Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

PERBANDINGAN PENCARIAN DATA MENGGUNAKAN *QUERY* HASH JOIN DAN *QUERY* NESTED JOIN

Junus Sinuraya[1*], Muhammad Zarlis[2], Erna Budhiarti Nababan[2]
[1]Program Studi Teknik Komputer Politeknik LP3I Medan
[2]Program S2 Teknik Informatika Universitas Sumatera Utara, Medan, Indonesia
Telp: 061-7322634, Fax: 061-7322649
*E-Mail: Junus.Sinuraya2012@gmail.com

ABSTRAK

Pengaksesan data atau pencarian data dengan menggunakan *Query* atau *Join* pada aplikasi yang terhubung dengan sebuah database perlu memperhatikan ketepatgunaan implementasi dari data itu sendiri serta waktu prosesnya. Ada banyak cara yang dapat dilakukan oleh database manajemen sistem dalam memproses dan menghasilkan jawaban sebuah *query*. Semua cara pada akhirnya akan menghasilkan jawaban (*output*) yang sama tetapi pasti mempunyai harga yang berbeda-beda, seperti kecepatan waktu untuk merespon data. Beberapa *query* yang sering digunakan untuk pemrosesan data yaitu *Query Hash Join* dan *Query Nested Join*, kedua *query* memiliki algoritma yang berbeda tapi menghasilkan *output* yang sama. Dengan menggunakan aplikasi yang dirancang menggunakan Microsoft Visual Studi 2010 dan Microsoft SQL Server 2008 berbasis jaringan untuk melakukan pengujian kedua algoritma atau *query* dengan parameter *running time* atau kecepatan waktu merespon data. Pengujian dilakukan dengan jumlah tabel yang dihubungkan dan jumlah baris/record. Hasil dari penelitian adalah kecepatan waktu *query* dalam merespon data untuk jumlah data yang kecil *query hash join* lebih baik dibandingkan dengan jumlah data yang besar *query nested join*.

Kata Kunci : *Query*, *Hash Join*, *Nested Join*.

PENDAHULUAN

Database Manajemen Sistem (DBMS) merupakan perantara bagi pemakai dengan basis data. Untuk berinteraksi dengan DBMS (basis data) menggunakan bahasa basis data yang telah ditentukan oleh perusahaan DBMS. Bahasa basis data biasanya terdiri atas perintah-perintah yang diformulasikan sehingga perintah tersebut akan diproses olah DBMS. Perintah-perintah biasanya ditentukan oleh *user*. Perintah *user* berinterakasi dengan Database Manajemen Sistem diantaranya yaitu memasukkan data, mengubah data, menghapus dan menampilkan data atau disebut dengan *Data Manipulation Language* (DML). *Structure Query Language (SQL)* adalah standar bahasa untuk berinteraksi dengan database dengan menggunakan operasi-operasi umum pada basis data. *Query* adalah semacam kemampuan untuk menampilkan suatu data dari database dimana mengambil dari tabel-tabel yang didatabase, namun tabel tersebut tidak semua ditampilkan sesuai yang diinginkan atau data yang ingin ditampilkan.

Join table adalah penggabungan tabel-tabel menggunakan *Query* yang dilakukan melalui kolom/*key* tertentu yang memiliki nilai terkait untuk mendapatkan satu set data dengan informasi lengkap. Lengkap artinya kolom data didapatkan dari kolom-kolom hasil *join* antar tabel tersebut. *Join* diperlukan karena perancangan tabel pada sistem transaksional kebanyakan dinormalisasi, salah satu alasannya adalah untuk mengurangi redundansi.

Pengaksesan data atau pencarian data dengan menggunakan *Query* atau *Join* pada database perlu memperhatikan ketepatgunaan implementasi dari data itu sendiri serta waktu prosesnya. Ada

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

banyak cara yang dapat dilakukan oleh database manajemen sistem dalam memproses dan menghasilkan jawaban sebuah *query*. Semua cara pada akhirnya akan menghasilkan jawaban (*output*) yang sama tetapi mempunyai harga yang berbeda-beda, misalnya total waktu yang diperlukan untuk menjalankan sebuah *query*. Pencarian data atau pengolahan data tersebut dapat dilakukan dengan mengakses data yang terdapat dalam *database*. Pengaksesan *data* tersebut dilakukan dengan melakukan *query-query* pada basisdata-basisdata dengan *database* manajemen sistem. Kecepatan akses data dapat ditingkatkan dengan banyak cara, selain dari sisi perangkat kerasnya (*hardware*) dapat juga dilakukan dari sisi perangkat lunaknya (*software*) lebih khusus pada program aplikasinya.

Algoritma *query* melakukan join ada beberapa algoritma yang sering digunakan yaitu Algoritma *Hash Join* dan Algoritma *Nested Join*. Masing-masing algoritma memilik kegunaan yang sama dalam menggabungkan beberapa tabel atau relasi tabel pada database manajemen sistem. Algoritma *Hash Join* adalah sebuah algoritma join untuk menggabungkan data yang berjumlah besar. Cara kerja *Hash Join* adalah *Optimizer* membuat sebuah *Hash Table* berdasarkan predikat *Join*. Setiap tabel di *Inner* maupun *Outer* masing-masing dijadikan sebuah kode dengan *Hash Function* kemudian setiap kode *Hash* dari *Inner* akan dibandingkan dengan *Hash* Kode dari *Outer*. Apabila kode *Hash* dari *Inner* dan *Outer* sama maka akan dilakukan proses pengecekan nilai dari kolom yang pada akhirnya akan dimasukkan ke dalam hasil jika nilai kolomnya sama. Algoritma *Nested Join* adalah sebuah *Join* yang efektif. Jika *subset* yang digabungkan berjumlah sedikit dan jika kondisi dalam perintah *join* efisien untuk menggabungkan 2(dua) tabel tersebut.

METODE PENELITIAN

Sumber data untuk menjadi studi kasus dalam penelitian ini adalah master basis data anggaran mempunyai 6 tabel yang saling berhubungan. Semua tabel tidak memiliki indeks atau *cluster* yang digunakan dalam pencarian data.

1. Skenario Pengujian *Query*

Pengujian pencarian data menggunakan *query* dua algoritma yaitu *Query Hash Join dan Query Nested Join*. Pengujian *Query* dalam hal pencarian data berdasarkan banyak jumlah data untuk melakukan pencarian data dibagi atas beberapa tahap relasi dan group data. Masing-masing group mempunyai jumlah data yang berbeda. Berikut group data sebagai uji coba dan berdasarkan jumlah data yang berbeda untuk penelitian berikut tahap relasi (tabel 1) dan tabel group data (tabel 2). Jumlah record/baris data masing-masing tabel menggunakan data *dummy* dan penambahan data menggunakan perintah *query* untuk melakukan pengujian *query* algoritma. Gambar 2 adalah flowchart untuk menguji *query* pencarian data berdasarkan kriteria diatas.

2. *Query* Hash Join dan Nested Join

Query-query kedua algoritma yang akan diuji coba dalam penelitian ini dilakukan beberapa tahap relasi dan menghasilkan informasi yang sama. Untuk menghitung waktu yang diperlukan dalam menampilkan data yang dicari maka dibuat sebuah aplikasi yang menggunakan *Visual Basic. Net 2010* dan *Microsoft SQL Server 2008* sebagai Database Manajemen Sistem.

3. Paramter Pengujian *Query*

Parameter yang digunakan untuk menguji *query-query* diatas dalam pencarian data adalah kecepatan waktu mengeksekusi dan menampilkan informasi dalam pencarian data. Untuk menghitung waktu kecepatan yang diperlukan dalam mengakses data maka dibuat sebuah aplikasi. Satuan waktu untuk menghitung *query-query* diatas adalah satuan detik (Gambar 3-Gambar 7).

