**Bab 2. File dan Indexing**

Dalam penggunaan database managemen system (DBMS), implementasi di lapisan bawah memerlukan mediasi system file melalui file manager. Ketika data disimpan dan diakses perlu untuk dipelajari tentang pengorganisasian dan metode akses. Salah satu dari beberapa pengorganisasian file hal yang dibicarakan adalah indexing. Berikut adalah pemaparan secara lebih terinci.

**2.1 Sistem File**

Terdapat dua sudut pandang dari system file yaitu

* Aplikasi

Sudut pandang aplikasi adalah sudut pandang perangkat lunak baik dari pengguna maupun system operasi. Sebagai gambaran beberapa hal yang dibicarakan seperti byte array, persistent across reboots and power failures, secondary (non-volatile) storage device seperti hard disks, USB, CD, etc, serta map bytes sebagai kumpulan blok pada storage.

* Hardware

**Lapisan Sistem File**



Setiap lokasi memori mempunyai alamat yang secara langsung dapat diakses. Pada files semuanya relative, seperti lokasi dari suatu file tergantung pada direktori dimana dia diletakkan dan lokasi tepatnya posisi sekarang dari read atau write di dalam file.

Ilustrasi misalkan untuk membaca satu byte pada suatu file spesifik:

* Pertama adalah mendapatkan file pada path direktori dan menyelesaikan pengenal (identifier). - expensive for each access
* Gunakan pointer baca untuk menempatkannya pada posisi byte

Hal2 tersebut dilakukan dengan cara menggunakan open system call untuk membuka file tersebut sebelum akses apapun.

Struktur file terdiri dari komponen2: character, field, record, blok, dan file itu sendiri yaitu file terdiri dari blok2 yang setiap blok terdiri dari satu atau lebih record dst.

|  |  |
| --- | --- |
| Pemrosesan data terjadi di memory, jadi ketika data pada storage ingin diproses (misalkan diubah), data tersebut harus dibaca dari storage (misalkan harddisk).  Terdapat buffer yaitu sejenis memory yang menampung hasil pembacaan, dan pembacaannya adalah per blok data pada file. Berikut mekanisme pembacaan data dari disk   * Read pertama adalah membaca 1 blok dari strorage dan record pertama pada blok tsb diberikan ke memory untuk diproses. * Pembacaan berikutnya akan mengambil record berikutnya dari blok tersebut dst sampai semua record terbaca. Ketika hal ini terjadi, pembacaan berikutnya adalah mengambil blok berikutnya, dst. |  |

**Operasi File**

* File Creation

1. Create FCB for the new file
2. Update directory contents
3. Write new directory contents to disk (and may • . cache it as well)

* Membuka File

|  |  |
| --- | --- |
|  | 1. Aplikasi mengirimkan nama file melalui open system call 2. sys\_open menelusuri tabel system-wide open file untuk melihat apakah file tersebut telah digunakan oleh proses lain  * Jika ya, increment count penggunaan dan tambahkan pointer pada table open file per-process |
| * Jika tidak, cari nama file pada struktur direktori (entah didalam cache atau disk) dan tambahkan ke table system-wide open file dan juga table open file per-process  1. Pointer tersebut (atau index) pada tebel open file perproses dikirim balik ke aplikasi. Hal ini menjadi file descriptor | |

* Akses Data (Read)

|  |  |
| --- | --- |
|  |  |

* Menutup file

1. Entri tabel open per process dihapus
2. Count referensi table system wide open dikurangi 1.

* Jika nilainya menjadi 0 maka perubahan2 dikopi kembali ke disk (jika dibutuhkan)
* Hapus entri table *system wide open*

**Meta data file**

|  |  |
| --- | --- |
| Name | the only information kept in human readable form. |
| Identifier | A number that uniquely identifies the file within the file system. Also called the inode number |
| Type | File type (inode based file, pipe, etc.) |
| Location | Pointer to location of file on device. |
| Size |  |
| Protection | Access control information. Owner, group (r,w,x) permissions, etc. |

