**Tugas 1 IF2240 Basis Data**

**ORGANISASI FILE DENGAN METODE SEKUENSIAL**

oleh

Dewita Sonya Tarabunga / 13515021

Erick Wijaya / 13515057

Kezia Suhendra / 13515063



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2017

***Abstract*—Organisasi file adalah teknik menyimpan data pada file. Organisasi file terbagi menjadi organisasi file dengan metode heap, sekuensial, hash, dan multitable clustering. Organisasi file sekuensial memiliki prinsip untuk menjaga record sesuai urutan search key. Mekanisme operasi file sekuensial adalah fetch, next, insert, update, retrieve, dan reorganize. File sekuensial dapat memanfaatkan pointer dan overflow block untuk meningkatkan performansi insert dan delete. Organisasi file sekuensial cocok digunakan pada penyimpanan data mahasiswa karena setiap proses cenderung dilakukan pada semua record. Kinerja organisasi file sekuensial dapat ditingkatkan dengan menambah indeks sehingga operasi-operasi dapat dilakukan lebih cepat tanpa melalui proses sekuensial.**

**Keywords— organisasi file, file sekuensial, operasi, indeks**

# **I.** **Pendahuluan**

Organisasi file adalah suatu teknik atau cara yang digunakan untuk menyatakan maupun menyimpan data dalam suatu file. Penyimpanan data harus diatur sedemikian rupa agar komputer dapat dengan mudah menemukan kembali data yang tersimpan sesuai dengan kebutuhan. Struktur penyimpanan yang digunakan akan sangat berpengaruh dalam menentukan kecepatan akses dan manipulasi data. Struktur penyimpanan akan berkaitan dengan cara penyimpanan baris-baris data di setiap tabel. Struktur penyimpanan file secara sekuensial adalah cara yang paling umum untuk mengorganisasikan kumpulan data-data dalam sebuah file. Setiap baris data akan disusun secara sekuensial berdasarkan kedatangannya (kronologis). Penyusunan baris-baris data tersebut akan berdampak pada besar atau kecilnya ruang penyimpanan yang dibutuhkan dan performansi pengaksesan data dalam tabel. Struktur penyimpanan sekuensial akan sangat efisien untuk penyimpanan data dalam jumlah yang besar.

# **II.** **Pengorganisasian File Sekuensial**

## **A.** **Prinsip Pengorganisasian Data di dalam File**

File sekuensial merupakan struktur penyimpanan yang menempatkan data dengan urutan tertentu di dalam tabel. Data-data yang ada akan diurutkan berdasarkan suatu atribut yang dinamakan search key. Data yang memiliki nilai key paling kecil akan disimpan di awal ruang penyimpanan pada tabel meskipun data tersebut disimpan belakangan. Oleh sebab itu, jika terjadi penambahan, pengubahan, maupun penghapusan data maka seluruh data yang ada perlu penataan kembali. Data yang satu dihubungkan dengan data berikutnya dengan menggunakan pointer. Pengorganisasian data di dalam file yang terurut atau sistem sekuensial akan sangat cepat dan efisien ketika jumlah data yang disimpan sangatlah banyak. Struktur penyimpanan ini akan cocok untuk tabel yang sifatnya statis. Selain itu, struktur penyimpanan ini akan lebih baik dari Hash, jika sering digunakan pencarian data kelompok pada suatu tabel.

Sistem penyimpanan ini dapat diterapkan pada penyimpanan data dari mahasiswa yang ada pada sebuah universitas berupa NIM, nama, dan mata kuliah yang diambil dari tiap mahasiswa. Ketika seorang dosen ingin memasukkan nilai yang didapatkan semua mahasiswa pada mata kuliah yang diajarnya, maka dosen tersebut perlu mengakses data dari seluruh mahasiswa yang ada dengan menggunakan NIM dan nama dari mahasiswa yang akan dimasukkan nilainya. Selain itu, dari sistem penyimpanan data mahasiswa dengan menggunakan file sekuensial ini maka dapat mempermudah seorang dosen dalam membuat suatu statistik dari nilai semua mahasiswa yang ada. Berikut ini adalah ilustrasi dari sebuah tabel yang datanya diurutkan berdasarkan NIM (search key) dari mahasiswa yang ada.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NIM** | **Nama** | **Mata Kuliah** | **Nilai** | **Keterangan** |
| 13510001 | Rossana Doonan | Strategi Algoritma | B | Lulus |
| 13510002 | Betsey Merry | Astronomi Lingkungan | AB | Lulus |
| 13510003 | Calvin Kim | Probabilitas & Statistika | C | Tidak Lulus |
| 13510004 | Debbra Garr | Basis Data | A | Lulus |
| 13510005 | Billy Labrum | Ekonomi Teknik | D | Tidak Lulus |