4. Alat Penelitian

Pada penelitian ini digunakan alat penelitian berupa perangkat keras dan perangkat lunak sebagai berikut:

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

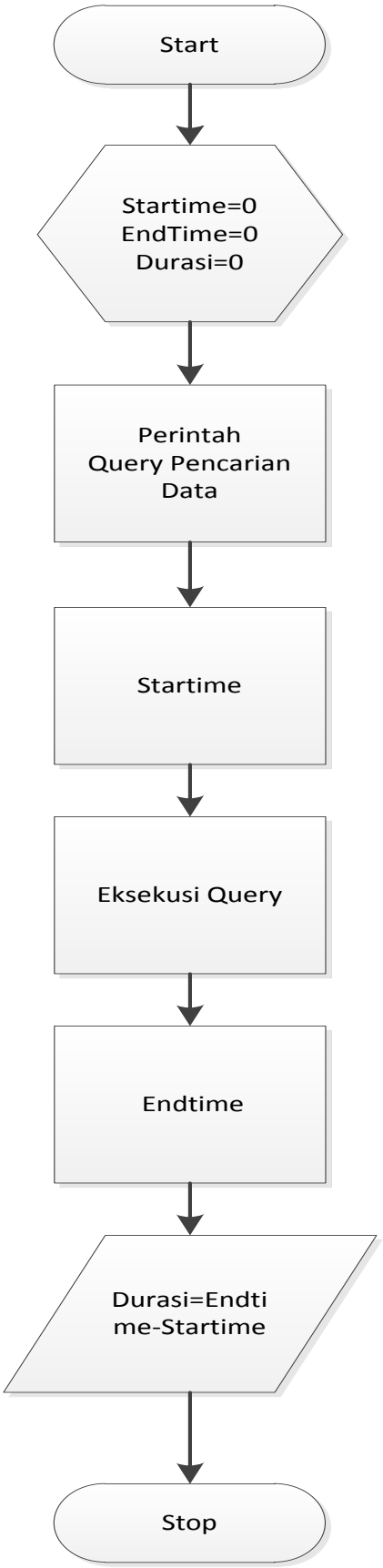
a. Perangkat keras
   1) *Processor* Intel Pentium Core Duo.
   2) RAM 2GB.
   3) *Harddisk* 500 GB.
   4) Monitor dengan resolusi 1024 x 768 *pixel* 32 *bit color.*
   5) *Mouse dan keyboard.*
b. Perangkat lunak
   1) Sistem Operasi Windows XP *service pack* 3.
   2) Visual Basic.Net 2010.
   3) Microsoft SQL Server 2008.

Tabel 1. Tahapan Relasi

No	Jumlah Relasi
1	1 Relasi
2	2 Relasi
4	3 Relasi
5	4 Relasi
6	5 Relasi

Tabel 2. Tabel Group data Uji Penelitian

NO	Jumlah Data(Record)	Kelompok Data
1	10	Kecil
2	20	
3	40	
4	80	
5	160	
6	320	
7	640	Sedang
8	1280	
9	2560	
10	5120	
11	10240	
12	20480	
13	40960	Besar
14	81920	
15	163380	
16	655360	
17	1.310.720	



Gambar 1. Flowchart Skenario Pengujian *Query*

HASIL DAN PEMBAHASAN

Percobaan pada jaringan *peer to peer* dengan spesifikasi hardware sebagai berikut:

Server
- Processor Intel Core I3 2.20 GHz
- Hardisk 360 GB
- Memory 2 GB
- Sistem Operasi Windows 7

Client
- Processor Intel Core Duo 2.10 GHz
- Harddisk 360 GB
- Memory 2 GB
- Sistem Operasi Windows XP
- Microsfot Visual Basic.Net 2010

Agar memudahkan melihat perbandingan *query hash join*, *nested join* maka mengelompokan data berdasarkan banyak jumlah data sebagai berikut:
- Kelompok data kecil berkisar antara 10-320 baris(*Record)*
- Kelompok data sedang berkisar antara 640-20480 baris(*Record)*
- Kelompok data besar berkisar antara 40960-1310720 baris(*Record)*

1. Hasil Pengujian Query 1Relasi

Hasil pengujian*query*dengan menghubungkan 2 (dua) tabel dalam bentuk *query* dan jumlah data yang berbeda-beda berdasarkan kelompok data maka didapat kecepatan waktu *query* dalam melakukan pencarian data. Adapun hasil pengujian *query*sebagai berikut:

Hasil Uji Query

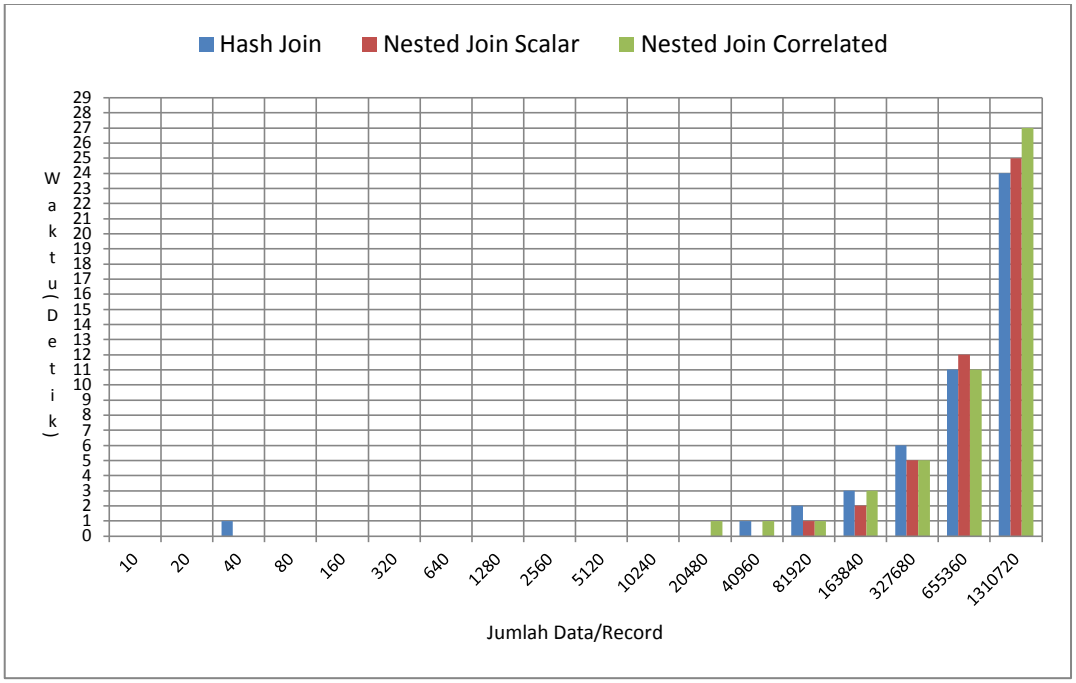
NO	Jumlah Data	Hash Join(detik)	Nested Join Scalar(Detik)	Nested Join Correlated(Detik)
1	10	0	0	0
2	20	0	0	0
3	40	1	0	0
4	80	0	0	0
5	160	0	0	0
6	320	0	0	0
7	640	0	0	0
8	1280	0	0	0
9	2560	0	0	0
10	5120	0	0	0
11	10240	0	0	0
12	20480	0	0	1
13	40960	1	0	1
14	81920	2	1	1
15	163840	3	2	3
16	327680	6	5	5
17	655360	11	12	11
18	1310720	24	25	27

Gambar 2. Perbandingan *Running Time Query*1 Relasi

Dari data diatas diatas dapat dilihat bahwa kelompok data kecil atau record yang sedikit kecepatan waktu(detik) query dalam melakukan pencarian data ketiga query diatas memilik kecepatan waktu yang hampir sama walapun *query hash join* kadang agak lambat tetapi tidak

terlalu besarsedangkankelompok data menengah memilik waktu yang sama dan kelompok data besarwaktu untuk mengakses data memiliki waktu yang berbeda dimana *Query Nested JoinScalar* menunjukan lebih baik dibanding query dua lainnya namun perbedaan waktu tidak begitu besar.

Agar memudahkan melihat perbedaan ketiga query dalam mengakses data dari aplikasi berikut grafik ketiga query:



Gambar 3. Grafik Perbandingan*Running Time*Query1 Relasi

Dari grafik diatas dapat dianalisa sebagai berikut:

1. *Query Hash Join* mengakses data 1 tabel secara penuh dari aplikasi dengan jumlah data yang kecil waktu dibutuhkan dalam mengakses tidak stabil sedangkan jumlah data yang besar waktu yang dibutuhkan lama semakin besar jumlah data diuji maka waktu yang dibutuhkan juga semakin lama untuk mengakses datanya.
2. *Query Nested Join Scalar* mengakses data 1 tabel secara penuh dari aplikasi jumlah data yang kecil waktu yang dibutuhkan hampir sama dengan ketiga *query* dan jumlah data yang besar waktu yang dibutuhkan lebih baik dibanding *query* lainnya.
3. *Query Nested Join Correlated* mengakses data 1 tabel secara penuh dari aplikasi jumlah data yang kecil waktu yang dibutuhkan untuk mengakses data sama dengan *Query Nested Join Scalar* tapi jumlah data yang besar lebih cepat dibanding *Query Hash Join* dan lebih lama dibanding *Query Nested Join Scalar*.

