y semánticamente) y traducida (para convertir en un algoritmo).

El sistema gerenciador debe asegurarse que la expresión sea lo mas optima posible, para eso se usa un optimizador, que transforma en un plan de ejecución, es un algoritmo que se espera que sea lo mas razonablemente optimo posible, eso es tomado como entrada de un motor de evaluación y produce el resultado de una consulta.

El análisis y traducción cumple la misma función que un analizador léxico y sintáctico de un compilador. Verifica que tenga sentido y traduce en expresión inicial. El resultado es una expresión de álgebra relacional. Una consulta puede tener distintos planes de ejecución, depende de los motores, el mas optimo depende del optimizador.

Se toma la expresión como punto de inicio y se trata de inferir una forma de acceso, se tienen planes.

Primitiva: operaciones anotadas (por ejemplo, acceso, proyección, selección) normalmente alguna primitiva puede tener una consideración especial de como ejecutar, llamada anotación.

Hay que entender cuanto cuesta un plan versus otro plan, el costo de una consulta es la suma del costo de sus primitivas. Normalmente las primitivas tienen que usar la misma métrica. (Leer esa parte)

Depende del motor de base de datos la métrica que se use, pero tiene que usar siempre la misma métrica o el mismo conjunto de métrica.

El optimizador puede usar datos anteriores para aproximar el coste de cada plan

Nunca calcula el costo real, puede ser equivalente al costo real, solo ocurre en pocos casos, siempre considera operaciones para los cuales no puede aproximar costo real sino que costo promedio, siempre es un costo estimado y quedarse con planes de ejecución de costo estimado menor

El optimizador siempre trata de devolver el mejor plan.

La optimización es una técnica por la cual se trata de buscar una solución en un espacio de solución tal que la solución se optimice o maximice.

Es un componente de la BD no TIENE MUCHO TIEMPO DE EJECUCIÓN, ES ACOTADO, no siempre escoge el plan mas optimo. Aproxima, trata de encontrar una solución razonablemente eficiente, resguardan la última solución que encontraron así que cuando se encuentre otra consulta con el mismo álgebra se utiliza y refina esa. Trabaja con información estadística (información relacionada con la forma física de los datos, forma física me permite estimar coste de acceso a los datos)

La información estadística puede ser sobre cualquier objeto de la base de datos, todos los objetos requieren de costo físico para acceder. Si fuese un B tres, necesito saber cuantos punteros puede manejar los nodos internos o el nodo hoja a nivel hoja para la búsqueda en rango o la altura del árbol, se puede estimar con la formula (ver formula) $V(A, r)$

Por que pregunto el numero de valores diferentes y no de fila? Porque para los repetidos no se indexan mas de una sola vez, y si hay se usa una estructura de cajón de punteros.

El como se consiguen queda en manos el sistema gerenciador de base de datos que es un sistema experto. Un sistema gerenciador de base de datos es un software que puede estar en distintas categorías, sistema de gestión de datos, de aprendizaje automatizado (porque aprende a media que optimiza las consultas), es uno de optimización (optimizar tiempo de procesamiento, responder en menor tiempo posible), es un sistema experto (dada una condición nos ayuda a determinar la mejor forma de recuperar la información). Clasifica en diferentes categoría de software.

Una de las operaciones que debe saber resolver es la selección.

1. Búsqueda lineal: propósito general, muy costoso, pero cuando la condición está basado en una columna que no tiene ni indice ni ordenada.
2. Búsqueda binaria: si la tabla esta ordenada por el atributo A y la condición es una igualdad sobre ese, y no existe indice en el atributo A. Ver el costo. Encontrar el bloque + recuperar información (Recorro la tabla, para recuperar el numero medio de filas que tienen el valor buscado / nro medio? de filas por bloque) se suma bloque con bloque, por eso CSAr convierte de filas con bloque. Sirve porque esta ordenado por la condición.
3. Índice primario, igualdad en clave candidata, búsqueda por índice primario para clave candidata: aquel en el cual hago la búsqueda, resuelvo una consulta de selección con un where basado en clave candidata (clave primaria que sus valores no se repite, e índice puede apuntar directo a la tabla o única que también, pero es candidato aunque no es primaria no ordena la tabla pero el índice puede ser único si es unique sirve para buscar mas eficientemente en la tabla chica)
4. tarea lo demás

Tital and tita2 que haríamos? Como la condición es and y se tiene que cumplir ambas usamos el índice para la primera condición y en memoria para la segunda condición

Si comienzo por la que no tiene índice haría búsqueda lineal.

