Analisis Efisiensi Operasi Natural Join pada Model Data Relasional Menggunakan Algoritma *Divide and Conquer*

Johann Christian Kandani - 13521138
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521138@std.stei.itb.ac.id

***Abstrak*—Pengolahan data pada sistem manajemen basis data sering berhadapan dengan persoalan efisiensi ruang. Penggunaan ruang yang efisien memungkinkan penyimpanan dan pemrosesan volume data besar, namun keterbatasan ruang selain dapat memperlambat pemrosesan data, dapat mengakibatkan pemrosesan data tertentu tidak dapat dilakukan. Sistem manajemen basis data yang menerapkan model data relasional dapat mengalami kendala tersebut, dan salah satu penyelesaian dari persoalan tersebut adalah dengan mengganti implementasi pemrosesan data. Makalah ini membahas penerapan algoritma divide and conquer dalam mengatasi kompleksitas ruang pada salah satu operasi yang sering digunakan dalam model data relasional dan membutuhkan kapasitas memori yang relatif besar, yakni natural join.**

Kata kunci—divide and conquer; model relasional; natural join;

I. PENDAHULUAN

Pada model data relasional, terdapat banyak metode untuk menggabungkan relasi, salah satunya adalah natural join. Operasi natural join merupakan operasi yang sering digunakan untuk menggabungkan dua buah relasi berdasarkan atribut-atribut yang sama di kedua relasi. Karena natural join mengoperasikan dua buah relasi secara keseluruhan dan menghasilkan relasi baru dengan ukuran yang bisa saja sangat besar, operasi natural join merupakan salah satu operasi mahal yang membutuhkan optimasi.

Pendekatan naif dalam mengimplementasikan operasi natural join membutuhkan waktu dan memori yang relatif banyak pada basis data relasional. Kebutuhan waktu yang besar umumnya dapat diatasi dengan *schema tuning* pada implementasi di kehidupan nyata, namun kebutuhan memori besar tentunya sulit diatasi dengan menambahkan kapasitas perangkat keras yang membutuhkan biaya besar.

Makalah ini akan membahas penerapan algoritma *divide and conquer* dalam mengkomputasikan operasi *natural join* yang dapat membantu sistem manajemen basis data meskipun dalam keadaan kapasitas memori yang tidak mencukupi. Makalah ini juga akan menganalisis perbandingan serta *trade-off* dari algoritma *divide and conquer* terhadap algoritma naif.

II. LANDASAN TEORI

A. Model Data Relasional

Model data relasional merupakan model data yang dikembangkan berdasarkan konsep relasi matematika. Secara konsep, sebuah relasi (atau biasa disebut tabel) pada model relasional merupakan himpunan tupel [1]. Anggota dari sebuah relasi merupakan baris dari tabel, dan anggota tupel dari relasi tersebut disebut sebagai atribut atau kolom dari tabel [2]. Relasi dalam model relasional harus memenuhi syarat tambahan [1]:

- Setiap relasi harus memiliki nama unik
- Setiap atribut harus bertipe atomik
- Setiap atribut harus memiliki nama unik

B. Natural Join

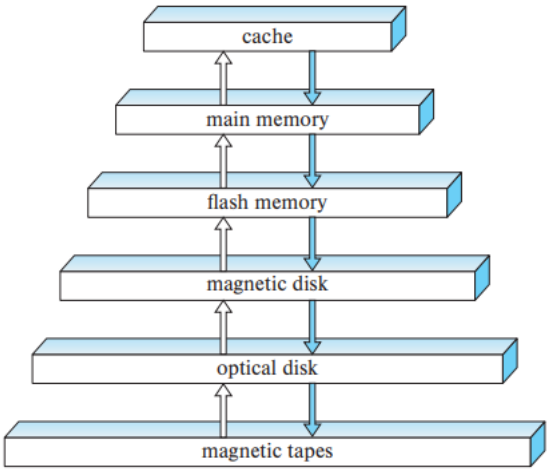
Dalam model data relasional, terdapat bahasa Query Formal yang menjadi teknik dasar dalam memproses data menjadi informasi yang dibutuhkan. Salah satu bahasa query "murni" adalah aljabar relasional, yang memiliki beberapa operasi dasar [3:47-53]:

- Operasi *select*, operasi yang menghasilkan relasi dengan tupel yang memenuhi predikat tertentu. Operasi *select* pada sebuah relasi r biasa dinotasikan sebagai $\sigma_p(r)$, dengan p adalah predikat boolean.
- Operasi *project*, menghasilkan relasi dengan atribut-atribut yang dideklarasikan berdasarkan sebuah relasi. Operasi *project* pada sebuah relasi r biasa dinotasikan sebagai $\Pi_{A1,A2,...,Ak}(r)$, dengan *A1*, *A2*, ..., *Ak* adalah nama-nama atribut.
- Operasi *cartesian-product*, operasi yang menghasilkan relasi hasil gabungan dua buah relasi. Relasi yang dihasilkan memiliki tupel hasil konkatenasi tupel-tupel relasi yang di-*cartesian-product*. Operasi *cartesian-product* pada dua buah relasi r_1 *dan* r_2 biasa dinotasikan sebagai $r_1 \times r_2$.