2. Hasil Pengujian 2 Relasi

Pengujian query dengan menghubungkan 3(tiga) tabel menampilkan secara penuh 1 tabel untuk melakukan pencarian data atau mengakses data dari aplikasi dengan menggunakan 3 query yaitu *Query Hash Join*, *Query Nested Join Scalar* dan *Query Nested Join Correlated*. Berikut hasil ketiga query untuk mengakses data dari aplikasi sebagai berikut:

Hasil Uji Query

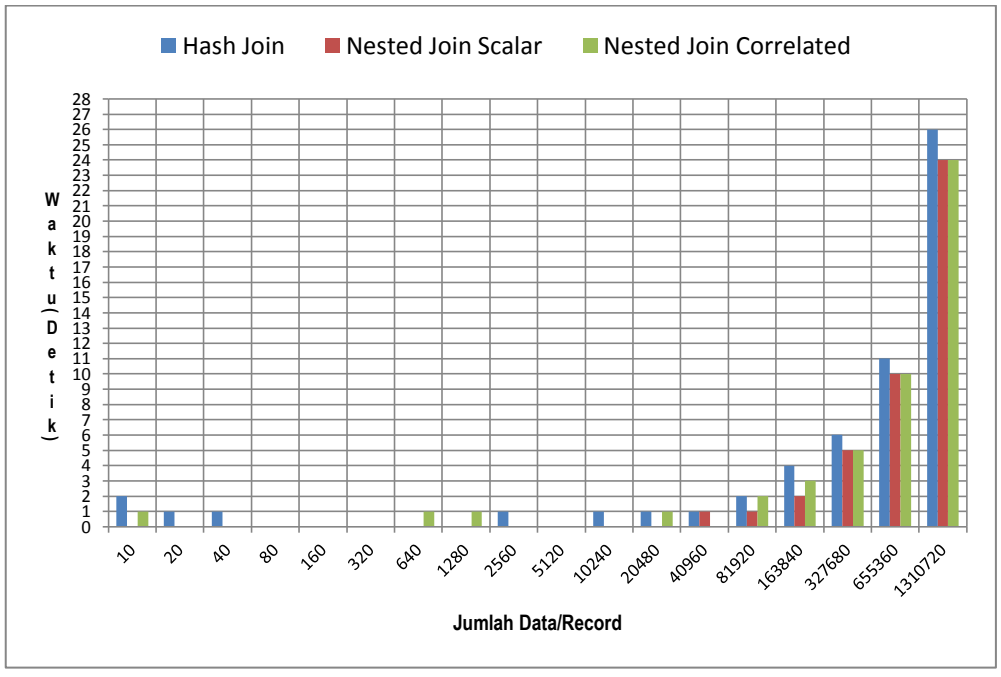
NO	Jumlah Data	Hash Join(detik)	Nested Join Scalar(Detik)	Nested Join Correlated(Detik)
1	10	2	0	1
2	20	1	0	0
3	40	1	0	0
4	80	0	0	0
5	160	0	0	0
6	320	0	0	0
7	640	0	0	1
8	1280	0	0	1
9	2560	1	0	0
10	5120	0	0	0
11	10240	1	0	0
12	20480	1	0	1
13	40960	1	1	0
14	81920	2	1	2
15	163840	4	2	3
16	327680	6	5	5
17	655360	11	10	10
18	1310720	26	24	24

Gambar 4. Perbandingan *Running TimeQuery*2 Relasi

Dari hasil pengujian *query* berdasarkan data diatas bisa dijelaskan sebagai berikut:

1. *Query* kelompok data kecil *Query Hash Join* lebih lama dibanding*query Nested Join* dan *query Nested Join scalar* lebih baik dibanding *Nested Join Correlated*.
2. *Query* kelompok data sedang *Query Hash Join* lebih lama dibandingkan *query nested join*, *query nested join scalar* lebih stabil dibanding yang lainnya.
3. *Query* kelompok data besar*query nested join* lebih baik dibanding *query hash join* sedangkan *query nested join scalar* lebih stabil dibanding *query nested join correlated.*

Agar lebih mudah untuk melihat perbandingan kinerja ketiga query dalam mengakses data dapat dibuat dalam bentuk grafik sebagai berikut:



Gambar 5. Grafik Perbandingan*Running TimeQuery*2 Relasi

Dari grafik diatas dapat dianalisa sebagai berikut:

1. *Query Hash Join* mengakses data 1 tabel secara penuh dari aplikasi dengan jumlah data yang kecil waktu dibutuhkan dalam mengakses tidak stabil sedangkan jumlah data yang besar waktu yang dibutuhkan lama semakin besar jumlah data diuji maka waktu yang dibutuhkan juga semakin lama untuk mengakses datanya.
2. *Query Nested Join Scalar* mengakses data 1 tabel secara penuh dari aplikasi jumlah data yang kecil waktu yang dibutuhkan hampir sama dengan ketiga *query* dan jumlah data yang besar waktu yang dibutuhkan lebih baik dibanding *query* lainnya.
3. *Query Nested Join Correlated* mengakses data 1 tabel secara penuh dari aplikasi jumlah data yang kecil waktu yang dibutuhkan untuk mengakses data sama dengan *Query Nested Join Scalar* tapi jumlah data yang besar lebih cepat dibanding *Query Hash Join* dan lebih lama dibanding *Query Nested Join Scalar*.

3. Hasil Pengujian Query 3 Relasi

Pengujian query yang dilakukan yang menghubungkan 4 tabel sekaligus dalam mengakses data satu tabel secara penuh. Berikut perbandingan waktu yang dibutuhkan query dalam mengakses data.

Hasil Uji Query

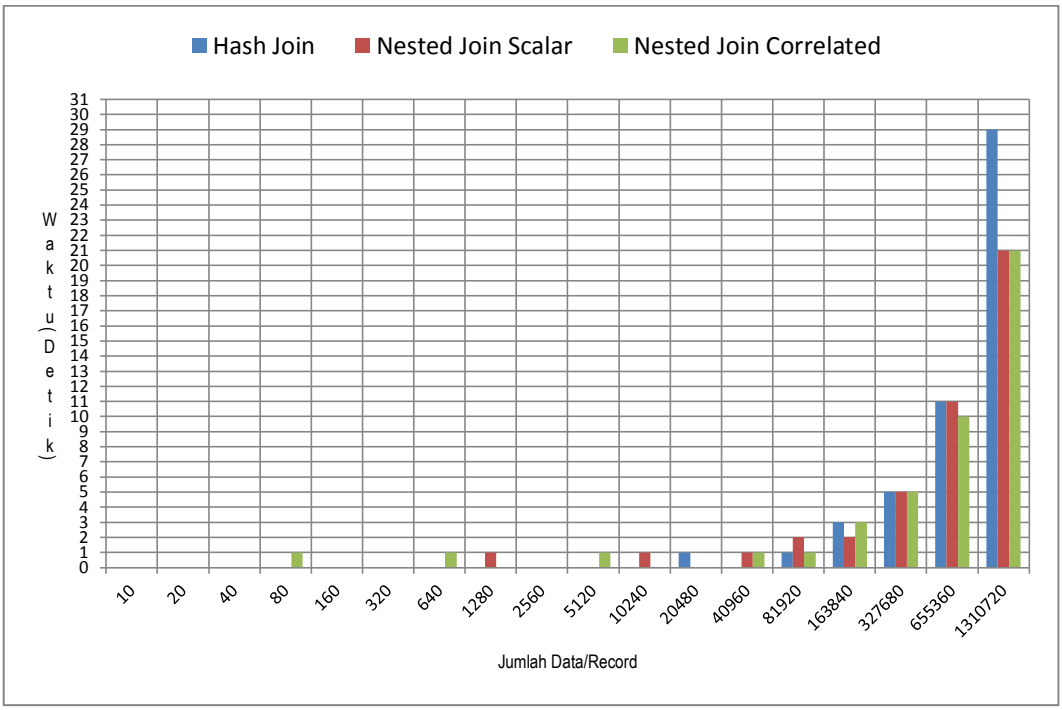
NO	Jumlah Data	Hash Join(detik)	Nested Join Scalar(Detik)	Nested Join Correlated(Detik)
1	10	0	0	0
2	20	0	0	0
3	40	0	0	0
4	80	0	0	1
5	160	0	0	0
6	320	0	0	0
7	640	0	0	1
8	1280	0	1	0
9	2560	0	0	0
10	5120	0	0	1
11	10240	0	1	0
12	20480	1	0	0
13	40960	0	1	1
14	81920	1	2	1
15	163840	3	2	3
16	327680	5	5	5
17	655360	11	11	10
18	1310720	29	21	21

