B-Tree dan Penerapan di Basis Data

Aldyaka Mushofan-13512094
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512094@std.stei.itb.ac.id

Abstract- B-Tree ialah suatu struktur data yang sangat populer digunakan dalam pengelolaan basis data. Hal ini dikarenakan B-Tree dapat menampung banyak data pada tiap nodenya. Dapat perkembangannya, B-Tree diterapkan dalam berbagai penanganan kasus pengelolaan basis data dan B-Tree juga dikembangkan bentuknya dalam berbagai variasi. Pada makalah ini, akan dijelaskan sedikit mengenai pemberian indeks pada basis data dan B-Tree serta perkembangan penerapannya pada zaman sekarang.

Index Terms—B-Tree, Basis Data, Indexing, Media penyimpanan

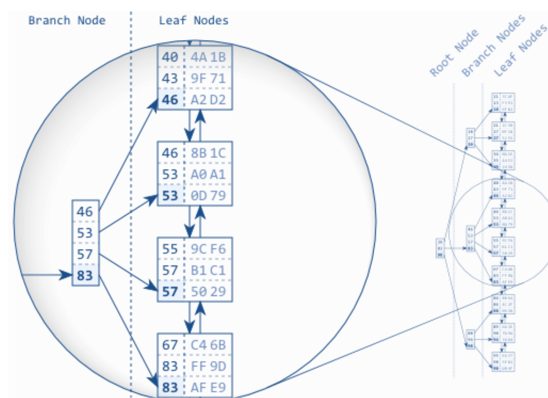
I. Pendahuluan

Sekarang ini banyak kegiatan yang menggunakan pengelolaan data. Data yang dikelola berasal dari basis data (*database*). Dalam pengelolaan data dari *database*, terdapat bermacam-macam cara, dan salah satunya ialah pemberian indeks pada *database* tersebut. Dengan menggunakan indeks, pencarian suatu data yang terdapat di *database* dapat dilakukan dengan cepat, sehingga dapat meningkatkan kinerja dalam mengelola data.

Dengan memberikan indeks pada data, pengelolaan data, seperti menambah, menghapus, atau memperbarui suatu data, dapat dilakukan secara mudah dan cepat. Oleh karenanya, pemberian indeks pada data merupakan hal yang sangat penting dan mendasar pada pengelolaan *database*.

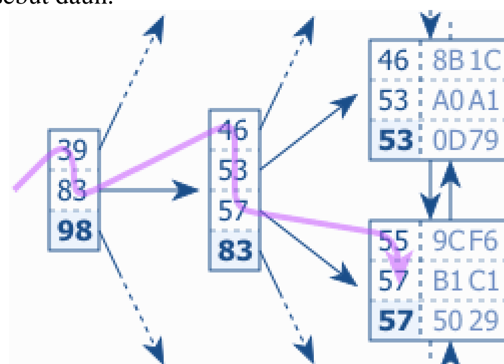
Pemberian indeks pada data di *database*-pun terdapat bermacam-macam metode, salah satunya ialah menggunakan pohon. Untuk pemberian indeks menggunakan pohon dilakukan tidak secara terurut, akan tetapi secara acak.Oleh karena itu, agar pencarian data dapat dilakukan secara cepat, maka dapat menggunakan *B-Tree* (*Balanced Tree*).

B-Tree ialah pohon dengan tinggi yang sama di setiap daunnya.



Gambar 1 Indeks dengan B-Tree

Gambar 1 menunjukkan ilustrasi pemberian indeks menggunakan pohon. Dari gambar di atas terlihat bahwa pohon tersebut di setiap daunnya memiliki tinggi yang sama, inilah yang dinamakan *B-Tree*. Pada gambar di atas, terlihat bahwa setiap cabang menghubungkan suatu nilai dari lapisan atas ke nilai yang sama pada lapisan di bawahnya yang merupakan nilai terbesar pada daun di lapisan tersebut. Berdasarkan metode ini, pohon akan terus membuat lapisan hingga suatu data yang dicari ditemukan. Lapisan terus dibuat hingga suatu data dapat termuat pada 1 *node* saja, yang kemudian disebut daun.



Gambar 2 Pencarian data 57 secara transversal

II. Teori

A. Indeks

Seperti yang sudah dijelaskan pada pendahuluan, indeks adalah suatu cara untuk mengelola suatu basis data. Indeks ialah pemberian suatu pengenal kepada data yang ada di basis data. Dengan melakukan *indexing*,

maka kita melakukan pembuatan struktur data yang baru. Struktur data tersebut selanjutnya bertanggung jawab dalam penunjukkan data tersebut.