- Operasi *join*, operasi yang menghasilkan relasi hasil gabungan dua buah relasi berdasarkan atribut-atribut dengan nama yang sama pada kedua relasi. Relasi yang dihasilkan memiliki tupel hasil konkatenasi tupel-tupel relasi yang di-*join*, dengan atribut dengna nama yang sama disatukan, dan baris-baris yang dihasilkan merupakan tupel-tupel yang memiliki nilai yang sama pada atribut yang sama. Operasi *join* pada dua buah relasi r_1 *dan* r_2 biasa dinotasikan sebagai $r_1 \bowtie r_2$.

C. *Sistem Penyimpanan Fisik*

Sistem penyimpanan pada sebuah mesin dapat dibagi ke dalam bentuk hierarki, yang diurutkan berdasarkan kecepatan dan biayanya secara menurun ke bawah. Hierarki dimodelkan seperti gambar 1 menampilkan model hierarki sistem penyimpanan fisik pada perangkat keras.



Gambar 1. Model hierarki sistem penyimpanan fisik. *(H. F. Korth, S. Sudarshan, and A. S. Professor, Database System Concepts. McGraw-Hill Education, 2019, pp 562.)*

Media penyimpanan dengan kecepatan tertinggi (cache dan memori) dinamakan penyimpanan primer, dan media penyimpanan di bawahnya dinamakan penyimpanan sekunder. Perbedaan kecepatan pada pemindahan data dari penyimpanan sekunder sangat besar dibandingkan pemindahan data dari penyimpanan primer [3:561-562].

D. *Divide and Conquer*

Divide and conquer adalah metode penyelesaian persoalan yang memanfaatkan pembagian suatu persoalan menjadi upa-persoalan yang lebih kecil dan relatif lebih mudah diselesaikan. Pendekatan *divide and conquer* secara umum dibagi menjadi tiga bagian; *divide*, *conquer*, dan (jika diperlukan) *combine*.

Tahap *divide* merupakan tahap pemecahan sebuah persoalan menjadi beberapa upa-persoalan dengan karakteristik yang sama, dan umumnya dibagi menjadi bagian-bagian dengan ukuran serupa. Setelah tahap *divide*, tahap *conquer* menyelesaikan upa-persoalan (umumnya dengan pendekatan rekursif) dengan metode yang sama, dan terkadang terdapat algoritma khusus untuk menyelesaikan upa-persoalan dengan ukuran yang cukup kecil. Tahap terakhir dalam *divide and conquer – combine*, terkadang dibutuhkan ketika solusi masing-masing upa-persoalan perlu digabungkan untuk menghasilkan solusi dari persoalan semula [4].

III. PEMBAHASAN

Operasi *natural join* (operasi *join* pada aljabar relasional) adalah operasi yang dapat dilakukan dengan gabungan operasi dasar *select, project, dan cartesian-product*, yaitu dengan prosedur:

1. *cartesian-product* pasangan relasi *r1* dan *r2* yang akan di-*join*
2. *select* relasi yang dihasilkan operasi *cartesian-product* dengan predikat $r1.A1 = r2.A1 \wedge r1.A2 = r2.A2 \wedge \ldots \wedge r1.Ak = r2.Ak$
3. *project* relasi hasil *select* berdasarkan atribut-atribut *A1, A2, ..., Ak* yang merupakan atribut-atribut pada kedua relasi, tanpa mengulang nama atribut yang sama

Implementasi *natural join* menggunakan prosedur naif tersebut akan menghasilkan algoritma yang dapat dituliskan dalam *pseudocode*:

```
naiveNaturalJoin(R1, R2) → R3
        // menerapkan operasi natural join pada r1 dan r2
        // Input: list of tuples R1 & R2
        // Output hasil natural join R1 & R2

        // tentukan atribut yang sama pada kedua relasi
        A3 ← []
        foreach a1 in R1.attributes:
                foreach a2 in R2.attributes:
                        if a1 = a2 then:
                        A3.add(a1)
        R3.attributes ← R1.attributes
                        + (R2.attributes - A3)
        R3 ← []
        // lakukan operasi cartesian product
        foreach row1 in R1:
                foreach row2 in R2:

                        // lakukan operasi select
                        if row1[A3] = row2[A3]
                        // tuple baru merupakan hasil
                        // konkatenasi kedua buah row
                                row3 ← row1 + row2
                                R3.add(row3)

        return R3
```