Gambar 6. Perbandingan Running Time*Query* 3 Relasi

Dari hasil pengujian *query* berdasarkan data diatas bisa dijelaskan sebagai berikut:

1. Query Kelompok data kecil query hash join dan nested join scalar lebih stabil dalam kecepatan mengakses data sedangkan *query nested join correlated* tidak berbeda jauh dengan 2 *query* lainnya tapi *nested join correlated* tidak stabil.
2. Query Kelompok data sedang *query hash join* lebih baik dibanding *nested join baik scalar* maupun *correlated*.
3. Query Kelompok data besar memiliki waktu yang berbeda jauh dengan *query hash join* dengan *query nested join* dimana waktu selisih jauh lebih cepat *query nested join* dibanding *query hash join.*

Agar lebih mudah untuk melihat perbandingan kinerja ketiga query dalam mengakses data dapat dibuat dalam bentuk grafik sebagai berikut:



Gambar 7. Grafik Perbandingan *Query* Pencarian Data Relasi 3 Tabel

Dari grafik diatas dapat dianalisa sebagai berikut:

1. *Query Hash Join* mengakses data 1 tabel secara penuh dari aplikasi dengan jumlah data yang kecil waktu dibutuhkan dalam mengakses tidak stabil sedangkan jumlah data yang besar waktu yang dibutuhkan lama semakin besar jumlah data diuji maka waktu yang dibutuhkan juga semakin lama untuk mengakses datanya.
2. *Query Nested Join Scalar* mengakses data 1 tabel secara penuh dari aplikasi jumlah data yang kecil waktu yang dibutuhkan hampir sama dengan ketiga *query* dan jumlah data yang besar waktu yang dibutuhkan lebih baik dibanding *query* lainnya.
3. *Query Nested Join Correlated* mengakses data 1 tabel secara penuh dari aplikasi jumlah data yang kecil waktu yang dibutuhkan untuk mengakses data sama dengan *Query Nested Join Scalar* tapi jumlah data yang besar lebih cepat dibanding *Query Hash Join* dan lebih lama dibanding *Query Nested Join Scalar*.

4. Hasil Pengujian Query 4 Relasi

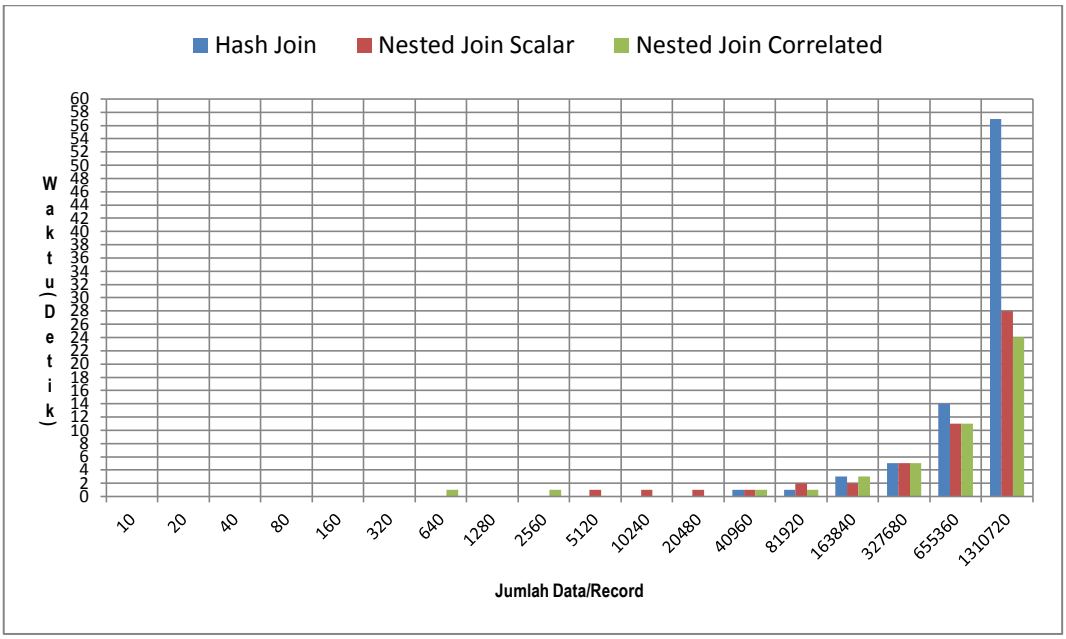
Pengujian query yang dilakukan yang menghubungkan 5 tabel sekaligus dalam mengakses data satu tabel secara penuh. Berikut perbandingan waktu yang dibutuhkan query dalam mengakses data.

Hasil Uji Query

NO	Jumlah Data	Hash Join(detik)	Nested Join Scalar(Detik)	Nested Join Correlated(Detik)
1	10	0	0	0
2	20	0	0	0
3	40	0	0	0
4	80	0	0	0
5	160	0	0	0
6	320	0	0	0
7	640	0	0	1
8	1280	0	0	0
9	2560	0	0	1
10	5120	0	1	0
11	10240	0	1	0
12	20480	0	1	0
13	40960	1	1	1
14	81920	1	2	1
15	163840	3	2	3
16	327680	5	5	5
17	655360	14	11	11
18	1310720	57	28	24

Gambar 8. Grafik Perbandingan *Query* Pencarian Data Relasi 3 Tabel

Hasil pengujian query diatas dapat dilihat bahwa untuk jumlah data yang kecil tidak ada perbedaan waktu yang dibutuhkan untuk mengakses data. Jumlah data yang besar terjadi perubahan yang signifikan dmana query hash join lebih besar waktu dibutuhkan untuk mengakses data dibandingkan Nested Join.



Gambar 9. Grafik Perbandingan *Query* Pencarian Data Relasi 5 Tabel

Dari grafik diatas dapat dilihat bahwa kelompok data kecil *query hash join*, *nestedjoin scalar* dan *correlated* memiliki waktu yang sama, kelompok data sedang *hash join* lebih baik dibanding *query* lainnya dan kelompok data *besar nested join correlated* lebih baik query lainnya.

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

5. Hasil Pengujian Query 5 Relasi

Hasil Pengujian *query Hash Join*, *Nested Join* yang dilakukan yang menghubungkan 6(enam) tabel sekaligus dalam mengakses data satu tabel secara penuh. Berikut perbandingan *running time query* yang dibutuhkan dalam mengakses data.

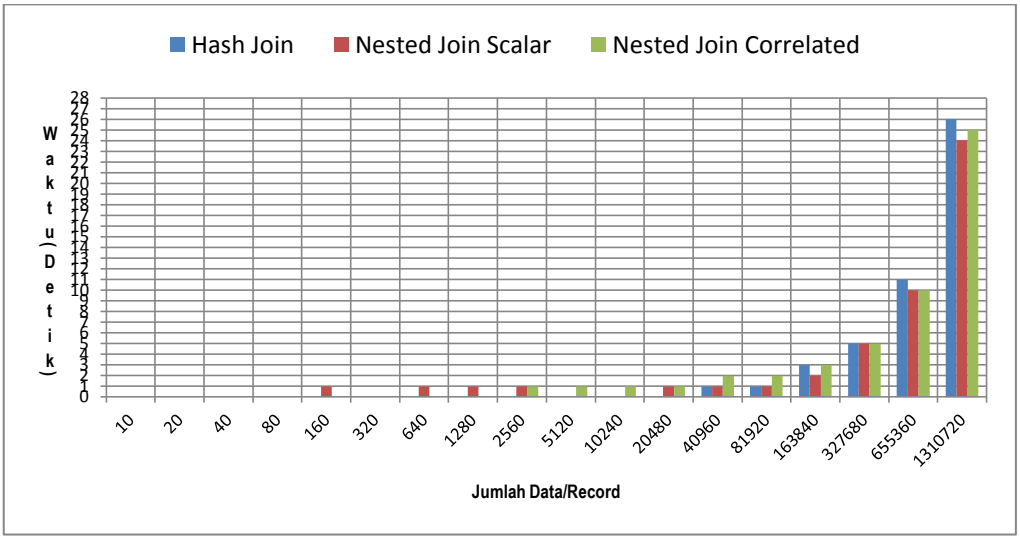
Hasil Uji Query

NO	Jumlah Data	Hash Join(detik)	Nested Join Scalar(Detik)	Nested Join Correlated(Detik)
1	10	0	0	0
2	20	0	0	0
3	40	0	0	0
4	80	0	0	0
5	160	0	1	0
6	320	0	0	0
7	640	0	1	0
8	1280	0	1	0
9	2560	0	1	1
10	5120	0	0	1
11	10240	0	0	1
12	20480	0	1	1
13	40960	1	1	2
14	81920	1	1	2
15	163840	3	2	3
16	327680	5	5	5
17	655360	11	10	10
18	1310720	26	24	25