*Tabel 1. Organisasi file sekuensial dengan NIM sebagai search key*

## **B.** **Mekanisme dalam Melakukan Operasi-Operasi**

Ada beberapa operasi yang yang penting dalam basis data, mulai dari mencari record hingga reorganisasi file. Berikut ada operasi-operasi beserta mekanisme dalam melakukan operasi tersebut.

*1.* *Fetching record*

Untuk mendapatkan record, diperlukan search key dari record yang bersangkutan. File sekuensial memiliki search key yang urut, maka untuk mendapatkan record yang diinginkan adalah dengan melakukan traversal satu per satu dari record paling pertama.

2. *Next record*

Record-record pada file sekuensial sudah terurut berdasarkan search key. Maka mekanisme untuk mendapatkan record selanjutnya sangatlah mudah. Pertama, jika file sekuensial sudah terurut secara fisik, maka jelas bahwa record selanjutnya terletak setelah record sekarang. Untuk kasus kedua jika file tidak terurut fisik juga mudah, yaitu dengan mengambil record yang ditunjuk oleh *pointer* dari record saat ini.

3. *Insert record*

Organisasi file sekuensial mendasarkan sistemnya pada urutan. Maka setiap terjadi insertion atau penambahan record baru ke dalam suatu file, satu hal yang harus diperhatikan adalah untuk menjaga urutan, tepatnya urutan dari search key. Terdapat dua mekanisme yang dapat dilakukan untuk melakukan penambahan record. Yang pertama adalah dengan tetap menjaga urutan fisik dari record-record yang terdapat pada file. Dengan cara ini, maka setiap terjadi penambahan record, record baru akan ditambahkan pada akhir file. Setelah itu, file akan di sort sesuai dengan search key nya. Selain itu terdapat cara lain, yaitu dengan tidak memperhatikan urutan fisik dari record. Tetapi setelah memasukkan record baru, record lain yang seharusnya terletak sebelum record baru akan memiliki semacam *pointer* yang menuju ke record baru. Begitu pula dengan record baru, akan memiliki *pointer* yang menunjuk ke record lain yang seharusnya terletak sebelum dia. Cara ini memiliki konsep yang tidak jauh berbeda dengan list. Cara kedua tentu lebih murah daripada cara pertama dilihat dari segi waktu. Namun cara kedua juga memiliki kelemahan, yaitu tidak menjaga urutan fisik dari record.

4. *Update record*

Perubahan record pada file dengan organisasi sekuensial memiliki mekanisme berikut. Pertama record yang akan di update harus dicari terlebih dalam memori blok. Hal ini sudah dibahas kurang lebih di fetch record. Setelah record didapat, lakukan perubahan terhadap record tersebut sesuai yang diinginkan. Lalu, konsep selanjutnya mirip dengan proses penambahan record. Yaitu, jika sistem ingin mempertahankan urutan fisik file, maka setiap melakukan update, lakukan sorting pada seluruh record.

5. *Retrieve record*

Organisasi file sekuensial merupakan metode yang paling efektif dalam hal retrieval record. Untuk mendapatkan suatu record, cukup lakukan proses sekuensial terhadap file sampai record ditemukan. Oleh karena itu, untuk mendapat semua record di dalam file, ambil record dari atas sampai akhir file ditemukan. Dengan kata lain, retrieve file berurutan dari record pertama sampai record terakhir.

6. *Reorganize*

Jika sudah diketahui bahwa record dalam file akan sering diubah, yaitu, akan sering dilakukan insertion dan deletion, maka biasanya akan digunakan sistem sekuensial yang tidak memperhatikan urutan fisik record, namun tetap memperhatikan urutan *pointer*. Hal ini menyebabkan urutan fisik record pada file tersebut susah dipertahankan. Sehingga lama kelamaan file tersebut perlu untuk direorganisasi untuk mengembalikan urutan fisik record nya. Proses reorganisasi memiliki kaitan yang erat dengan proses sorting. Jika digambarkan dengan pemrograman, algoritma tercepat untuk melakukan sorting adalah O(n log(n)). Jika dilakukan dalam organisasi file dengan record yang banyak, dapat dilihat bahwa proses reorganisasi merupakan proses yang mahal. Sehingga best practice nya adalah dengan melakukan reorganisasi hanya ketika ukuran file tidak terlalu besar.

