ANALISIS FRAGMENTASI TABEL SECARA VERTIKAL MENGGUNAKAN ALGORITMA *BOND ENERGY* DALAM HUBUNGANNYA DENGAN KECEPATAN EKSEKUSI QUERY

Made Hanindia Prami Swari[1], Ngurah Agus Sanjaya ER[2]
Program Studi Teknik Informatika, Jurusan Ilmu Komputer,
Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Udayana
Email: hanindia.prami@cs.unud.ac.id[1], agus.sanjaya@cs.unud.ac.id[2]

ABSTRAK

Kecepatan eksekusi *query* pada suatu basis data dengan jumlah atribut dan baris yang besar, sangat bergantung pada jumlah atribut yang diakses. *Query* yang optimal hanya akan mengakses atribut yang diperlukan. Fragmentasi tabel secara vertikal dapat membantu untuk memisahkan antara atribut yang sering diakses dengan yang jarang digunakan.

Pada penelitian ini dikembangkan suatu aplikasi untuk membandingkan waktu eksekusi *query* pada tabel yang normal dengan tabel yang difragmentasi secara vertikal. Fragmentasi vertikal pada tabel dibentuk dengan menggunakan algoritma *Bond Energy*. Fragmentasi yang dilakukan harus memenuhi kondisi kelengkapan dan *disjointness*.

Uji coba aplikasi dilakukan pada 40 *query* yang terdapat pada Sistem Informasi Perencanaan Universitas Udayana. Hasil uji coba memperlihatkan bahwa fragmentasi vertikal mempercepat waktu eksekusi pada *query* yang melibatkan beberapa atribut pada tabel dan *query* dengan operasi aritmatika seperti *sum* atau rata-rata. Sedangkan pada *query insert* dan *update* waktu eksekusinya cenderung tidak berubah.

Kata kunci: fragmentasi vertikal, *Bond Energy*

ABSTRACT

The required time to run a query on a database which possesses a large number of attributes and rows highly depends on the number of attributes accessed. An optimal query will only access necessary attributes. A vertical fragmentation of a table helps in isolating the frequently accessed attributes with those which are less frequently used.

In this research, we develop an application to help the comparison of query execution time on a normal and vertically fragmented table. We use a Bond Energy algorithm to form the table's vertical fragmentation. In this algorithm, the fragmentation process has to fulfill the completeness and disjointness conditions.

We perform and test this application by using 40 queries from "Sistem Informasi Perencanaan Universitas Udayana". Our test results show that a vertical fragmentation using Bond Energy algorithm increases the speed of query execution on the following: queries which access only several attributes from a table and queries with arithmetic operations such as sum or average. For insert and update queries, vertical fragmentation has no significant impact.

***Keywords**: vertical fragmentation, Bond Energy*

1. Pendahuluan

Semakin besar dan kompleksnya aliran data pada sebuah sistem informasi dapat menyebabkan suatu sistem informasi mengalami penurunan performa terutama pada waktu pengaksesan informasi yang dibutuhkan. Hal ini disebabkan fungsi pengaksesan data *(query)* akan mencari dan mengambil data dari banyak kolom (atribut) pada tabel basis data, padahal tidak semua atribut yang terdapat pada tabel tersebut dibutuhkan. Metode yang dapat digunakan untuk meningkatkan waktu pemrosesan *query* adalah dengan melakukan fragmentasi tabel menjadi beberapa fragment berdasarkan tingkat keseringan aksesnya.

Untuk melakukan fragmentasi secara vertikal pada basis data yang akan diuji, maka dilakukan pembuatan aplikasi yang diberi nama aplikasi Planarian. Melalui aplikasi ini, fragmentasi vertikal akan

dilakukan secara otomatis berdasarkan hasil perhitungan dari algoritma *bond energy*. Setelah dilakukan proses fragmentasi terhadap tabel basis data yang akan diuji, maka dapat diuji waktu pemrosesan masing-masing *query* yang ada..Tujuan dari hasil dari implementasi algoritma fragmentasi *bond energy* ini adalah untuk membandingkan kecepatan akses data pada basis data yang telah difragmentasi dengan basis data sebelum difragmentasi.