Gambar 10. Hasil Perbandingan Pengujian *Running TimeQuery* 5 Relasi

Pada gambar diatas hasil pengujian *query Hash Join*untuk jumlah data yang kecil waktu yang dibutuhkan mengakses data kecil dan jumlah data yang besar waktu yang dibutuhkan semakin tinggi. Hasil Pengujian *Query Nested Join Scalar*untuk jumlah data yang kecil waktu yang dibutuhkan kecil akan tetapi pada saat jumlah data yang sedang atau belum begitu besar hasil querynya cendrung naik dan waktu jumlah data yang besar juga semakin tinggi. Hasil Pengujian *Query Nested Join Correlated* untuk jumlah data yang kecil waktu yang dibutuhkan hampir sama dengan *Query Hash Join* tapi sedikit lebih baik dibanding *Query Nested Join Scalar* untuk data yang sedang dan jumlah data yang besar waktu yang dibutuhkan juga semakin tinggi.

Hasil pengujian *query* 5 relasi dapat dilihat dalam grafik sebagai berikut:



Gambar 11. Grafik Perbandingan Hasil Pengujian *Running TimeQuery* 5 Relasi

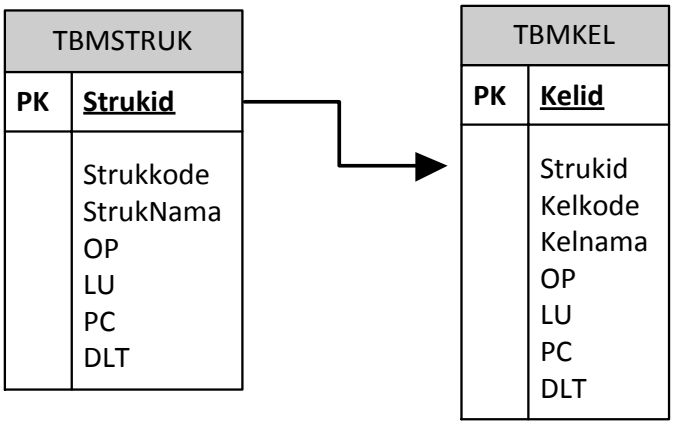
Dari grafik diatas dapat jelaskan bahwa kelompok data kecil *query hash join*,*Nested join correlated* lebih baik dibanding *Query Nested Scalar*akan tetapi kelompok data sedang *query* hash join lebih baik dibanding kedua *query* lainnya akan tetapi kelompok data yang lebih besar jauh lebih unggul *Nested Join Scalar* dibanding *query* lainnya.

6. Pembahasan Penelitian

Kecepatan query untuk mengakses data khususnya pencarian data banyak faktor yang mempengharui kecepatan query tersebutakan tetapi disini penulis meneliti query dari kecepatan waktu untuk menakses data dari aplikasi.

7. Analisis Hasil Pengujian Query 1 Relasi

Tabel yang akan diuji dan direlasikan dapat dilihat sebagai berikut:



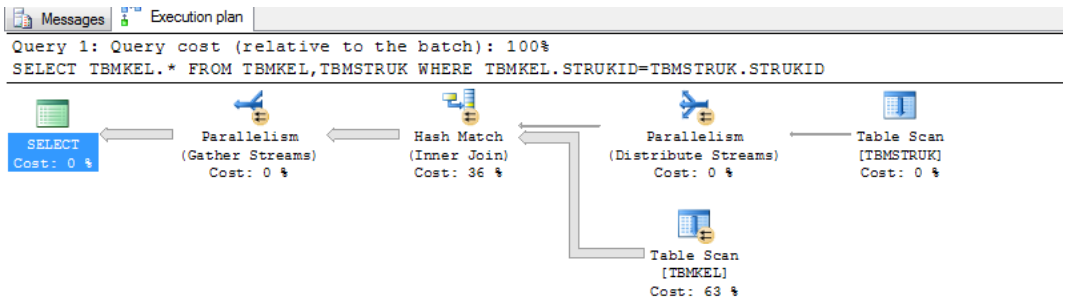
Gambar 12. Relasi antar 2 tabel

Pengujian pencarian data dengan menghubungkan dengan kedua tabel menggunakan *query hash join*, *nested join*. *Nested join* dibagi dua yaitu *scalar* dan *correlated*.

Query Hash Join yang dianalisa sebagai berikut:

```
SELECT TBMKEL.*WHERE TBMKEL.STRUKID=TBMSTRUK.STRUKID
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



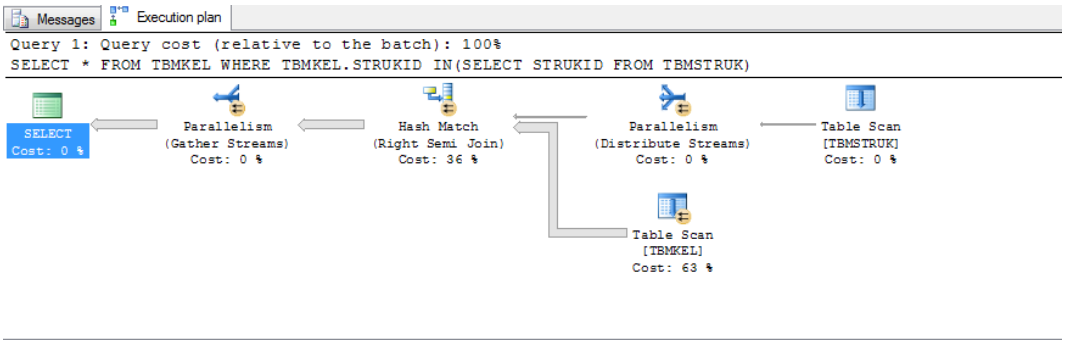
Gambar 13. *Display Estimated Execution Plan Query Hash Join* 1 Relasi

Query diatas dieksekusi dengan *inner join* mengembalikan semua baris dari kedua tabel dimana terdapat kecocokan atau merupakan irisan kedua tabel. Jika ada ada yang tidak cocok dalam tabel –tabel tersebut, maka baris tersebut tidak ditampilkan dan mencari kecocokan dengan cara table scan yaitu mencari kecocokan baris perbaris diantara satu tabel dengan tabel yang terhubung, semua data yang cocok akan disimpan dimemory sehingga untuk data yang lebih sedikit akan cepat, jika data yang besar maka memory akan penuh sehingga proses akan lebih lama.

Query Nested Join Scalar yang diuji dan analisa sebagai berikut:

```
SELECT* FROM TBMKEL WHERE TBMKEL.STRUKID IN(SELECT STRUKID
FROM TBMSTRUK)
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



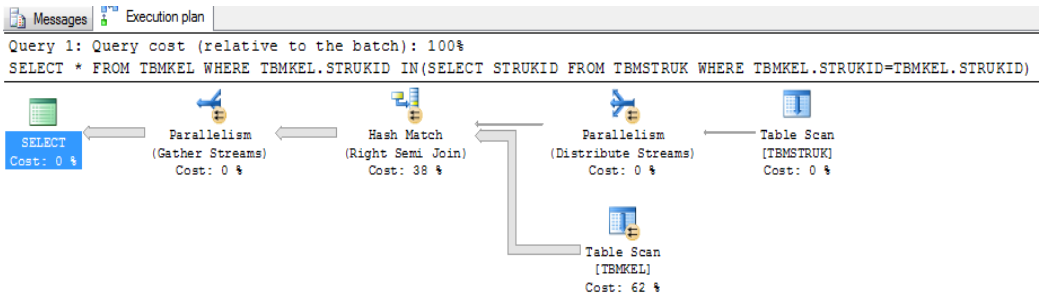
Gambar 14. *Display Estimated Execution Plan Nested Join Scalar* 1 Relasi

Query pada gambar 15 berdasarkan *execution plan* diekseksui dengan *right semi join*berarti query diproses disub querykan terlebih dahulu sehingga akan lebih cepat kinerjanya dalam proses pencarian data dikarenakan melakukan filter terlebih dahulu sebelum penggabungan tabel.