# **III.** **Performansi dan Analisis Organisasi File Sekuensial**

## **A.** **Performansi Akses Berdasarkan Parameter**

File sekuensial didesain untuk memroses record dengan mudah dengan menjaga keterurutan record berdasarkan *search key* yang ditentukan. Secara umum ada dua jenis file sekuensial yang dikenal, yaitu file sekuensial yang memanfaatkan *pointer* dan *overflow block*, dan file sekuensial yang tidak menggunakan *pointer* (terurut secara fisik). Kedua jenis file sekuensial ini memiliki performansi yang berbeda dalam mencari, menambahkan, dan menghapus record.

Performansi organisasi file sekuensial, dilihat dari parameter untuk melakukan pencarian (*searching*), dapat dinilai dengan melihat seberapa cepat pencarian berlangsung. Apabila file sekuensial tidak menggunakan *pointer* (memerhatikan urutan secara fisik), proses pencarian dapat dilakukan dengan cepat karena *binary search* dapat diimplementasikan dalam organisasi file tersebut sehingga tidak menguras waktu. Karena *binary search* menggunakan prinsip *divide and conquer* dengan membagi-bagi record, proses pencarian dapat berlangsung dalam O(log n). Namun, apabila organisasi file sekuensial yang digunakan memanfaatkan *pointer*, proses pencarian record haruslah linear dengan kompleksitas O(n) karena file harus diakses dari awal dan dijelajahi hingga ditemukan record yang diinginkan atau hingga mencapai *null pointer* (akhir file). Hal ini tentu dapat membuat performa menjadi lambat bila proses pencarian menjadi hal yang sering dilakukan dalam organisasi file sekuensial tersebut. Akan tetapi, penggunaan *pointer* menguntungkan dalam parameter lain terutama dalam menambahkan record.

Parameter berikutnya adalah operasi menambahkan record (*insert*)pada file sekuensial. Operasi penambahan record pada file sekuensial yang tidak menggunakan *pointer* akan memiliki performa yang buruk, karena setiap penambahan record harus disertai dengan pengurutan sehingga keterurutan file tetap terjaga. Hal ini bearti ketika record mau disisipkan pada file sekuensial, record tersebut akan “digeser” sedemikian rupa hingga ditemukan posisi yang benar sesuai dengan keterurutan file. Hal ini buruk karena setiap “penggeseran” record memerlukan record untuk di-*overwrite* sehingga performa menjadi sangat buruk untuk data yang besar. Proses penggeseran membutuhkan kompleksitas O(n) hingga ditemukan posisi yang benar untuk menyisipkan record baru. Ada alternatif yang dapat dilakukan untuk meningkatkan performa, yaitu dengan memanfaatkan *pointer*. Hal ini dapat dilakukan pada organisasi file sekuensial yang menggunakan *pointer*, yaitu dengan menyediakan *overflow block*. Blok tersebut akan dipakai apabila pada saat penyisipan record tidak ditemukan blok kosong (*free space*). Apabila ada blok kosong tepat sebelum blok record yang adalah blok selanjutnya dari yang mau disisipkan sesuai urutan, maka record akan diisi pada blok tersebut. Proses pencarian posisi untuk menambah record membutuhkan kompleksitas O(n) namun proses penyisipan record hanya membutuhkan kompleksitas O(1). Walaupun kompleksitasnya sama saja dengan file sekuensial tanpa *pointer*, performa penambahan record menjadi lebih baik karena tidak terjadi proses *overwrite* yang mahal, yang terjadi adalah mencari posisi berdasarkan *search key*. Karena menggunakan *overflow block*, secara periodik record yang tersimpan pada blok tersebut akan disatukan pada file utama.