2. Tinjauan Pustaka

2.1 Fragmentasi Tabel

Fragmentasi merupakan sebuah proses pengambilan bagian-bagian baris ataupun kolom dari tabel-tabel sebagai unit data terkecil yang akan dikirimkan melalui jaringan komputer. Sedangkan fragmentasi data merupakan proses dimana basis data akan dipecah-pecah kedalam unit-unit logika yang disebut fragment yang kemudian akan disimpan dalam site yang berbeda (Rob, 2007). Fragmentasi data tergabung dalam proses desain basis data, yang kemudian akan menentukan apa yang kemudian akan dilakukan terhadap fragment yang telah dibuat.

Alasan-alasan diperlukannya fragmentasi, yaitu (Connolly, 2010):

1. Penggunaan

Umumnya aplikasi bekerja dengan *table views* dibandingkan dengan semua hubungan data. Oleh karenanya untuk distribusi data, yang cocok digunakan adalah bekerja dengan subset dari sebuah relasi sebagai unit dari distribusi.

2. Efisiensi

Data akan disimpan dekat dengan aplikasi yang mengaksesnya dan tambahan data yang tidak sering diakses tidak akan disimpan.

3. Pararelisme

Dengan fragment-fragment tersebut sebagai unit dari suatu distribusi, sebuah transaksi dapat di bagi kedalam beberapa sub *query* yang dioperasikan pada fragment tersebut. Hal ini meningkatkan konkurensi atau paralelisme dalam sistem, sehingga memeperbolehkan transaksi dieksekusi secara aman dan paralel.

4. Keamanan

Data yang tidak dibutuhkan oleh aplikasi tidak disimpan dan hanya pengguna yang memiliki hak akses saja yang dapat mengakses data yang ada.

Sedangkan kerugian fragmetasi yaitu:

1. Kinerja

Cara kerja dari aplikasi yang membutuhkan data dari beberapa lokasi fragment di beberapa situs akan berjalan dengan lambat.

2. Integritas

Pengawasan integritas akan lebih sulit jika data dan *functional dependency* di fragmentasi dan dilokasi pada beberapa situs yang berbeda.

Beberapa peraturan yang harus diidentifikasikan ketika mendefinisikan fragment (Connolly, 2010) :

1. Kondisi lengkap

Jika relasi contoh R di dekomposisi ke dalam fragment R_1, R_2, R_3, … R_n, masing-masing data yang dapat ditemukan pada relasi R harus muncul paling tidak di salah satu fragment. Aturan ini di perlukan untuk meyakinkan bahwa tidak ada data yang hilang selama fragmentasi.

2. *Disjointness*

Jika item data d_i muncul pada fragment R_i, maka tidak boleh muncul di fragment yang lain. Fragmentasi vertikal diperbolehkan untuk aturan yang satu ini, dimana kunci utama dari atribut harus diulang untuk melakukan rekonstruksi. Aturan ini untuk meminimalkan redudansi.

2.2 Algoritma *Bond Energy*

Algoritma *Bond Energy* atau *Bond Energy Algorithm* (BEA) merupakan algoritma yang sering digunakan dalam proses fragmentasi vertikal pada basis data terdistribusi (Ray, 2009). Algoritma ini diusulkan oleh McCormick, Hoffer dan Severande. BEA terdiri dari dua algoritma, yang pertama digunakan untuk menempatkan sekumpulan (himpunan) data dengan mengalokasikan elemen yang paling berkaitan secara bersama-sama (dan dipisahkan dengan kumpulan elemen yang

tidak berkaitan), yang kedua merupakan algoritma yang digunakan untuk membuat kelompok yang menentukan point untuk membuat potongan dari sebuah data set (*cluster* yang dibuat). Hal utama yang dilakukan dalam membuat fragmentasi vertikal dalam basis data terditribusi adalah dengan menemukan pengelompokan-pengelompokan atribut dalam tabel relasi berdasarkan nilai affinitas pada matriks affinitas atribut. Matriks affinitas merupakan matriks yang memuat jumlah keterikatan antar satu atribut dengan atribut lainnya (banyaknya pengaksesan dua atribut secara bersamaan). Algoritma ini disarankan untuk digunakan pada frgmentasi tabel karena beberapa alasan sebagai berikut :