Pada sistem operasi lain terdapat kebutuhan lain sehingga diperlukan meta data berikut yang tidak dibahas satu persatu: external name, current state, sharable, owner, user, locks, protection settings, length, time of creation, time of last modification, time of last access, reference count

**File Access Methods**

* Sequential Access: data diproses satu persatu per block satu setelah lainnya. Penggunaannya adalah sudah jarang dan hanya tipikal kebutuhan tertentu
* Direct Access: data diproses mengakse blok langsung sesuai yang ditunjuk. Untuk kebutuhan tersebut perlu dibuat mekanisme yang memungkinkan misalkan dengan indexing atau hashing, Metode akses seperti ini dibutuhkan oleh system basis data untuk mengefisienkan pengaksesan data.

Berikut sedikit tentang penangan file pada bahasa pemrograman

|  |
| --- |
| **Files in C**  C supports two types of files: character files (streams) and binary files  Operations on files:   * open a file:   FILE\* fopen (char \* name, char mode)   * close a file   void fclose (FILE\* file)   * read a file   size\_t fread (void\* b, size\_t s, size\_t n, FILE\* f)   * write a file   size\_t fwrite (void\* b, size\_t s, size\_t n, FILE\* f)   * position a file   int fseek (FILE\* f, long offset, int base)  where *base* can be:  SEEK\_SET - from the beginning  SEEK\_CUR - from current position  SEEK\_END - from the end   * test end of file   int feof (FILE\* f)  Example Create a file containing records of the following form:  struct *Records*{  char *first*[20] ;  char *last*[20];  int *id\_num*; // this is the key  int age;  }  together with an index file |

***Indexing***

*Indexing* adalah salah satu cara mengorganisasi record pada suatu file. Ia menghasilkan seperangkat file yang disebut file index mengandung penunjuk menuju record dari file data. Untuk gambaran berikut rinciannya

* file index - adalah file berisi suatu searchkey dan pointer record yang sesuai pada file data

  struct Index{

KEY key;

long int record;

};

* file data file – mengandung record data

Terdapat dua jenis index dari perspektif kepadatan: dense dan sparse. Dense index adalah index yang berisi semua entry atribut (padat) yang terpilih sebagai search key menunjuk ke record sesuai nilai atribut. Sparse index adalah index yang berisi entry atribut search key record pertama setiap blok (jarang).

Baik dense maupun sparse index Jika search key adalah atribut primary key, maka index yang terbentuk disebut primary index, sedangkan jika search key adalah atribut selain primary key disebut secondary index. Berikut ilustrasi dense dan sparse index dimana a sebagai search key.

|  |  |
| --- | --- |
| Primary index secara Dense index | Primary index secara Sparse index |
|  |  |

Pada implementasinya biasanya digunakan mixed kedua jenis index tersebut seperti contoh berikut, mengilustrasikan secondary index dimana nilainya tidak unik, perhatikan nilai pada atribut a sebagai search key tidak unik. (seperti telah disebut sebelumnya, suatu index pada atribut yang bukan primary key sebagi search key disebut secondary index )

|  |  |
| --- | --- |
|  | Index level-1 adalah dense index sedangkan index level berikutnya adalah sparse index |

Berikut diberikan beberapa kesimpulan dari hal2 yang telah dijabarkan

|  |  |  |
| --- | --- | --- |
| Kriteria | Dense | Sparse |
| **Space required**  **Block access**  **Access to entry**  **Exists-queries**  **Usage**  **Updates/changes** | One index field per entry  “Many”  Direct access  Uses only the index  All cases  Always updated if entry  sequence is changed | One index field per data block  “Feẘ”  Must search in the data block  Must always access the data  Block  Not on unsorted elements  Updated only if the first entry of the data block is changed |

Pada contoh implementasi secara mixed kedua jenis index tersebut tidak hanya baik untuk kasus secondary index, namun pada primary index sekalipun. Seperti kita tahu file data sebenarnya biasanya berisi data yang tidak terurut (acak) seperti contoh berikut yang disatukan dengan pembahasan metode indexing. Jika file data sebenarnya adalah tidak terurut maka sparse index tidak mungkin digunakan. Sparse index hanya bisa digunakan ketika data sudah terurut. Berikut akan dibahas beberapa metode indexing: Linear, BST, dan B-Tree.