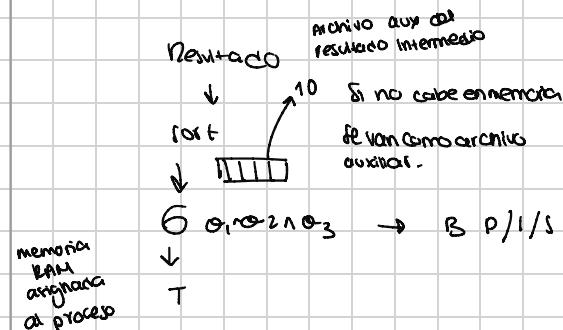
Por que es peligroso que el evaluador u optimizador cree objetos? No tienen noción del tamaño o si puede perjudicar alguna inserción o actualización y borrado, no se puede dar mucha autonomía para que creen los objetos que faltan.

Para la disyunción y negación también hay estrategias.

Selección es buscar y recuperar a través de algoritmos de búsqueda porque la operación implica encontrar fila que cumple con una condición.

Diferencia entre motores cada uno implementa de diferente manera o no tienen un algoritmo implementado y responde más lento.

Después de seleccionar datos hay una que implica ordenación, la orden by, implica que necesito una consulta ordenada.



Tengo clave de ordenación

Algoritmos de joven

Se puede resolver por un bucle anidado, por cada fila se procesa cada fila en la tabla s, si encuentro que ti y ts satisface el joven añado el par ti, ts al resultado, producto cartesiano entre tablas.

Por cada fila tengo que leer todas las tuplas de S + el número de bloque de R, es el costo asintótico cuando la combinación de los archivos no pueden caber simultáneamente en memoria. Siempre y cuando se use la estrategia de bloque anidado. Siempre ambas tablas caben en memoria el costo es bs+br porque solo leo una vez a la memoria y recorrer n veces es más rápido que el esfuerzo de leer esa cantidad de bloques, si la memoria es amplia es lineal.

Se degrada cuando un sistema está óptimo y luego cae su su su su ????????

join

La otra opción es jovén en bucles anidados por bloque. Leo dos bloques y trato de agotar, eso hace que la componente de costo cambie a br*bs+br

Qué relación hay entre br y nos, br puede estar varios ordenes por magnitud, en este caso podría ser 20*10+20 en vez de hacer 1000*10+20 como en el anterior ejemplo. El caso óptimo es si ambas caben en memoria, es un bucle anidado en memoria en esfuerzo de exceso es bs+br

Jónico con tabla que tiene índice.

Hago la búsqueda por índice, la búsqueda se hace entre clave foránea y primaria o en su defecto es una clave única/cantidades si bien no ordena la tabla apunta directo al dato porque existe una sola vez.

El costo es leer toda la tabla externa y por cada fila de venta accedemos al índice en la tabla cliente

join

Jónico de tablas ordenadas: una tabla transacción y otra de reversa, TRX y RVX, algunas transacciones tienen reversa.