1. Algoritma ini didesain secara spesifik untuk mengelompokkan item-item yang mempunyai sifat yang sama (pengelompokan dalam *cluster* yang memiliki nilai affinitas yang besar dan pengelompokan yang lainya berupa atribut dengan nilai affinitas yang kecil).
2. Pengelompokkan fragment final bersifat *insensitive* terhadap pengelompokan lainnya sebagai hasil dari algoritma. Maksud dari *insensitive* ini adalah urutan atribut-atribut yang terlibat dianggap sama. Misalnya fragment a_1, a_2, a_3 | a_4 dan fragment a_1, a_3, a_2 | a_4 dianggap merupakan fragment yang sama.
3. Jumlah atribut yang dapat difragment ini tidak dibatasi sehingga perbandingan waktu eksekusi *query* yang akan diuji akan terlihat perbedaanya. Selain itu jumlah fragment yang dihasilkan pada fragmentasi ini tidak bersifat mutlak sehingga dapat secara pasti menemukan fragment yang terbaik.

Adapun langkah-langkah pada algoritma ini adalah sebagai berikut (Ray, 2009) :

Bentuk n x n affinitas matriks yang akan dijadikan matriks dasar pada proses fragmentasi tabel yang akan dilakukan. Setelah itu lakukan langkah-langkah berikut :

- Initialization : ambil 1 kolom dan letakkan pada kolom pertama di matriks outputnya.
- Iteration step i : letakkan n-i kolom yang tersisa pada posisi i+1 yang memungkinkan pada matriks output yang akan menyebabkan kontribusi terbesar pada perhitungan affinitas.
- Row ordering : pada tahap ini, baris-baris akan diatur sama seperti pengaturan kolom. Kontribusi dari kolom Ak, yang diletakkan antara Ai dan Aj dapat direpresentasikan sebagai berikut :

Cont(Ai, Ak, Aj)=bond(Ai, Ak) + bond(Ak,Aj) – bond(Ai, Aj)

dimana,

$$\textbf{bond (Ax, Ay)} = \sum_{z=1}^{n} \textbf{aff (Ax, Az)}$$

aff (Az, Ay)

Langkah selanjutnya adalah hitung banyak pengaksesan yang dilakukan pada masing-masing fragment yang terbentuk, lalu hitung nilai *maximaze split quality* dari masing-masing fragment. Nilai *Maximaze split quality* dapat dihitung menggunakan rumus :

Mazimaze split quality sq = acc (VF1) * acc(VF2) – acc (VF1, VF2)^2

Berikut ini merupakan contoh fragmentasi tabel secara vertikal pada sebuah kasus :
A = (A_1, A_2, A_3, A_4, A_5) merupakan atribut-atribut yang terdapat pada tabel, sedangkan Q = (Q_1, Q_2, Q_3, Q_4) merupakan *query-query* pengaksesan data. Berikut ini merupakan matriks pengaksesan atribut dari masing-masing *query* yang ada :

Tabel 1. Matriks Pengaksesan Atribut

	A_1	**A_2**	**A_3**	**A_4**	**A_5**
Q_1	1	1	1	0	0
Q_2	0	0	1	1	0
Q_3	0	1	0	1	1
Q_4	0	0	1	0	1

Sedangkan tabel 2 merupakan tabel jumlah pengaksesan data masing-masing *query*

pada masing-masing *site* yang ada. Dari tabel 2, didapatkan jumlah pengaksesan atribut pada seluruh *site* adalah sebagai berikut :

Q_1 : A1 A2 A3 21
Q_2 : A3 A4 24
Q_3 : A2 A4 A5 90
Q_4 : A3 A5 11

Tabel 2. Tabel Pengaksesan Data Masing-Masing *Site*

	A	**B**	**C**	**Sum**
Q_1	20	1	0	21
Q_2	10	5	9	24
Q_3	80	15	9	90
Q_4	2	5	4	11

Maka, didapatkan 5 x 5 matriks affinitas sebagai berikut :