|  |  |
| --- | --- |
| 1. **Linear indexing**   Metode ini menggunakan file data secara sequential dengan penyimpanan fisik blok secara contiguous satu setelah lainnya. Di atas file data adalah file index itu sendiri. Walaupun dense index dapat digunakan namun dengan data besar sparse indexlah yang layak digunakan karena lebih efisien. Penemu metode ini adalah IBM yang dikenal dengan metode index sequential access metode (ISAM), yaitu terdapat index di atas file sequential. Berikut penggambaran dari metode ini.  Anggap spesifikasi file data adalah sebagai berikut:   * Jumlah data = 200 records * 1 blok = 20 records * Maka jumlah blok file adalah 200/10 = 10 blok   Pada gambar kotak-kotak paling kanan adalah data itu sendiri (200 records, dengan jumlah blok 10). Jadi record ke 181 atau R181 berada pada blok 10 sedangkan R22 berada pada blok 2 dst.  Oleh karena itu pada Level-1 index hanya diperlukan 1 blok index. Jumlah record pada blok index hanya 10 record (sesuai dengan jumlah blok data). Nilai Record pertama dari blok index menunjuk ke nilai record pertama dari file data, dst | R40  R200  R181  R20  R2  R1  …  20  10  2  1  …  Level-1 Index  (Hanya 10 recs)  21  …  …  39  40  1  21  …  181  22  Data Sebenarnya  20  …  …  1  2  181  …  …  199  200 |

Apabila file data bukan file sequential, yang berarti data di dalamnya tidak terurut maka mekanisme indexingnya sedikit berbeda. Pada file data sequential di atas dapat langsung digunakan sparse index, tidak demikian jika fie data tidak terurut. Berikut adalah mekanisme pengindex-an file data tidak terurut.

10 recs sisanya kosong

R181

…

20

10

2

1

1 blok = 20 records

R40

R21

R199

R200

…

…

R20

R1

R2

Hanya terisi 10 recs pointer

Level-2 Index

Level-1 Index

Data Sebenarnya

20

…

…

1

2

181

…

…

199

200

23

40

199

21

23

…

…

39

40

21

2

42

200

19

175

1

1

21

…

181

22

Biasanya data sesungguhnya pada file tidak tersusun (not sorted), sehingga diperlukan jenis index dense dimana setiap record dari index tersebut menunjuk ke record data sebenarnya. Ini adalah index level-1, dimana isian data di sini tersusun (sorted) biasanya menaik. Pada level index berikutnya yaitu level-2 sudah dapat digunakan jenis index sparse karena sudah terurut.

Isian data setiap record menunjuk ke blok, misalkan record-1 (berisi nilai 10), menunjuk ke blok pertama level index berikutnya.Lalu isian data pada record-2 (berisi nilai 30) menunjuk ke blok kedua dst.

Mekanisme pencarian adalah pertama masuk ke index level tertinggi (seperti root), baca blok lalu bandingkan nilai yang dicari dengan isian record-per-record pada blok tersebut. Terdapat dua keadaan di sini:

* Ketika pada record tertentu nilai sama ikuti pointer pada record tersebut menunjuk ke blok level index di bawahnya
* Ketika pada record tertentu nilai lebih kecil, periksa record berikutnya dan lakukan perbandingan lagi
* Ketika pada record tertentu nilai lebih besar, ikuti pointer pada record sebelum ini menunjuk ke level index di bawahnya.

Lakukan berulang sampai tercapainya index dense , dimana pada index dense ini baca blok data sebenarnya sesuai yang ditunjuk oleh record dengan nilai yang dimaksud

1. **Binary Search Tree Indexing**

Pohon pencarian biner adalah pohon biner yang mungkin kosong. Pohon biner yang tidak kosong memiliki properti berikut:

* Setiap elemen memiliki sebuah komponen kunci, yang nilainya berbeda
* Kunci di subtree kiri dari root adalah lebih kecil dari kunci di root
* Kuncinya di sub-pohon kanan dari akar lebih besar dari yang di root
* Baik kiri dan kanan subpohon juga merupakan pohon pencarian biner.