Query Nested Join Correlated yang diuji dan analisa sebagai berikut:

```
SELECT * FROM TBMKEL WHERE TBMKEL.STRUKID IN(SELECT STRUKID
FROM TBMSTRUK WHERE TBMKEL.STRUKID=TBMKEL.STRUKID)
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



Gambar 15. *Display Estimated Execution Plan Nested Join Correlated* 1 Relasi

Berdasarkan gambar 16 *Estimated Execution Plan Nested Join correlated* tidak ada bedanya dengan yang bukan correlated hanya akan lebih lama dalam proses pencarian data karena sebelum *subquery* diproses terlebih dahulu setelah diproses disubquery kemudian dijoin sehingga terjadi proses yang panjang, sehingga untuk data besar membutuhkan waktu akan lama.

Ketiga query diatas memilik output informasi yang sama yaitu menampilkan satu tabel secara penuh yaitu tabel TBMKEL.Pengujian query-query tersebut diuji berdasarkan banyaknya jumlah data mulai dari data yang kecil sampai data yang besar dan mengeksekusi query tersebut secara bergiliran.

*Query Hash Join*pada kelompok data kecil waktu yang dibutuhkan untuk mengakses data tidak terlalu besar walaupun kadang waktu yang dibutuhkan tidak stabil. Kelompok data yang sedang lebih baik dari kelompok data kecil dimana tidak ada waktu yang dibutuhkan untuk mengakses data. Kelompok data besar waktu yang dibutuhkan untuk mengakses data lebih besar dibanding kelompok lain karena data yang yang diakses juga semakin besar.

Query Nested Join Scalar pada kelompok data kecil dan kelompok data besar hampir tidak ada nilai dalam pengaksesan data sedangkan kelompok data besar waktu yang dibutuhkan mengakses data lebih besar.

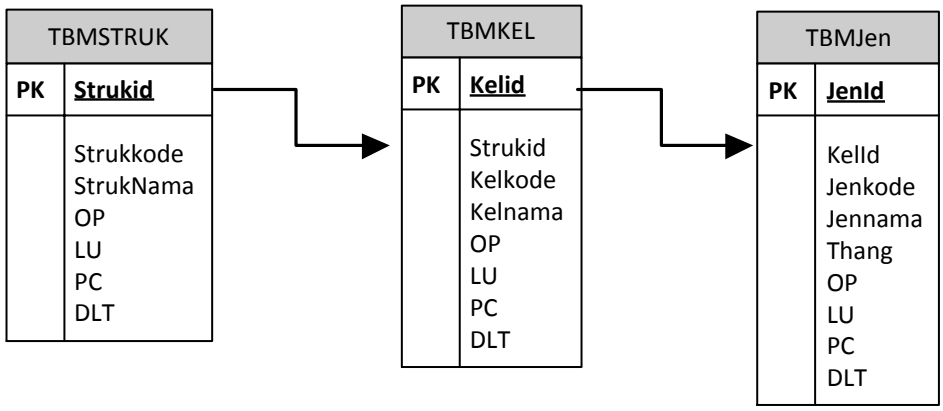
Query Nested Join Correlated pada kelompok data kecil dan kelompok data besar tidak ada nilai dalam pengaksesan data sedangkan kelompok data yang besar semakin besar jumlah datanya semakin besar waktu yang dibutuhkan untuk mengakses data.

Berdasarkan data hasil percobaan diatas maka dapat disimpulkan sebagai berikut:

1. *Query Hash Join* pada jumlah data yang besar 1 relasi waktu yang dibutuhkan untuk mengakses data lebih cepat dibanding *querynested join* baik *scalar* maupun *correlated.*
2. *Query Nested Join* baik *Scalar* maupun *Correlated* pada jumlah data yang kecil dan sedang lebih cepat dibanding *Query Hash join.*

8. Analisis Hasil Pengujian Query 2 Relasi

Tabel yang akan diuji dan direlasikan dapat dilihat sebagai berikut:



Gambar 16. Relasi antar 3 Tabel

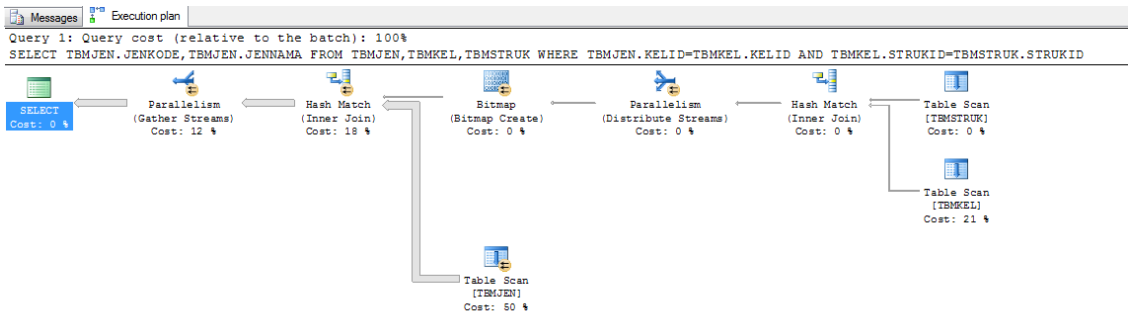
Pengujian pencarian data dengan menghubungkan dengan kedua tabel menggunakan *query hash join*, *nested join*. *Nested join* dibagi dua yaitu *scalar* dan *correlated*.

Query Hash Join yang dianalisa sebagai berikut:

```
SELECT TBMJEN.*FROM TBMJEN, TBMKEL, TBMSTRUK WHERETBMJEN.
KELID=TBMKEL. KELID AND TBMKEL. STRUKID=TBMSTRUK. STRUKID
```

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



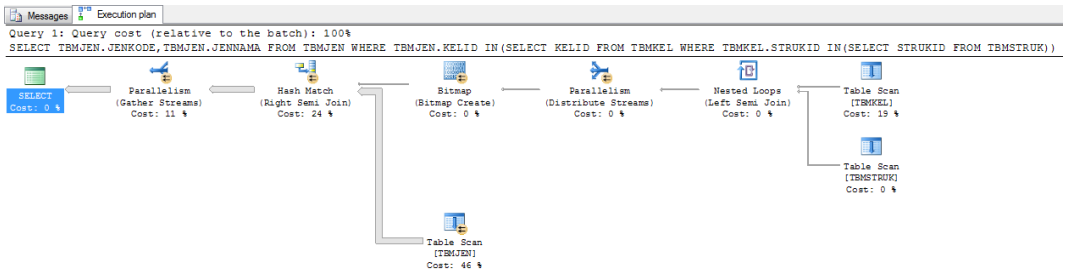
Gambar 17. *Display Estimated Execution Plan Query Hash Join* 2 Relasi

Berdasarkan gambar 18 *query* yang dieksekusi secara *hash join* dengan menggabungkan 3 tabel, *execution plan query* diatas menggabungkan tabel secara *inner join*dan proses pencarian data *query* dengan cara*table scan*.

Query Nested Join Scalar yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMJEN
WHERE TBMJEN. KELID IN(SELECT KELID FROM TBMKEL WHERE TBMKEL.
STRUKID IN(SELECT STRUKID FROM TBMSTRUK))
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



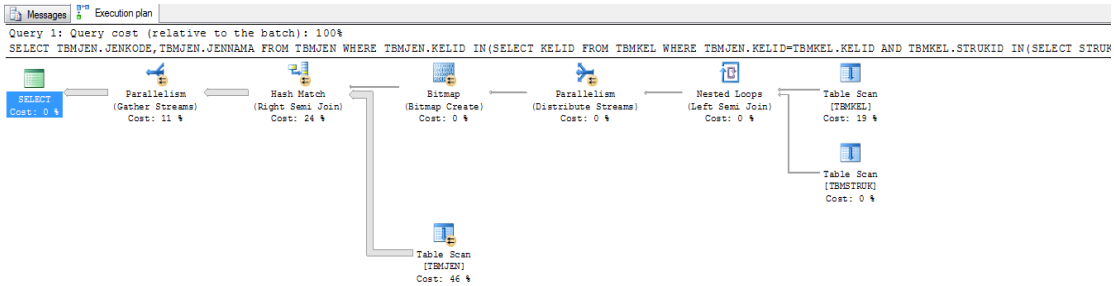
Gambar 18. *Display Estimated Execution Plan Query Nested Join Scalar* 2 Relasi

Berdasarkan gambar 19 query 2 relasi atau 3 tabel yang digabungkan secara *right semi join* dan pencarian data antar tabel menggunakan *table scan*, penggabungan tabel secara semi join dengan membuat partisi terlebih dahulu sebelum penggabungan tabel.

Query Nested Join Correlated yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMJEN WHERE TBMJEN.KELID IN(SELECT KELID FROM
TBMKEL WHERE TBMJEN.KELID=TBMKEL.KELID AND TBMKEL.STRUKID
IN(SELECT STRUKID FROM TBMSTRUK WHERE
TBMKEL.STRUKID=TBMSTRUK.STRUKID))
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



Gambar 19. *Display Estimated Execution Plan Query Nested Join Correlated* 2 Relasi

Berdasarkan gambar 20 penggabungan 3(tiga) tabel dilakukan dengan metode*right semi join*dan proses pencarian data menggunakan*table scan*, jumlah proses tersebut berdasarkan tergantung banyaknya jumlah relasi.

Ketiga query diatas memilik output informasi yang sama yaitu menampilkan satu tabel secara penuh yaitu tabel TBMJEN. Pengujian query-query tersebut diuji berdasarkan banyaknya jumlah data mulai dari data yang kecil sampai data yang besar dan mengeksekusi query tersebut secara bergiliran.

Query Hash Join yang menghubungkan 3(tiga) tabel pada kelompok data kecil terjadi perubahan waktu saat pengaksesan data dibandingkan relasi 2 (dua) tabel menjadi lebih lama. Kelompok data sedang 1 relasi lebih baik dibandingkan 2 relasi dalam pengaksesan data.Kelompok data besar 1 relasi lebih cepat dibandingkan 2 relasi.

Query Nested Join Scalar yang menghubungkan 3(tiga) tabel pada kelompok data kecil 1 relasi dan 2 relasi tidak ada perubahaan waktu untuk mengakses data dan kelompok data sedang juga hampir sama tidak ada perubahaan 1 relasi dan 2 relasi tetap stabil akan tetapi pada saat kelompok data besar lebih baik relasi 2 dibandingka relasi 1.

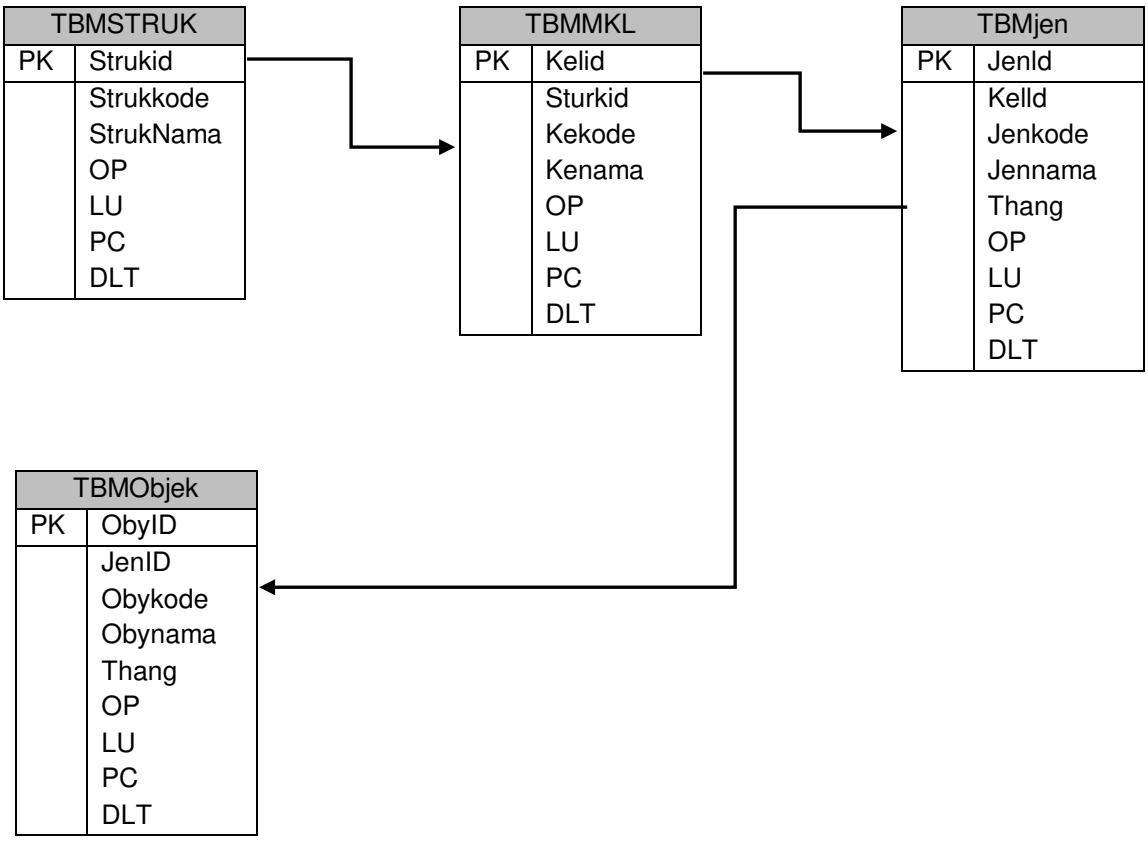
Query Nested Join Correlated yang menghubungkan 3(tiga) tabel pada kelompok data kecil terjadi perubahan dimana 1 relasi lebih stabil dibandingkan 2 relasi, dan kelompok sedang juga sama 1 relasi lebih stabil dibandingkan 2 relasi akan tetapi kelompok data besar lebih baik 2 relasi dibandingkan dengan 1 relasi.

Berdasarkan data hasil percobaan diatas maka dapat disimpulkan sebagai berikut:

1. Kelompok data kecil *query nested join scalar* lebih cepat dibandingkan *Hash Join* dan *Nested Join Correlated*.
2. Kelompok data menengah*query nested join scalar* lebih cepat dibandingkan *hash join* dan *nested correlated*.
3. Kelompok data besar *query nested join scalar* juga lebih cepat dibandingkan *hash join* dan *nested join correlated*.

9. Analisis Hasil Pengujian Query 3 Relasi

Tabel yang akan diuji dan direlasikan dapat dilihat sebagai berikut:

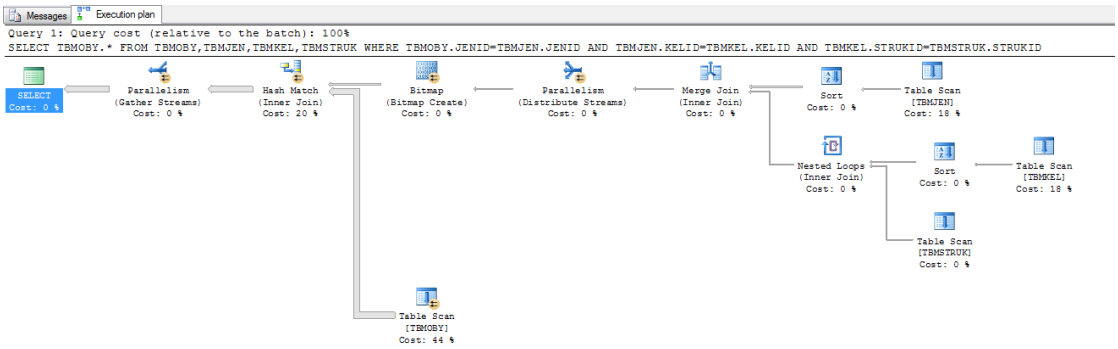


Gambar 20. Relasi antar4 tabel

Query Hash Join yang dianalisa sebagai berikut:

```
SELECT TBMOBY.*FROM TBMOBY,TBMJEN,TBMKEL,TBMSTRUK
WHERE TBMOBY.JENID=TBMJEN.JENID AND TBMJEN.KELID=TBMKEL.KELID
AND TBMKEL.STRUKID=TBMSTRUK.STRUKID
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