Metode dalam pemberian indeks ini bermacam-macam, mulai dari metode pemberian indeks dalam 1 lapisan hingga pemberian indeks menggunakan sub-lapis.

Metode yang digunakan dalam pemberian indeks ini nantinya akan berpengaruh terhadap pengelolaan basis data. Misalkan menggunakan metode 1 lapis, bila data yang ada sangat banyak, maka memerlukan waktu yang lama dalam pencarian data tersebut. Namun apabila telah menggunakan metode yang memungkinkan adanya sub-lapisan, maka data yang dicari dapat ditemukan dengan lebih cepat.

Indexing merupakan hal yang sangat penting dalam pengelolaan basis data. Apabila metode *indexing* yang digunakan tidak tepat dan mengakibatkan pencarian data membutuhkan waktu yang lama, maka pengelolaan data pun tidak akan maksimal dan menghabiskan banyak waktu, namun apabila metode *indexing* yang digunakan tepat, maka pengelolaan data pun dapat lebih menghemat waktu dan lebih optimal.

Oleh karenanya, banyak dilakukan penilitian terhadap metode *indexing* demi tercapainya pengelolaan basis data yang sangat cepat dan optimal sehingga sampai saat ini struktur data yang digunakan dalam *indexing* cukup beragam dan banyak.

B. B-Tree

Salah satu struktur data yang dibuat dan digunakan untuk *indexing* ialah *B-Tree* (*Balanced Tree*). *B-Tree* ialah struktur data pohon yang tiap daunnya memiliki tinggi yang sama.

B-Tree pertama kali diciptakan oleh Rudolf Bayer dan Ed McCreight pada tahun 1972. *B-Tree* dibuat memungkinkan untuk menyimpan banyak data dalam satu node, jumlah subpohonnya juga dapat sangat banyak. Karena itulah, *B-Tree* sangat cocok untuk digunakan dalam pengelolaan data pada *disk*.

B-Tree dapat digunakan pada situasi dimana sebagian atau semua bagian dari pohon harus disimpan dalam *secondary storage*. Hal ini memungkinkan *B-Tree* dapat meminimalisasi jumlah akses terhadap *disk*.

Sebuah *B-Tree* memiliki jumlah minimal anak pada setiap *node*-nya, disebut faktor minimal. Misalkan factor minimal dari suatu *B-Tree* ialah n, maka setiap *node* harus memiliki minimal *keys* sejumlah n-1.

Tinggi dari *B-Tree* dengan jumlah *key* n dan *degree* t dapat dicari memiliki nilai

$$h \leq \log_t \frac{n+1}{2}$$

Dari persamaan tersebut dapat ditentukan kasus terburuk dari *B-Tree* sebesar O(log n).

III. Operasi Pada B-Tree

Pada *B-Tree*, terdapat beberapa operasi, *search, insert, delete,* dan *create*.

1. Search (pencarian)

Untuk mencari suatu data dengan indeks yang diinginkan, maka dicari terlebih dulu suatu nilai pada node yang memiliki nilai yang lebih besar dari nilai yang diinginkan. Bila telah ditemukan, maka penunjuk akan menunjuk nilai sebelumnya, lalu masuk ke anak yang ditunjuk oleh nilai tersebut, dan mengulangi langkah tersebut hingga ditemukan. Bila pada suatu node tidak ditemukan nilai yang lebih besar dari nilai yang dicari, maka penunjuk akan mengikuti node yang ditunjuk oleh nilai terbesar dari node tersebut.

Dalam notasi algoritma, perintah pencarian dalam *B-Tree* dapat ditulis sebagai berikut:

```
i <- 1
while i <= n[x] and k > key_i[x]
    do i <- i + 1
if i <= n[x] and k = key_i[x]
    then return (x, i)
if leaf[x]
    then return NIL
    else Disk-Read(c_i[x])
        return B-Tree-Search(c_i[x], k)
```

Kemungkinan terburuk dari pencarian dalam *B-Tree* ialah $O(\log_t n)$.

2. Insert (memasukkan)

Untuk melakukan perintah *insert*, sebelumnya harus melakukan perintah *search* terlebih dahulu untuk menentukan letak dimana data akan diletakkan. Setelah letak berhasil ditentukan, maka perintah *insert* dapat dieksekusi. Bila node belum penuh, maka perintah *insert* langsung dapat dilakukan, tetapi apabila node telah penuh, maka node harus dipecah terlebih dahulu untuk membuat node baru dan memasukkan anak baru dari node tersebut. Bila ternyata node orangtua dari node yang akan dipecah telah penuh, maka node orangtua tersebut harus dipecah terlebih dahulu. Perintah terus diulang hingga node tidak penuh.