Tabel 3. Matriks Affinitas

	A_1	**A_2**	**A_3**	**A_4**	**A_5**
A_1	21	21	21	0	0
A_2	21	111	21	90	90
A_3	21	21	56	24	11
A_4	0	90	24	114	90
A_5	0	90	11	90	101

Setelah itu ambil dua kolom secara acak pada kolom matriks affinitasnya dan lakukan perhitungan nilai kontribusi, seperti berikut :

1. Pilih A_1

- Kontribusi A_1 bila diletakkan pada posisi 0 adalah : [_, A_1, A_5]
  Cont(_, A_1, A_5) = bond(_, A_1) + bond(A_1, A_5) – bond(_, A_5) = 0 + ((21 * 0)+(21 * 90)+(21 * 11)+(0 * 90)+(0 * 101)) + 0 = **2121**
- Kontribusi A_1 bila diletakkan pada posisi 1 adalah :[A_5, A_1, A_3] Cont(A_5, A_1, A_3) = bond(A_5, A_1) + bond(A_1, A_3) – bond(A_5, A_3) = 2121 + 2058 – 5777= **-1598**
- Kontribusi A_1 bila diletakkan pada posisi 2 adalah : [A_3, A_1, _]
  Cont(A_3, A_1, _) = bond(A_3, A_1) + bond(A_3, _) – bond(A_3, _) = 2058 + 0 - 0= **2058**

Nilai kontribusi A_1 terbesar adalah ketika A_1 diletakkan pada posisi 0, maka hasil pengurutan yang didapat adalah sebagai berikut : **[A_1, A_5, A_3]**

2. Pilih A_2

- Kontribusi A_2 bila diletakkan pada posisi 0 adalah : [_, A_2, A_1]
  Cont(_, A_2, A_1) = bond(_, A_2) + bond(A_2, A_1) – bond(_, A_1) = 0 + 3213 + 0 = **3213**
- Kontribusi A_2 bila diletakkan pada posisi 1 adalah : [A_1, A_2, A_5] Cont(A_1, A_2, A_5) = bond(A_1, A_2) + bond(A_2, A_5) – bond(A_1, A_5) = 3213 + 27411 – 2121 = **28503**
- Kontribusi A_2 bila diletakkan pada posisi 2 adalah : [A_5, A_2, A_3] Cont(A_5, A_2, A_3) = bond(A_5, A_2) + bond(A_2, A_3) – bond(A_5, A_3) = 27411 + 7098 – 5777 = **28732**
- Kontribusi A_2 bila diletakkan pada posisi 3 adalah : [A_3, A_2, _]
  Cont(A_3, A_2, _) = bond(A_3, A_2) + bond(A_2, _) – bond(A_3, _) = 2058 + 0 – 0 = **7098**

Nilai kontribusi A_2 terbesar adalah ketika A_2 diletakkan pada posisi 2, maka hasil pengurutan yang didapat adalah sebagai berikut : **[A_1, A_5, A_2, A_3]**

3. Pilih A_4

- Kontribusi A_4 bila diletakkan pada posisi 0 adalah : [_, A_4, A_1]
  Cont(_, A_4, A_1) = bond(_, A_4) + bond(A_4, A_1) – bond(_, A_1) = 0 + 2394 + 0 = **2394**
- Kontribusi A_4 bila diletakkan pada posisi 1 adalah : [A_1, A_4, A_5]Cont(A_1, A_4, A_5) = bond(A_1, A_4) + bond(A_4, A_5) – bond(A_1, A_5) = 2394 + 27714 – 2121 = **27987**
- Kontribusi A_4 bila diletakkan pada posisi 2 adalah : [A_5, A_4, A_2] Cont(A_5, A_4, A_2) = bond(A_5, A_4) + bond(A_4, A_2) – bond(A_5, A_2) = 27714 + 28854 – 27411 = **29157**
- Kontribusi A_4 bila diletakkan pada posisi 3 adalah : [A_2, A_4, A_3] Cont(A_2, A_4, A_3) = bond(A_2, A_4) + bond(A_4, A_3) – bond(A_2, A_3) = 28854 + 6960 – 7098 = **28716**
- Kontribusi A_4 bila diletakkan pada posisi 4 adalah : [A_3, A_4, _]