Tengo tabla de lectura y de prueba, leo una fila y pruebo con una de reversa, cada que encuentro coincidencia el puntero avanza (el segundo), pero cuando el puntero menor (de A) no es igual, avanza y se puede volver a preguntar y veo si se hace el join, si es que no son, se avanza del lado menor, si son iguales se reúne y avanza el lado RVX, si no son iguales avanza TRX y cuando RVX llega al final termina la operación. Tiene un costo constante e ideal pero solo si las tablas están ordenadas por sus claves de join, cada columna de la condición es la clave de join 1 y la clave de join 2

M = Memoria disponible para ordenar

br = Cont. de bloques del archivo

1. Leer archivo desordenado de aM bloques y ordenar

Lo que tanto cuesta? Leer Br bloques

$N=3$

Escribir Br bloques

2. Ahora uso $M=2$ y b1 para leer y b2 para escribir. Y así hasta combinar.

Tomo un archivo auxiliar, escribo otro y descarto el inicial y se renombra el otro

$$2 \times \log_2 \left(\frac{12}{3} \right) + 1 = 5 \text{ bloques si llego al resultado final}$$

Y si necesito otro resultado intermedio $2 \times 2 + 1 + 1$

Join por mezcla híbrida: cuando el join por un lado es la clave primaria de la tabla y la otra es una clave única o indexada, hago join de R con el nivel hoja del índice del join 2, y cuando hago un join de una fila con registro índice de un lado queda una fila y del otro lado queda un join con puntero. Leer punteros en orden, leer la segunda tabla en orden.

El costo no se degrada tanto y es un caso su óptimo nro de bloques tabla 1 + nro de bloques nivel hoja de índice + numero de bloques tabla 2, la componente adicional es un par de ordenes de magnitud por debajo de las otras variables, es una pequeña sobrecarga, no es el óptimo es subóptimo.

Reunión por asociación de hash: por asociación, se usa cuando las tablas no caben en la memoria, hacer un bucle anidado es costoso por el tamaño de las tablas, uno indexado también es muy costoso. Cuando son muy grandes, no caben en la memoria y no se aplica razonablemente los otros algoritmos de join.

El primer paso es usar una función hash para crear N particiones, función de dispersión que mapea en N subconjuntos de entradas en subconjuntos de salida. (Leer esa parte)

Cada tabla se reparticiona con la misma función hash, no necesariamente siempre se reúnen porque puede haber colisión

Si la memoria no es suficiente para achicar las tablas hay que rehashear, quedan particiones que no quedan en la memoria, para que el algoritmo cumple una de las particiones debe caber en memoria, el algoritmo de join por hash tiene costo de $br+bs$ + la necesidad de rehash si es que la partición no entra en la memoria. En el caso que no se necesita rehasheo el término de log queda 1, solo si hay rehasheo queda mayor a uno.

Reunión híbrida al hashear bloque hoja, me aseguro que la tabla interna va a caber en memoria, entonces la asociación hash no necesita particionamiento recursivo y va a pasar que el join va a salir híbrido, fila join puntero que resuelvo en orden y así me queda el costo subóptimo.

La condición puede ser simple o complejas, conjuntivas, disyuntivas TAREA!!

Optimizador tiene que encontrar el menos costoso.

Buscamos minimizar costos dominantes.

Toma una expresión de entrada en álgebra relacional y busca variantes para poder determinar que variante y con que conjunto se puede optimizar la métrica de costo elegida.

Trata de generar planes de ejecución equivalentes entre si. Expresiones de álgebra relacional equivalentes entre si.

Un plan de ejecución (lo que produce el optimizador) define el algoritmo a ser usado en cada operación y el orden de evaluación de los operaciones.

Dados dos planes de ejecución la diferencia puede ser muy grande en cuanto al tiempo de ejecución, podría ser segundos vs días.

Optimización basada en costo: expresiones lógicamente equivalentes, anotar cada una y elegir el mas barato. La estimación se basa en información estadística y que se puede calcular de los resultados intermedios (si es que necesita para el resultado final por ejemplo) y la mejor estimación posible según los algoritmos.

Como transformar una expresión en otra: que parte se puede reescribir de otra forma, para entender esto el álgebra tiene unas reglas de equivalencia que me permite conocer cuando una expresión se puede sustituir por otra. El optimizador usa las reglas para transformar una expresión en otra.