Dalam notasi algoritma, perintah *insert* dapat ditulis seperti berikut:

```
i <- n[x]
if leaf[x]
```

```
then while i >= 1 and k < key_i[x]
     do key_i+1[x] <- key_i[x]
      i <- i - 1
    key_i+1[x] <- k
   n[x] <- n[x] + 1
   Disk-Write(x)
else while i >= and k < key_i[x]
     do i <- i - 1
   i <- i + 1
   Disk-Read(c_i[x])
   if n[c_i[x]] = 2t - 1
      then B-Tree-Split-Child(x, i, c_i[x])
         if k > key_i[x]
               then i <- i + 1
    B-Tree-Insert-Nonfull(c_i[x], k)
```

Dalam perintah *insert*, kasus terburuk memerlukan waktu sebesar $O(\log_t n)$, mengingat perintah membagi node memakan waktu secara linear, maka waktu yang diperlukan tidak terlalu berpengaruh terhadap waktu yang diperlukan dalam perintah *insert* ini.

3. Delete (hapus)

   Dalam melakukan perintah hapus, maka banyak hal yang harus diperhatikan. Apabila penghapusan dapat mengurangi nilai *key* dari suatu node dan nilainya menjadi di bawah minimal nilai *key,* maka harus dilakukan penyesuaian yang cukup rumit seperti pengurangan tinggi dari pohon yang mempengaruhi node-node lainnya.

4. Create (membuat)

   Perintah *create* pada *B-Tree* akan membuat node yang kosong. Pembuatan ini dilakukan dengan mengalokasikan sebuah akar baru dengan nilai *key* kosong dan menjadikannya daun.

   Perintah *create* dapat ditulis dalam notasi algoritma sebagai berikut:

```
x <- Allocate-Node()
leaf[x] <- TRUE
n[x] <- 0
Disk-Write(x)
root[T] <- x
```

   Perintah pembuatan dalam *B-Tree* memerlukan waktu sebanyak O(l).

IV. Penerapan B-Tree Pada Basis Data

Dalam perkembangannya, *B-Tree* banyak diadaptasi dalam berbagai struktur data dalam *indexing*. Perkembangan ini mulai dari mengembangkan *B-Tree* itu sendiri sampai menggabungkan struktur data *B-Tree* dengan struktur data lainnya. Selain itu, penerapan *B-Tree* juga di berbagai macam tipe penyimpanan data.

A. B-Tree Pada Flash Memory

Flash memory merupakan desain tipe penyimpanan alternatif selain *harddisk*. *Flash memory* yang tahan goncangan, hemat energi, dan *nonvolatile* menjadikannya media penyimpanan alternatif yang populer.

Pada *Flash memory*, *B-Tree* dapat digunakan dalam pengelolaan operasi *bit-wise* yang ada di *Flash Transition Layer* (*FTL).* Penggunaan *B-Tree* pada *Flash memory* ini diajukan oleh Chin-Hsien Wu, Tei-Wei Kuo dan Li Ping Chang dalam makalahnya “An Efficient B-Tree Layer Implementation for Flash-Memory Storage Systems”. Mereka mengajukan untuk menggunakan *B-Tree* dalam pengelolaan *Flash Memory* karena menurut mereka akses *B-Tree* sangat cocok dalam mengelola data di *Flash memory*, mengingat data yang ada di *Flash memory* tidak bisa ditulis ulang dan jika berkeinginan untuk menulis suatu data, maka harus dilakukan penghapusan terlebih dahulu di data yang akan ditimpa. Dalam proses penghapusan inilah terdapat batasan, ketika menghapus, maka daya guna dari *flash memory* ini juga berkurang, karena setiap bagian yang dapat dihapus memiliki batas penghapusan.

B-Tree nantinya akan membuat lapisan baru yang dinamakan *BFTL*. *BFTL* dapat sewaktu-waktu ditambahkan pada *FTL* atau dihilangkan, sesuai kehendak dari pengguna. *BFTL* akan diletakkan di antara lapisan *file system* dengan *FTL*. Dengan adanya *BFTL, B-Tree* dapat memanipulasi indeks dengan lebih leluasa. Pada penerapan *B-Tree* pada *FTL*, tidak diperlukan perubahan pada *B-Tree*.