Cont(A_3, A_4, _) = bond(A_3, A_4) + bond(A_4, _) – bond(A_3, _) = 6960 + 0 – 0 = **6960**

Nilai kontribusi A_4 terbesar adalah ketika A_4 diletakkan pada posisi 2, maka hasil pengurutan yang didapat adalah sebagai berikut : **[A_1, A_5, A_4 A_2, A_3].**

Maka, dari proses pengurutan diatas, didapat pengurutan dengan nilai kontribusi terbesar adalah : **[A_1, A_5, A_4 A_2, A_3].**

Setelah, proses pengurutan selesai, langkah selanjutnya adalah menghitung banyak pengaksesan yang dilakukan pada masing-masing fragment yang terbentuk, lalu hitung nilai *maximaze split quality* dari masing-masing fragment sebagai berikut :

1. Jika fragmentasi dilakukan pada : **[A_1,A_2, A_3, A_4] | [A_5]**

Pengaksesan frag1 : 45
Pengaksesan frag2 : 0
Pengaksesan frag1 dan frag2 : 101
Split quality : (45 *0) – (101^2) = **-10201**

2. Jika fragmentasi dilakukan pada : **[A_1,A_2, A_3] | [A_4, A_5]**

Pengaksesan frag1 : 21
Pengaksesan frag2 : 0
Pengaksesan frag1 dan frag2 : 125
Split quality : (21 *0) – (125^2) = **-15625**

3. Jika fragmentasi dilakukan pada : **[A_1, A_3] | [A_2, A_4, A_5]**

Pengaksesan frag1 : 0
Pengaksesan frag2 : 90
Pengaksesan frag1 dan frag2 : 56
Split quality : (*90) – (56^2) = **-3136**

4. Jika fragmentasi dilakukan pada : **[A_1] | [A_2, A_3, A_4, A_5]**

Pengaksesan frag1 : 0
Pengaksesan frag2 : 125
Pengaksesan frag1 dan frag2 : 21
Split quality : (0 *125) – (21^2) = **-441**

5. Jika fragmentasi dilakukan pada : **[A_1,A_2, A_3, A_5] | [A_4]**

Pengaksesan frag1 : 32
Pengaksesan frag2 : 0
Pengaksesan frag1 dan frag2 : 114
Split quality : (32 *0) – (114^2) = **-12996**

6. Jika fragmentasi dilakukan pada : **[A_1, A_3, A_5] | [A_2, A_4]**

Pengaksesan frag1 : 11
Pengaksesan frag2 : 0
Pengaksesan frag1 dan frag2 : 135
Split quality : (11 *0) – (135^2) = **-18225**

7. Jika fragmentasi dilakukan pada : **[A_1, A_5] | [A_2, A_3, A_4]**

Pengaksesan frag1 : 0
Pengaksesan frag2 : 24
Pengaksesan frag1 dan frag2 : **122**
Split quality : (0 *24) – (122^2) = **-14884**

8. Jika fragmentasi dilakukan pada : **[A_1,A_3, A_4, A_5] | [A_2]**

Pengaksesan frag1 : 35
Pengaksesan frag2 : 0
Pengaksesan frag1 dan frag2 : 111
Split quality : (35 *0) – (111^2) = **-12321**

9. Jika fragmentasi dilakukan pada : **[A_1, A_4, A_5] | [A_2, A_3]**

Pengaksesan frag1 : 0
Pengaksesan frag2 : 0
Pengaksesan frag1 dan frag2 : 146
Split quality : (0 *0) – (146^2) = **-21316**

10. Jika fragmentasi dilakukan pada : **[A_1,A_2, A_4, A_5] | [A_3]**

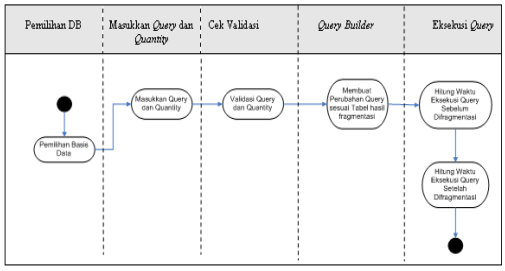
Pengaksesan frag1 : 90
Pengaksesan frag2 : 0
Pengaksesan frag1 dan frag2 : 56
Split quality : (90 *0) – (56^2) = **-3136**