Parameter terakhir yang menentukan performansi organisasi file adalah parameter menghapus record (*delete*). Karena proses penghapusan record membutuhkan proses pencarian terlebih dahulu, perfoma organisasi file dalam menghapus record dapat juga dilihat dari performanya ketika melakukan pencarian record. Untuk organisasi file sekuensial yang memerhatikan urutan secara fisik (tanpa *pointer*), performa pencarian record yang mau dihapus sangat baik karena *binary search*, akan tetapi setelah record dihapus, setiap record harus “digeser” supaya keterurutan record secara fisik tetap terjaga dengan baik. Dalam hal ini performa penghapusan record menjadi buruk karena “penggeseran” record merupakan operasi yang mahal untuk record yang mengandung banyak data. Apabila performa ini dibandingkan dengan file sekuensial yang menggunakan *pointer*, pencarian record untuk dihapus membutuhkan kompleksitas linier tetapi penghapusan record sangat cepat karena penghapusan dapat dilakukan dengan mengubah *pointer* pada record sebelumnya untuk menunjuk pada record setelah record yang mau dihapus. Karena hanya melibatkan proses pencarian yang linier dan penghapusan yang cepat, performa dapat dikatakan baik.

Ketiga parameter yang telah dibahas dapat menunjukkan performa organisasi file sekuensial, namun belum dibandingkan dengan organisasi file lain seperti *heap*, *hash*, dan *multitable clustering*. Pada parameter pencarian record, organisasi file dengan *heap* membutuhkan waktu linier karena record tidak terurut sehingga performanya dapat dikatakan lambat. Berbeda dengan *heap* dan *sequential*, *hash* memiliki performa yang baik karena fungsi *hash* menghitung indeks untuk menemukan indeks pada *bucket*. Pada *multitable clustering* pencarian juga dilakukan secara linier. Kemudian bila dilihat dari parameter penambahan record, *heap* dan *hash* memiliki performa yang sangat baik. Pada *heap* record langsung ditambah di bagian belakang file sedangkan pada *hash*, indeks dihitung dengan fungsi *hash* untuk menentukan posisi record tersebut. Terakhir, dilihat dari parameter menghapus record, *heap* memiliki performa yang buruk karena selain harus mencari secara linier, penghapusan menyebabkan adanya lubang pada organisasi file. Untuk *hash*, performa berjalan dengan baik karena memanfaatkan fungsi *hash*. Apabila semua organisasi file tersebut dibandingkan dari parameter-parameter, performa dapat dirangkum pada tabel berikut.

|  |  |  |  |
| --- | --- | --- | --- |
| **Organisasi File** | **Parameter Performa** | | |
| **Search** | **Insert** | **Delete** |
| Heap | Kurang Baik | Kurang Baik | Buruk |
| Sequential tanpa ptr | Baik | Buruk | Buruk |
| Sequential dengan ptr | Kurang Baik | Kurang Baik | Kurang Baik |
| Hash | Baik | Baik | Baik |
| Multitable Clustering | Baik | Kurang Baik | Kurang Baik |

*Tabel 2. Performansi Organisasi File dengan Parameter*

## **B.** **Analisis Kapan Sebaiknya Organisasi File Sekuensial Digunakan**

Organisasi file sekuensial merupakan salah satu metode paling sederhana dibandingkan dengan metode lain seperti heap, hash, atau multitable clustering. Metode lain cenderung memiliki sistem yang lebih rumit dalam upaya menyimpan datanya, walaupun dari segi waktu akan memberikan hasil yang lebih cepat dibanding file sekuensial. Selain itu, file dengan organisasi sekuensial dapat disimpan pada magnetic tape, sehingga lebih murah dari segi biaya. Namun selain kelebihan, organisasi file sekuensial juga memiliki kelemahan. Kelemahan-kelemahan dari organisasi file sekuensial sudah dibahas di bagian performansi akses, dimana proses insertion dan deletion dari sistem ini membutuhkan waktu yang relatif lebih lama dibanding metode lain.

Oleh karena itu, dapat dilihat bahwa metode ini sangat cocok digunakan untuk suatu data dimana akan jarang terdapat proses perubahan baik insertion, deletion, maupun updating suatu record dalam file. Terdapat banyak contoh jenis data yang seperti itu, contohnya yang sudah dipaparkan di bagian 1, yaitu penyimpanan data mahasiswa pada suatu universitas dengan NIM merupakan *search-key*. Data mahasiswa jarang akan terdapat perubahan selain di awal ketika data pertama kali dimasukkan. Hal ini sering terdapat pada suatu data dimana search key nya ordinal, sehingga pendekatan paling intuitif adalah menggunakan organisasi file sekuensial.