Berikut adalah contoh mengilustrasikan property tersebut di atas.

|  |  |
| --- | --- |
| x  > x  < x | 23  31  7  11  3  37 |

Pada BST data ada di setiap node. Ketika ada operasi terhadap BST digunakanlah langkah sesuai properti juga tentunya. Operasi-operasi yang ada pada BST adalah create, insert, update, delete.

Untuk mendukung beberapa operasi biasanya dibutuhkan tambahan operasi misalkan adakah suatu nilai dari search key ada pada himpunan ini. Berikut diberikan operasi dari hal terakhir tersebut (mendapatkan adakah suatu nilai search key pada himpunan)

**type**

*nodetype* = **record**

*element*: elementtype;

*aki, aka*: *nodetype*

**end**;

**type**

*SET* = *nodetype*

**function** *find*(*x*:elementtype, *a*: *SET*) : **boolean**

**begin**

**if** (*a* = **nil**) **then**

**return false**;

**else if** (*x* = *a*.*element*) **then**

**return true**

**else if** (*x* < *a*.*element*) **then**

**return** (*find*(*x*, *a*.*aki*))

**else if** (*x* > *a*.*element*) **then**

**return** (*find*(*x*, *a*.*aka*))

**end**;

Latihan:

Create dua buah BST untuk searchkey bernilai berturut-turut berikut, dan analisis hasilnya

* 1, 2, 3, 4, 5, 6, 7, 8, 9
* 5, 3, 7, 2, 1, 4, 8, 6, 9

|  |
| --- |
| Studi kasus: implementasikan BST untuk indexing permasalahan dokumen retrieval |

1. **2-3 Trees**

Suatu 2-3 tree adalah tree yang mempunyai properties sebagai berikut:

* Setiap interior node mempunyai dua atau tiga anak
* Tidak seperti BST, pada 2-3 tree data sebenarnya ada di leaf
* Setiap path dari root ke leaf mempunyai panjang sama, jadi termasuk dalam balanced tree

|  |  |
| --- | --- |
| Interior node  leaf  7  16  5  -  8  12  19  - |  |

* Pada interior node terdapat dua field, sebut saja x, dan y dimana
  + x adalah descendant kedua terkecil
  + y adalah descendant ketiga terkecil
* descendants dari interior node memenuhi criteria berikut:

y

x

< x

>= x

>= y

< y

**Insert ke 2-3 tree**

Dari keadaan tree pada gambar di atas, diinsert 18, prosesnya:

* Pergi ke root
* Bandingkan 18 dengan x dan y, hasilnya 18>16, maka pergi ke anak ke tiga.
* Bandingkan 18 dengan x dan ya pada anak ketiga, hasilnya 18<19, maka pergi ke anak pertama
* Ternyata sudah mencapai leaf, pergi ke posisi yang layak untuk 18.

7

16

5

-

8

12

19

-

Exercise: Lakukan operasi berikut dari keadaan terakhir tree: Insert 10, diikuti Delete 10, diikuti delete 7

**Btree**

Adalah jenis khusus balanced m-ary tree yang memungkinkan me-retrieve, insert, delete record dari file eksternal. Ia adalah generalisasi dari 2-3 tree.

Secara formal suatu B-tree adalah suatu m-ary search tree dengan properties berikut:

* Root adalah salah satu: dia leaf atau jika tidak paling sedikit punya anak 2.
* Setiap node kecuali root dan dan leaves mempunyai anak diantara m/2 dan m.
* Setiap path dari root ke leaves mempunyai panjang sama.

18

10

12

22

28

34

38

4 6 8

12 14 16

10

18 20

22 24 26

28 30 32

34 36

38 40 42

B-Tree dengan order=5

* Seperti halnya 2-3 tree, pada Btree data sebenarnya ada di leaf