Berdasarkan hasil perhitungan split quality di atas, maka fragmentasi optimal yang dihasilkan adalah **sq = -441** yaitu pada fragmentasi :

[A_1] | [A_2, A_3, A_4, A_5]

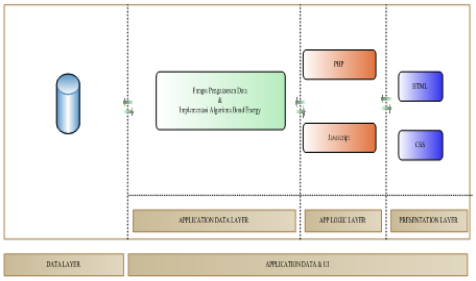
3. Rancangan Aplikasi Planarian menggunakan Algoritma *Bond Energy*

Secara umum Aplikasi Planarian dibagi ke dalam lima proses utama, yaitu Input *Query* dan *Quantity*, Cek Validitas, Proses *Bond Energy Algorithm*, Fragmentasi Tabel, Hitung Estimasi. Kelima proses tersebut dapat dilihat dari aktivitas diagram pada gambar 1.



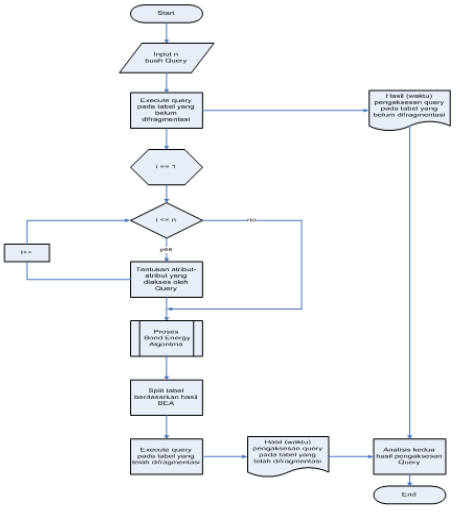
Gambar 1. Diagram Aktivitas

Adapun arsitektur dari sistem dapat dilihat pada gambar 2.



Gambar 2. Arsitektur Sistem

Secara umum alur dari sistem yang dibangun adalah seperti pada gambar 3.



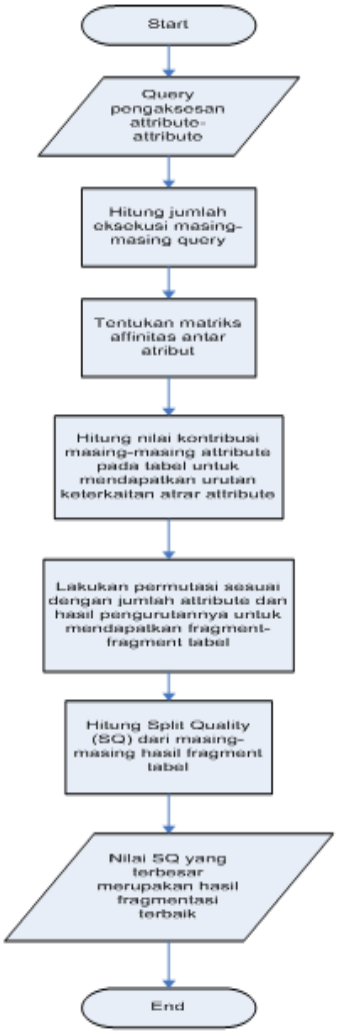
Gambar 3. Diagram Alir Sistem

Diagram alir untuk proses algoritma *Bond Energy* digambarkan secara terpisah pada gambar 4.

4. Implementasi dan Uji Coba

4.1 Implementasi

Pada tahapan awal dilakukan pengkodean bahasa pemrograman untuk menghasilkan aplikasi berbasis web yang diinginkan. Bahasa pemrograman yang dipakai adalah PHP, HTML, dan Javascript. Alur logika fragmentasi yang dibuat harus disesuaikan dengan pendekatan algoritma *bond energy* sehingga nantinya akan dihasilkan suatu aplikasi yang dapat melakukan fragmentasi tabel secara otomatis terhadap sekumpulan *query* yang akan dieksekusi pada suatu sistem informasi.