Por que un producto cartesiano convertir en join? Para poder usar los algoritmos de join, para productos cartesianos solo tenemos bucles anidados con coste cuadrático que es el peor coste. Si no queremos caer en la operación mas costosa podemos convertir en un join y aprovechar

QUE HACE QUE EL ESPACIO DE BÚSQUEDA sea simple o complejo?

La cantidad de tablas porque la cantidad de operaciones depende de la cantidad de tablas? porque esa consulta implica varios joins. El optimizador trata la mejor forma de resolver los joins anidados el orden de los joins es importante, siempre el optimizador trata de encontrar el orden que requiera menor esfuerzo de computación

Se intenta convertir a un árbol por profundidad de la izquierda??, evitar arboles que no sean en profundidad. Ventajas: se puede reordenar las relaciones mediante una permutación de las tablas, con distintas permutaciones voy estimando cual me genera un orden interesante (Aquel ordenamiento por la izquierda o derecha pero profundidad que tenga el costo más óptimo), también se achica el espacio de búsqueda, su espacio es $O(n!)$

Cada variante de cada uno de los arboles es una subexpresión de álgebra relacional, espero que el optimizador contemple la menor cantidad para minimizar su trabajo y apurar o promover que encuentre el óptimo más rápido

Que pasa con los algoritmos que guardan en memoria? Tienen doble complejidad computacional (en cantidad de pasos) y complejidad espacial el costo de guardar en la memoria.

Buscamos el orden interesante para que la búsqueda sea la menor posible. Cuando no había optimización basada en costo se aplicaba optimización heurística por lo general, no siempre pero suelen funcionar. Se hacia la verificación sintáctica y semántica, se hacia el árbol. La optimización eurística. Es un paso subóptimo. Tratan de minimizar el tiempo de respuesta minimizando otras cosas

escibimos del ejemplo

select distinct customer-name

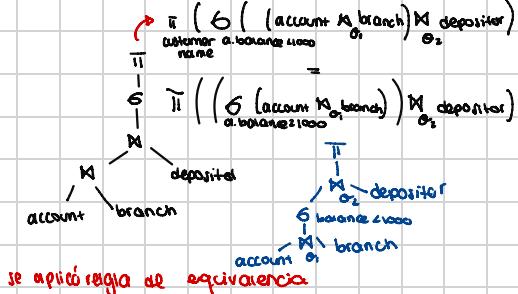
From account a

join branch b on Θ_1 $\xrightarrow{\text{no hay en el ejemplo}}$

join depositor d on θ_2

where b.branch_city = "

and a.balance < 1000;



Todo depende de la comparación, el más óptimo

$$\begin{aligned}
 &= \bar{v} \left(6 \underbrace{\left(6 \text{ variance } b_{\text{branch-city}} \right)}_{\sigma_4} \left(A \otimes B \right) \right) M_{\theta_2 D} \\
 &= \bar{v} \left(\left(6_{\theta_3}(A) M_{\theta_1} 6_{\theta_4}(B) \right) M_{\theta_2 D} \right) \\
 &\quad \swarrow \text{selection} \quad \searrow \\
 Q &= Q_1 \cup Q_2
 \end{aligned}$$

Complejo: más de 2 relaciones

$$R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$$

busca individualmente los mejores planes para cada operación y los une en un plan de ejecución.

La optimización es traducir una consulta y transformar sucesivamente hasta encontrar la más óptima en término de una métrica considerada por el motor (ej acceso al almacenamiento cuando es centralizado, si es distribuido es comunicación + acceso al almacenamiento)

Transacción: conjunto de operaciones que acceden y posiblemente actualizan datos, deben ser tratados como unidad, se tratarán como un todo para que cuando se confirme la transacción los cambios sean permanentes ante fallos.

Insertas en un programa externos o formar parte de un proceso de bd cliente-servidor, o ser parte de una script y ejecutar 10 operaciones una tras otra para que sean tratada como unidad.