Selain itu, salah satu hal yang akan menyita waktu pada organisasi file sekuensial adalah proses pencarian search key. Karena itu, metode ini juga lebih cocok diterapkan pada suatu file penyimpanan dimana proses pencarian jarang dilakukan, yaitu ketika setiap proses cenderung akan dilakukan pada setiap record dalam file. Contohnya pada data mahasiswa, proses yang sering dilakukan adalah input nilai, dan penghitungan nilai, dimana hal tersebut seharusnya dilakukan pada hampir seluruh record (mahasiswa). Selain itu, contoh lain adalah data pegawai beserta gajinya. Karena itu, metode ini cocok digunakan untuk data yang akan sering diterapkan penghitungan statistik, karena penghitungan statistik harus dilakukan terhadap setiap record dalam file.

# **IV.** **Peningkatan Kinerja Organisasi File Sekuensial dengan Indeks**

Struktur penyimpanan file indeks sekuensial merupakan struktur gabungan dari file sekuensial dengan file yang memiliki indeks. Struktur penyimpanan ini akan membuat pencarian data menjadi lebih mudah. Indeks disusun atau diurutkan berdasarkan nilai key pada setiap data yang ada dalam struktur penyimpanan. Indeks ini digunakan untuk mencatat posisi atau alamat dari suatu data di dalam file. Indeks ini akan berguna untuk membantu mempercepat pencarian data dalam suatu tabel karena data dapat langsung di akses pada indeks yang bersesuaian. Selain itu, pada struktur penyimpanan ini tetap dapat dilakukan pencarian dengan sekuensial maupun kombinasi dari pencarian sekuensial dan pencarian langsung berdasarkan indeks yang bersesuaian. Dibandingkan dengan struktur penyimpanan file sekuensial, struktur penyimpanan ini menjadi lebih cepat dalam hal pemanggilan data yang ada. Pada struktur penyimpanan file sekuensial yang memiliki indeks akan menjadi mudah dalam hal penambahan, penghapusan, maupun pengubahan data yang ada karena hal ini dapat dilakukan secara langsung melalui indeksnya tanpa perlu dilakukan proses sekuensial terlebih dahulu.

##### **V. Kesimpulan**

Berdasarkan analisis organisasi file sekuensial yang sudah dibahas, disimpulkan bahwa organisasi file sekuensial memiliki prinsip untuk menyimpan data dengan menjaga keterurutan data sesuai ­*search key*. Organisasi file sekuensial memiliki mekanisme operasi dimulai dari *fetching record* hingga melakukan reorganisasi file. Organisasi file sekuensial memiliki performa yang lemah untuk melakukan *insert* dan *delete*. Dengan memanfaatkan *pointer* dan *overflow block*, performa file sekuensial dapat ditingkatkan. Selain itu, menambahkan indeks pada organisasi file sekuensial dapat meningkatkan kinerja karena pencarian beserta pengubahan data dapat dilakukan lebih cepat tanpa melalui proses sekuensial. Organisasi file sekuensial cocok digunakan pada data mahasiswa karena setiap proses cenderung dilakukan pada semua record.

##### **Ucapan Terima Kasih**

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya yang melimpah dalam hidup penulis sehingga penulis dapat menyelesaikan makalah ini. Penulis juga ingin mengucapkan terima kasih kepada Ibu Gusti Ayu Putri Saptawati selaku dosen mata kuliah Basis Data. Penulis juga mengucapkan terima kasih kepada orang tua dan keluarga penulis yang senantiasa mendoakan dan mendukung studi penulis.

##### **Referensi**

[1] Silberschatz, Abraham. *Database System Concepts*. New York: McGraw-Hill. 2011.

[2] http://www.ibm.com/support/knowledgecenter/en/SS6SGM\_4.1.1/com.ibm.cobol.aix.doc/PGandLR/concepts/cpfio05.html

[3] https://www.tutorialcup.com/dbms/sequential-file-organization.htm

[4] http://informatika.web.id/struktur-penyimpanan-tabel.htm

[5] http://www.slideshare.net/Gokul017/file-organization-41344017

[6] http://elib.unikom.ac.id/files/disk1/370/jbptunikompp-gdl-diandharma-18489-6-vii-file-l.pdf

Bandung, 7 Februari 2017

Dewita Sonya Tarabunga - 13515021

Erick Wijaya – 13515057

Kezia Suhendra – 13515063