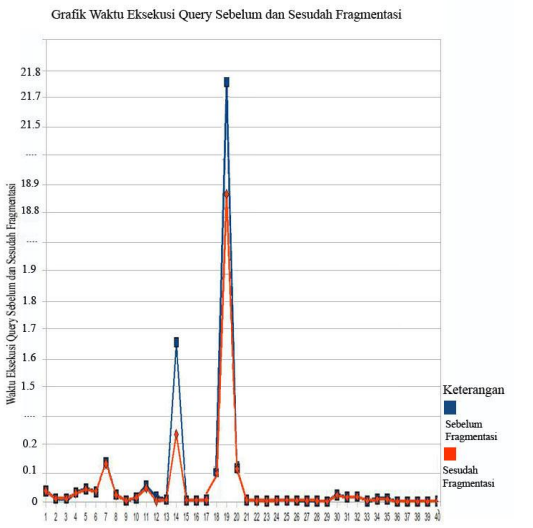
Gambar 4. Diagram Alir Proses *Bond Energy*

4.2 Uji Coba

Aplikasi yang dibuat diuji coba pada Sistem Informasi Perencanaan Universitas Udayana. Aplikasi yang dibuat sudah memenuhi syarat-syarat fragmentasi, diantaranya tidak terjadi kerangkapan data (kecuali *primary key*) dan suatu data minimal harus muncul pada satu tabel fragmentasi.

Pada satu kali proses transaksi yang dijalankan, terdapat sebanyak 360 buah *query* yang dijalankan. Namun dalam

penelitian ini dipilih 40 buah *query* yang paling sering diakses pada transaksi tersebut dan *query-query* yang dipilih terdiri dari mewakili *query* dasar (*insert*, *update*, *delete*, *select*), selain itu *query* yang dipilih juga merupakan *query* yang memiliki waktu eksekusi terbesar, sehingga diharapkan hasil perbandingan waktu eksekusi *query* pada tabel yang belum difragmentasi dan yang telah difragmentasi dengan menggunakan algoritma *bond energy* ini dapat terlihat lebih jelas. Perbandingan waktu eksekusi dapat dilihat pada gambar 5.



Gambar 5. Grafik Perbandingan Waktu Eksekusi Sebelum dan Sesudah Fragmentasi

5. Kesimpulan

Melakukan fragmentasi secara vertikal terhadap tabel-tabel pada basis data tidak selalu mempercepat waktu eksekusi *query*. Secara umum, fragmentasi tabel secara vertikal cenderung akan mempercepat waktu eksekusi pada *query select* yang hanya mengakses beberapa atribut pada suatu tabel dan *query* yang melibatkan proses aritmatika seperti *sum* atau rata-rata. Sedangkan untuk *query insert* dan *update*, waktu eksekusinya cenderung tidak berubah.

Daftar Pustaka

(1) Connoly, Thomas. 2010. *Database System A Practical Approach to Design, Implementation, and Managemen Fifth Editiont.* New Jersey: Pearson Education, Inc

(2) Hammer, M., and Niamir, B. *A heuristic approach to* atribute *partitioning. In Proceedings ACM SZGMOD International Conference on Management of Data* (Boston, Mass., 1979), ACM:New York.

(3) Kroenke, David. 2010. *Database Concepts Fourth Edition.* New Jersey : Pearson Education, Inc

(4) Navathe, Shamkant, Ceri, Stefano, Wiederhold, Gio, and Dou, Jinglie. 1984. *"Vertical Partitioning for Database Design".* ACM Transactionson Database Systems Vol. 9, No. 4, December 1984.

(5) Ray, Chhanda. 2009. *Distributed Database System.* India: Dorling Kindersley

(6) Ritchie, Colin. *Database Principles and Design Third Edition.* China: C&C Offset Printing Co Ltd.

(7) Rob, Peter. 2007. *Database Systems Design, Implementation, and Management Seventh Edition.* Canada:Thomson Learning, Inc

[Halaman ini sengaja dikosongkan]