Si concluye exitosamente se dice que se ha confirmado y esos cambios son permanentes. Hay un ciclo de estados de transacción, se empieza por una palabra reservada begin y se inicia y encuentra en estado activo, realizando operaciones de lectura y escritura, las operaciones no se discriminan como update, insert, delete o merge, se discriminan como lectura y escritura, porque las transacciones deben ser de forma simultánea entonces no se analiza qué operación ejecuta sino que escribe o lee. Deben ser simultáneas pero independientes y deben tener en cuenta entonces lo que leen o escriben otros.

Al encontrar END pasa a un estado parcialmente confirmado, se sabe que terminó la lectura/Escritura y lo último que queda es confirmar en forma permanente. En confirmado se supone que los datos ya son permanentes y no se pierde ante datos, igualmente la transacción es una unidad de ejecución, recién deja de existir como proceso en el sistema operativo cuando se termina. Esto cuando pasa correctamente, hay caminos pocos felices cuando hay un fallo, se pasa a un estado fallido y hay un ROLLBACK y deshacer, que sea explícita puesta por usuario o implícita por la base de datos, de eso pasa al estado abortado, asegura que los cambios fueron deshechos de tal manera que la base de datos volvió a su estado anterior. Cuando se retiran los recursos de memoria y CPU se da por estado terminada. Otra opción es cuando en parcialmente confirmada, y no pasan los chequeos de concurrencia pueden pasar que haya una cascada. chequeo hace fallo, el fallo un rollback y eso que se aborte.

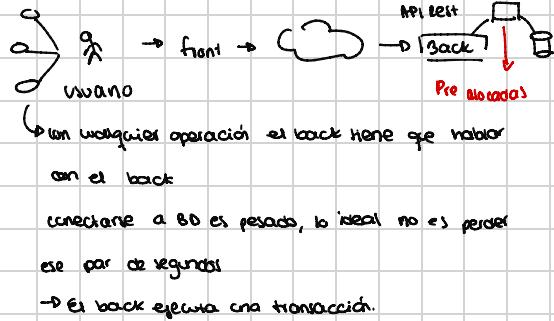
el sistema gerenciador de base de datos es expreso (lenguaje de 4to nivel, realiza optimizaciones, maneja conjunto de datos, maneja un conjunto de procesos ejecutados de manera simultánea. el sistema tiene que garantizar ciertas propiedades que se aseguran y se consigue que la transacción sea correcta.

1. **atomicidad:** una transacción es una única unidad de almacenamiento o es sin fallos o es nada
2. **consistencia:** ejecución debe ser consistente, de un estado consistente a otro consistente.
3. **aislamiento:** una transacción no hace visible sus cambios hasta que este confirmada, es la única transacción en el sistema hasta que confirma sus cambios.
4. **durabilidad:** que implica que cuando cambio en la BD y son confirmados, el cambio es permanente.

la dificultad está dada no por como se implementa como conjunto de transacciones con inicio fin, su dificultad está dada por la concurrencia.

la concurrencia es una característica necesaria, permite mayor productividad, puedo aprovechar al máximo los recursos de hardware y software. reduce el tiempo de respuesta, evita serializaciones, durante el mismo periodo de tiempo dos o más usuarios pueden ser atendidos.

Si se deja sin control la base de datos queda en estado inconsistente.



El SGBD tiene una parte que se denomina componente de control de concurrencia, controla desde el punto de vista práctico, es muy simple, cuando una transacción se ejecuta el control se hace en el nivel mas bajo posible, cuando se lee o escribe una unidad de datos (un X), el tamaño depende de la base de datos (granularidad de concurrencia) se realiza desde el punto de vista de leer o escribir una fila.

Cuando quiero leer una fila hay ciertos pasos (ver ppt cuadro).