Untuk menghitung kompleksitas waktu dan memori algoritma *natural join* tersebut, didefinisikan beberapa variabel untuk mempermudah analisis. Variabel yang didefinisikan adalah sebagai berikut:

- *a* adalah jumlah atribut *R1*

- *b* adalah jumlah atribut *R2*
- *n* adalah jumlah baris *R1*
- *m* adalah jumlah baris *R2*

Kompleksitas waktu algoritma ini dapat dihitung melalui *nested loop* setiap baris pada kedua tabel. Karena jumlah atribut (kolom) pada sebuah relasi pada umumnya jauh lebih sedikit dibandingkan banyak baris, maka perhitungan *nested loop* dalam mencocokkan atribut dapat diabaikan, waktu yang dibutuhkan untuk melakukan operasi *natural join* adalah

$$T((a+b)*n*m) = O(n*m) \quad (1)$$

Kompleksitas memori algoritma ini dapat dihitung melalui jumlah data *R1*, *R2*, dan *R3*. Perhitungan memori untuk variabel lokal dan atribut dapat diabaikan karena pada umumnya, jumlah data pada suatu tabel jauh lebih besar. Jumlah memori yang dibutuhkan pada kasus terburuk (operasi *natural join* menghasilkan tabel yang sama dengan operasi *cartesian-product*) adalah

$$T(a*n + b*m + (a+b)*n*m) = O(n*m) \quad (2)$$

Menjalankan operasi *natural join* secara langsung membutuhkan waktu pembacaan dari penyimpanan sekunder perangkat keras. Dalam pemindahan data dari penyimpanan sekunder, terdapat beberapa variabel tambahan yang dapat memengaruhi performa, yaitu:

- *r*, *overhead* pembacaan dari penyimpanan sekunder
- *w*, *overhead* penulisan ke penyimpanan sekunder

Pembacaan dari penyimpanan sekunder membutuhkan waktu sebanyak T(a*n + b*m + r), dan waktu penulisan ke penyimpanan sekunder sebanyak T((a+b)*n*m + w), sehingga kompleksitas waktu pemindahan data dari penyimpanan sekunder adalah

$$T(a*n + b*m + r + (a+b)*n*m + w) = O(n*m) \quad (3)$$

yang lebih besar dibandingkan operasi *natural join*, dan mempertimbangkan kemampuan teknologi penyimpanan sekunder saat ini, waktu yang dibutuhkan untuk proses pemindahan data jauh lebih besar dibandingkan waktu yang dibutuhkan untuk operasi *natural join*.

Selain itu, kebutuhan kapasitas memori dalam menjalankan operasi *natural join* secara naif akan membutuhkan ruang yang sama dengan menjalankan operasi tersebut, seperti yang ditunjukkan pada (2). Keterbatasan teknologi saat ini menyebabkan perangkat keras tidak dapat memproses data ketika operasi *natural join* diterapkan pada data yang sangat banyak dan kebutuhan memori melebihi kapasitas yang dapat digunakan.

Untuk mengatasi keterbatasan memori, dapat diterapkan algoritma *divide and conquer* untuk membagi data-data yang akan diproses menjadi beberapa bagian kecil untuk diproses pada memori. Prosedur *natural join* menggunakan algoritma *divide and conquer* adalah:

1. Bagi data menjadi kumpulan data yang lebih kecil (dapat menggunakan ukuran konstan, *p* baris R1 dan *q* baris R2) untuk dioperasikan *natural join*, lakukan pembacaan dari penyimpanan sekunder sesuai dengan ukuran data setelah dibagi
2. Jalankan operasi *natural join* semula untuk kumpulan data tersebut
3. Simpan hasil operasi *natural join* pada penyimpanan sekunder, lakukan hingga semua data telah dioperasikan *natural join*

Implementasi *natural join* menggunakan algoritma *divide and conquer* dengan memanfaatkan prosedur naif dalam *pseudocode* adalah:

```
dncNaturalJoin(RF1, p, RF2, q, RF3)
        // menerapkan operasi natural join
        //          pada R1 dan R2, dan menyimpan hasil
        //          pada disk
        // Input: alamat tabel 1, tabel 2, dan tabel 3
        //          pada penyimpanan sekunder
        //          ukuran "chunk" tabel 1 & 2
        //
        // Final State: hasil natural join R1 & R2
        //          disimpan pada penyimpanan sekunder

        k ← 0

        for i in [0..n div p]:
        // baca "chunk" tabel 1 dari disk sebesar p
                R1←read(RF1, i*p, (i+1)*p-1)
                for j in [0..m div q]:
        // baca "chunk" tabel 2 dari disk sebesar q
                        R2←read(RF2, j*q, (j+1)*q-1)

        // catat hasil natural join kedua chunk
                        R3←naiveNaturalJoin(R1, R2)
        // tuliskan ke disk "chunk" hasil natural join
        // agar memori dapat digunakan kembali
                        write(RF3, k, k+R3.size)
                        k ← k + R3.size
```