Menurut Chin-Hsien Wu, Tei-Wei Kuo dan Li Ping Chang, *BFTL* dianggap sebagai bagian dari Sistem Operasi. *BFTL* tersusun atas *reservation buffer* dan *node translation table*. Ketika terdapat perintah menambah, menghapus atau mengubah data, maka data yang diolah ialah data pada lapisan *BFTL,* tepatnya pada bagian *reservation buffer*. Setelah beberapa pengubahan, barunlah data tersebut kemudian dimasukkan ke *flash memory*. Hal ini akan mengoptimalkan masa guna dari *flash memory*, karena pengolahan data tidak langsung pada *flash memory,* akan tetapi pada lapisan *BFTL* terlebih dahulu, sehingga bila terdapat kesalahan atau banyak pengubahan data, mengurangi jumlah penghapusan dari *block memory* pada *flash memory.*

Pada bagian *node-translation table* dari *BFTL,* dilakukan pengelolaan indeks. Pada bagian ini, indeks dikumpulkan dan dijaga,

sehingga pengelolaan indeks dapat dilakukan secara lebih efisien.

B. String B-Tree

String B-Tree ialah sebuah struktur data yang digunakan untuk pencarian *string* yang dilakukan di memori eksternal. *String B-Tree* ini merupakan suatu penghubung antara struktur data *external-memory* dan *string matching*.

String B-Tree secara singkat merupakan penggabungan antara *B-Tree* dengan percobaan *Patricia* pada *internal-node*. Dengan demikian, struktur data ini dapat menjadi lebih efektif dengan menambahkan penunjuk tambahan sehingga dapat mempercepat proses pencarian dan pengolahan data.

String B-Tree secara teori memiliki kecepatan yang sama dengan *B-Tree* pada kasus terburuk, akan tetapi memiliki kemampuan yang jauh lebih unggul dalam pencarian, seperti pada *suffix tree*. Selain itu, *String B-Tree* juga lebih efektif dalam pengelolaan memori utama, karena *String B-Tree* telah mampu mengembangkan *online suffix tree search* pada set *string string* yang dinamis.

Kelebihan lain dari *string b-tree* ialah ia dapat diaplikasikan pada *indexing* dan duplikasi perangkat lunak. Secara singkat, *String B-Tree* memiliki kelebihan dari *B-Tree* dan *suffix tree* tetapi menghilangkan beberapa kelemahan dari *B-Tree* dan *suffix tree* tersebut.

String B-Tree ini diajukan oleh Paolo Ferragina dan Roberto Grossi pada makalah mereka yang berjudul “The String B-Tree: A New Data Structure for String Search in External Memory and Its Application”.

Mereka membuat struktur data ini karena selama ini, perkembangan pengelolaan data pada memori eksternal sangat lamban dan semakin banyaknya data yang berupa *string* yang terdapat di memori eksternal.

C. iDistance

iDistance ialah salah satu penerapan dari pengembangan *B-Tree* (*B⁺-Tree*) yang digunakan pada *indexing* untuk pencarian tetangga terdekat (*Nearest Neighbor Search*) pada ruang metrik dengan dimensi yang tinggi (*high-dimentional metric space*). *iDistance* membagi data berdasarkan strategi pembagian data, kemudian memilih titik referensi untuk setiap bagian-bagian data tersebut. Titik-titik data tersebut kemudian dikelompokkan ke dalam satu dimensi metrik berdasarkan kemiripan terhadap titik referensi yang telah dipilih sebelumnya. Hal ini menjadikan titik-titik tersebut dapat diberi indeks dengan menggunakan struktur data *B^+-Tree* dan *K-nearest nighbori* (*KNN*) secara pencarian satu dimensi.

Penggunaan *iDistance* diajukan oleh H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, dan Rui Zhang dalam makalah mereka “iDistance: An Adaptive B^+-Tree Based Indexing Method for Nearest Neighbor Search”.

Menurut mereka, *iDistance* merupakan struktur data yang dapat diaplikasikan pada berbagai macam distribusi data, berbeda dengan berbagai macam struktur data dengan tujuan yang sama yang hanya dapat digunakan pada distribusi data tertentu dan gagal diaplikasikan pada distribusi lainnya.

Tingkat keefektifan dari struktur data ini dalam pengelolaan suatu distribusi data bergantung pada bagaimana data terbagi-bagi, dan bagaimana titik referensi tersebut dipilih.

Teknik yang digunakan pada struktur data ini ialah dengan membagi terlebih dahulu suatu data dan menentukan titik referensi dari setiap bagian data tersebut. Setelah itu, jarak dari tiap bagian data diberi indeks. Karena jarak antar bagian data tersebut berupa angka skalar yang sederhana, maka hanya dibutuhkan sedikit upaya dalam penjagaan jarak dari data tersebut, dan *B^+-Tree* yang sederhana mampu untuk menanganinya.