Dos o mas transacciones tienen dos o mas informaciones de la misma fila, una de ellas esta errada y no se debería procesar. El componente tiene que evitar es que se lean o se escriban informaciones de forma inapropiada, prevenir problemas de la concurrencia,

1. Problema de la actualización perdida: dos transacciones de forma concurrente, sus operaciones no están en forma concurrente, sus operaciones se ejecutan en orden respecto a si mismas, pero entre ellas se ejecutan de forma intercalada (una parte para 1, otra parte para la 2). Son procesos que se ejecutan en formato multitarea. T1 y T2 pueden compartir datos que leen o escriben y pasa el fenómeno que se llama el de la actualización perdida, cuando dos o mas transacciones acceden a los mismos datos y sus operaciones en formato multitarea que queda un valor incorrecto en alguno de los datos. Dos procesos concurrentes con operaciones intercaladas y en algún momento la actualización de uno con respecto a otro se pierde.
2. Lectura sucia: la transacción actualiza la BD y falla por alguna razón, el ítem modificado es accedido por otra transacción antes que sea deshecho y el ítem vuelva a su valor original
3. Análisis inconsistente: una transacción realiza operaciones sobre algunos datos y otras modifica los valores de los mismos. Se puede evitar con un bloqueado pero si es larga la transacción puede reducir la concurrencia
4. Problema de la lectura no repetible: ????????? Ambos normalmente son aceptables y se permiten que ocurran porque no modifican información solo leen información y llegan a datos no exactos. No escriben en simultáneo

Se tiene que tener una lógica de control que prevenga esos 4 problemas, para tener esa lógica ese componente tiene que ejecutar las operaciones simultáneas en un formato llamado planificación: conjunto de transacciones donde se analizan el intercalamiento de esas transacciones pero se conserva el orden interno de cada transacciones individual. Todas las operaciones de T1 seguida de T2 o al revés, ahí se dicen que son planificaciones secuenciales: para n transacciones existe $n!$ Planificaciones secuenciales posibles. Esas siempre son correctas porque no hay concurrencia.

Tomando como premisa que un plan secuencial siempre es correcto se puede hacer una prueba de secuencialidad, que prueba que un plan concurrente es equivalente a uno secuencial entonces el concurrente es valido y no tiene problema. Analiza las planificaciones para ver si son equivalentes a plan secuencial, de tal manera precautelamos que se puedan reordenar en forma secuencial.

Se permite la concurrencia siempre y cuando sea equivalentes a una secuencial, y así siempre es correcta.

			Mensajes
1	N1	M1	E0 en Pend
2	N2	M2	E0 Pend E0
3	N3	M3	E0 Pend E0
4	N4	M4	Pend
5	N5	M5	Pend
6	N6	M6	Pend
7	N7	M7	Pend
8	N8	M8	Pend
9	N9	M9	Pend.

W1: begin
select * from mens
where estado = pend
order by id
limit 3;
for update
W2: update mensaje
set estado = "en proceso"
where id in (c.id)
commit
begin
enviar el mensaje incl
update mensaje
set estado = enunciado
commit;

Una estrategia es secuencialidad en cuanto a conflictos, veo las instrucciones

No puedo intercambiar en relación al tiempo. Al menos una de las operaciones siempre es escribir. En el momento que hay una escritura el orden de las operaciones importa.

Secuencialidad en cuanto a vistas (SI concurrente y S2 secuencial)

Si hay T1 que lee valor inicial de una columna campo fila en todas las planificaciones siempre tiene que leer el valor inicial.

Si en SI hay una relación entre T1 y T2 en el otro plan también tiene que haber una relación

Para cada columna, fila, tabla, para que sean equivalentes si T1 escribe el ultimo valor en el plan 1, en el plan 2 también debe ocurrir eso.

Una estrategia es secuencialidad en cuanto a conflictos, veo las instrucciones

No puedo intercambiar en relación al tiempo. Al menos una de las operaciones siempre es escribir. En el momento que hay una escritura el orden de las operaciones importa.

Secuencialidad en cuanto a vistas (S1 concurrente y S2 secuencial)

Si hay T_i que lee valor inicial de una columna campo fila en todas las planificaciones siempre tiene que leer el valor inicial.

Si en S1 hay una relación entre T_i y T_j en el otro plan también tiene que haber una relación

Para cada columna, fila, tabla, para que sean equivalentes si T_i escribe el último valor en el plan 1, en el plan 2 también debe ocurrir eso.