Waktu yang dibutuhkan oleh algoritma *divide and conquer* pada operasi *natural join* dengan memperhitungkan pemindahan data dari penyimpanan sekunder dapat dihitung melalui *nested loop*, yaitu T((N/p)*(p + r + (M/q)*(q + r + (p * q) + w))), dengan $N = a*n$, dan $M = b*m$. Sehingga kompleksitas waktu yang dibutuhkan adalah

$$T(N+M*(N/p)+NM+r*(N/p+NM/qp)+w*(NM/qp))=O(N*M) \quad (4)$$

Selain itu, memori yang dibutuhkan oleh algoritma *divide and conquer* dapat dihitung pada setiap iterasi loop terdalam,

karena memori yang digunakan demi menyimpan data sementara untuk diproses hanya memanfaatkan R1, R2 dan R3, yang jumlahnya memiliki ukuran T(p + q + p*q), sehingga kompleksitas ruang algoritma *divide and conquer* adalah

$$T(p + q + p*q) = O(p*q) \quad (5)$$

Berdasarkan hasil implementasi *divide and conquer* dalam operasi *natural join*, dapat diamati bahwa waktu yang dibutuhkan oleh algoritma meningkat jika ukuran data yang diproses mengecil, namun ukuran memori yang digunakan semakin sedikit jika ukuran data yang diproses mengecil. Meskipun demikian, kompleksitas waktu pemindahan data berada pada orde yang sama dengan algoritma naif (tanpa menggunakan *divide and conquer*), sementara kompleksitas ruang dapat mengalami penurunan berdasarkan kebutuhan sistem.

IV. KESIMPULAN

Algoritma *divide and conquer* tidak hanya dapat meningkatkan efisiensi terhadap penggunaan waktu, namun juga dapat meningkatkan efisiensi terhadap penggunaan memori. Peningkatan efisiensi memori merupakan salah satu kebutuhan penting, terutama dalam sistem manajemen basis data, yang menyimpan dan mengolah data dalam volume yang relatif besar. Salah satu potensi optimasi dalam meningkatkan efisiensi memori pada sistem basis data relasional adalah dengan menerapkan algoritma *divide and conquer* dalam operasi *natural join*. Penerapan algoritma *divide and conquer* untuk meningkatkan efisiensi memori memungkinkan sistem untuk menjalankan operasi tersebut tanpa harus mengeluarkan biaya tambahan dalam menambah kemampuan perangkat keras yang biayanya besar. Namun, penghematan memori berlebihan untuk solusi tersebut menghasilkan *trade-off* pada waktu yang dibutuhkan untuk menjalankan operasi *natural join*, sehingga dapat dilakukan *tuning* untuk mencapai titik optimal antara efisiensi waktu dan memori.

UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa atas kesempatan dan hikmat yang diberikan-Nya dalam proses pengerjaan makalah ini. Penulis mengucapkan terima kasih kepada dosen pengampu mata kuliah IF2211 Strategi Algoritma Semester Genap 2022/2023 kelas 02, Dr. Nur Ulfa Maulidevi, S.T, M.Sc., dan dosen pengampu mata kuliah IF2240 Basis Data Semester Genap 2022/2023 kelas 02, Dr. Fazat Nur Azizah, S.T., M.Sc. yang telah menyalurkan ilmu-ilmu yang digunakan untuk menulis makalah ini. Tugas makalah ini telah menjadi wadah menuangkan buah pikiran penulis mengenai topik ini dan telah memberi wawasan yang lebih mendalam bagi penulis. Selain itu, tugas ini telah menambah pengalaman bagi penulis dalam pembuatan makalah ilmiah.

REFERENSI

[1] E. F. Codd, The Relational Model for Database Management: Version 2. Addison Wesley Publishing Company, 1990, pp. 1-5.

[2] J. A. Hoffer, V. Ramesh, and H. Topi, *Modern Database Management*. Pearson, 2016, pp. 188-189.

[3] H. F. Korth, S. Sudarshan, and A. S. Professor, *Database System Concepts*. McGraw-Hill Education, 2019.

[4] A. Levitin, Introduction to the Design and Analysis of Algorithms. Pearson Education, 2011, pp. 195-197.

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, 22 Mei 2023

Johann Christian Kandani 13521138