Pada *KNN* dengan *query* yang berpusat di q, dengan radius *query* sebesar r, pencarian jarak yang lebih kecil dari r dilakukan dengan mengeksplorasi bagian-bagian data yang telah dibuat sebelumnya. Ini dilakukan hingga kondisi berhenti tercapai.

Pembagian data yang dilakukan pada *iDistance* terdapat beberapa cara:

a. *Space-based Partitioning*

Pembagian data dengan metode ini ialah dengan membagi data yang ada berdasarkan jumlah dari ruang kosong sehingga besar/jumlah ruang kosong pada setiap bagian data sama.

Dengan metode ini, dimungkinkan beberapa titik referensi, antara lain:

- *Pusat dari Hyperplane, jarak terdekat*

  Dengan metode ini, dihasilkan pembagian data yang memiliki bentuk mirip seperti pohon piramida (*Pyramid Tree*)

- *Pusat dari Hyperplane, jarak terjauh*
- *Titik luar*

  Titik luar yang dimaksud ialah semua titik sepanjang garis yang dibentuk dari titik tengah *hyperplane* yang jatuh pada luar ruang data.

b. *Data-Based Partitioning*

Sesuai dengan nama yang ada, pembagian data ini berdasarkan banyak data yang ada, kemudian data dibagi sehingga tiap bagian memiliki jumlah data yang sama.

Terdapat dua kemungkinan pemilihan titik referensi dengan menggunakan metode ini, antara lain:

- *Pusat kluster*

  Pusat dari kluster ini selalu menjadi kandidat untuk menjadi titik referensi.
- *Sisi kluster*

D. Query dan Update pada B^+-Tree berdasarkan Indexing Moving Object

Pada struktur data ini dimungkinkan struktur data *B^+-Tree* untuk mengelola indeks dari objek yang bergerak.

Untuk dapat menjadikan *B^+-Tree* dapat mengelola indeks untuk objek yang bergerak, kita terlebih dahulu harus mampu merepresentasikan lokasi dari objek yang bergerak tersebut ke dalam fungsi lurus terhadap waktu. Selanjutnya, kita "membagi" indeks, menempatkan setiap masukan ke bagian-bagian tersebut sesuai waktu.

Indexing pada objek yang bergerak menggunakan *B^+-Tree* ini diusulkan oleh Christian S. Jensen, Dan Lin, dan Beng Chin Ooi.

V. Kesimpulan

Dalam perkembangan dunia teknologi, pengelolaan basis data menjadi suatu hal yang sangat penting, dan salah satu metode pengelolaan basis data ialah dengan menggunakan suatu struktur data yang disebut *B-Tree* (*Balanced Tree*). Struktur data ini banyak digunakan oleh para pengembang karena kemampuannya dalam mengelola data yang sangat banyak. Dan dalam perkembangannya, struktur data ini banyak digunakan untuk mengatasi suatu permasalahan, baik dengan mengubah struktur data ini maupun tidak.

VI. Daftar Pustaka

1. http://use-the-index-luke.com/sql/anatomy/the-tree, tanggal akses: 13 Desember 2013
2. http://cis.stvincent.edu/html/tutorials/swd/btree/btree.html, tanggal akses 16 Desember 2013
3. http://www.bluerwhite.org/btree/, tanggal akses 14 Desember 2013
4. http://stackoverflow.com/questions/1108/how-does-database-indexing-work, tanggal akses 16 Desember 2013
5. Wu, Chin-Hsien., Kuo, Tei-Wei., and Chang, Li Ping. 2007. *An Efficient B-Tree Layer Implementation for Flash-Memory Storage Systems.*
6. Jensen, Christian S., Dan, Lin., and Ooi, Beng Chin. 2004. *Query and Update Efficient B^+-Tree Based Indexing of Moving Object.*
7. Jagadish, H.V., Ooi, Ben Chin., Tan, Kian Lee., Yu, Cui., and Zhang, Rui. 2005. *iDistance: An Adaptive B^+-Tree Based Indexing Method for Nearest Neighbor Search.*
8. Ferragina, Paolo., and Grossi, Roberto. 1999. *The String B-Tree: A New Data Structure for String Search in External Memory and Its Applications.*
9. Gambar 1 : http://use-the-index-luke.com/img/fig01_02_tree_structure.en.svg
10. Gambar 2 : http://use-the-index-luke.com/img/fig01_03_tree_traversal.en.svg

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Desember 2013

Aldyaka Mushofan, 13512094