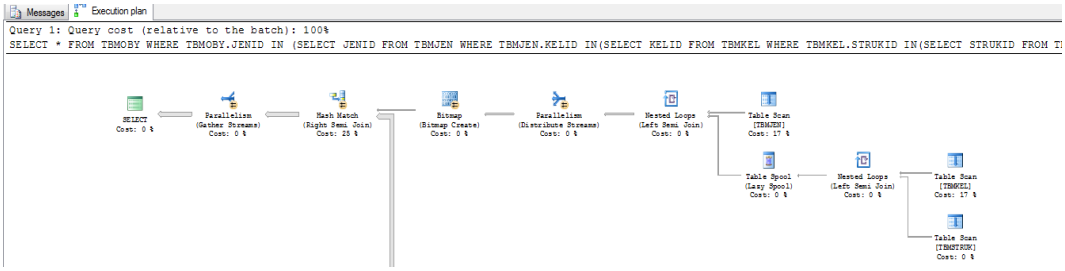
Gambar 21. *Display Estimated Execution Plan Query Hash Join 3* Relasi

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

Query Nested Join Scalar yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMOBY WHERE TBMOBY.JENID IN(SELECT JENID FROM
TBMJEN WHERE TBMJEN.KELID IN(SELECT KELID FROM TBMKEL WHERE
TBMKEL.STRUKID IN(SELECT STRUKID FROM TBMSTRUK)))
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:

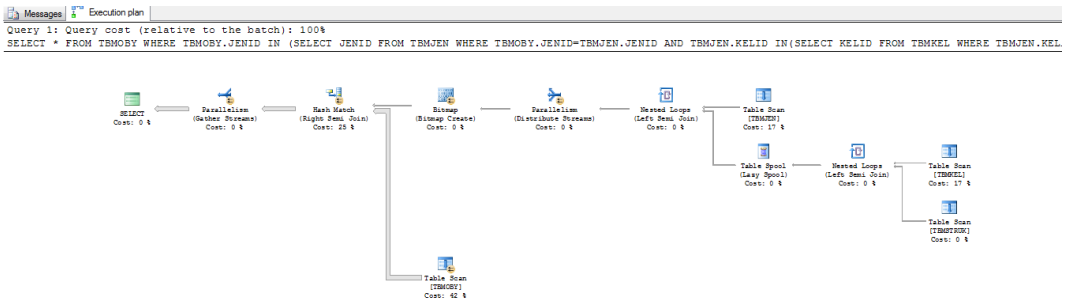


Gambar 22. *Display Estimated Execution Plan Query Nested Join Scalar 3* Relasi

Query Nested Join Correlated yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMOBY WHERE TBMOBY.JENID IN(SELECT JENID FROM
TBMJEN WHERE TBMOBY.JENID=TBMJEN.JENID AND TBMJEN.KELID
IN(SELECT KELID FROM TBMKEL WHERE TBMJEN.KELID=TBMKEL.KELID AND
TBMKEL.STRUKID IN(SELECT STRUKID FROM TBMSTRUK WHERE
TBMKEL.STRUKID=TBMSTRUK.STRUKID)))
```

Analisis query diatas menggunakan Estimated execution plan yang terdapat di Microsoft SQL Server sebagai berikut:



Gambar 23. *Display Estimated Execution Plan Query Nested Join Correlated 3* Relasi

Ketiga query diatas memilik output informasi yang sama yaitu menampilkan satu tabel secara penuh yaitu tabel TBMOBY. Pengujian query-query tersebut diuji berdasarkan banyaknya jumlah data dari data yang kecil sampai data yang besar dan mengeksekusi query tersebut secara bergiliran.

Query Hash Join kelompok data kecil pada relasi ini lebih cepat dibandingkan relasi sebelumnya, kelompok data menengah juga hampir sama dengan kelompok data kecil lebih baik dari relasi sebelumnya dan pada kelompok data besar terjadi perubahan lebih besar waktu yang dibutuhkan untuk mengakses data.

Query Nested Join Scalar kelompok data kecil tidak ada perbedaan dengan relasi-relasi sebelumnya yang terjadi perubahaan pada kelompok menengah lebih lama dibanding relasi sebelumnya dan kelompok data besar lebih cepat dibandingkan relasi sebelumnya.

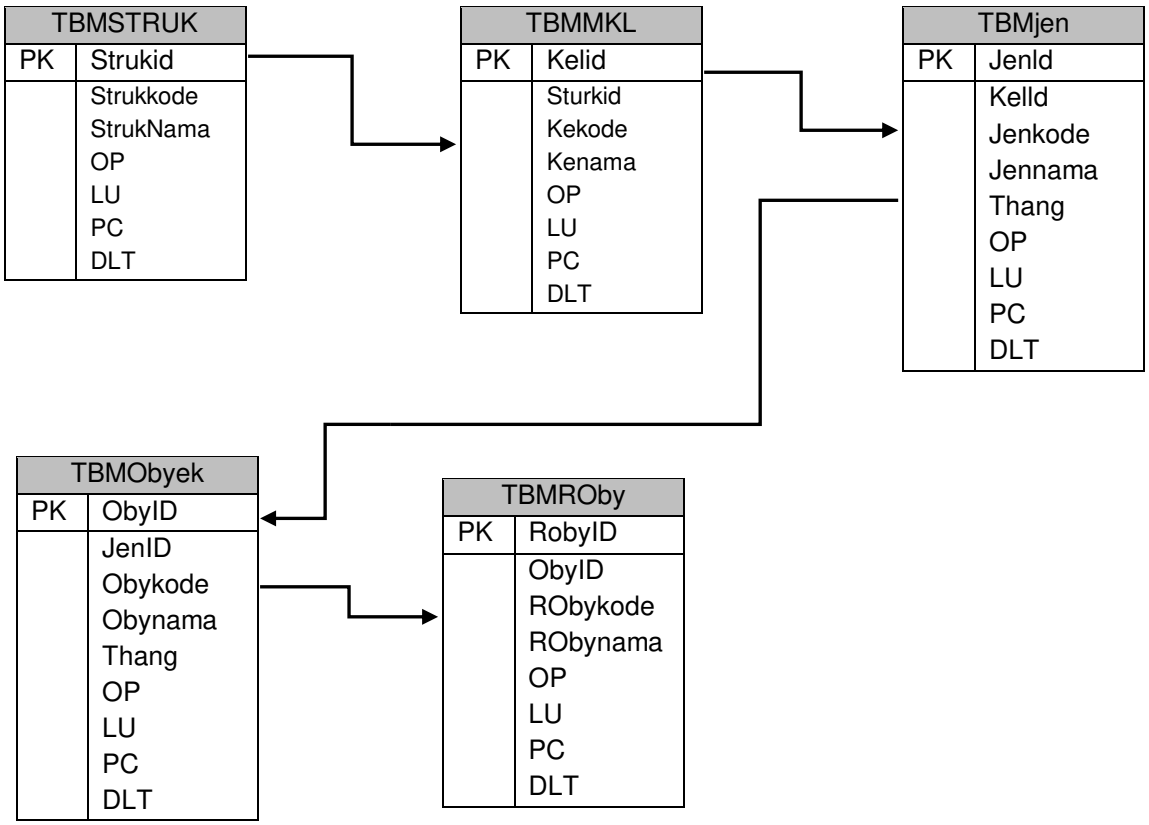
Query Nested Join Correlated kelompok data kecil lebih baik relasi sebelumnya dan begitu juga kelompok data menengah dan besar.

Berdasarkan data hasil percobaan diatas maka dapat disimpulkan sebagai berikut:

1. Kelompok data kecil Query Hash Join dan Query Nested Join Scalar lebih baik dari segi kecepatan waktu pengaksesan data.
2. Kelompok data menengah *Query Hash Join* lebih cepat dibandingkan *query nested join scalar* maupun *query nested correlated*.
3. Kelompok data besar Nested Join Scalar lebih baik dibandingkan Query Hash Join dan tidak ada perbedaan yang besar dengan Query Nested Join Correlated.

10. Analisis Hasil Pengujian Query 4 Relasi

Tabel yang akan diuji dan direlasikan dapat dilihat sebagai berikut:



Gambar 24. Relasi antar 5 tabel

Query Hash Join yang dianalisa sebagai berikut:

```
SELECT TBMROBY.*FROM TBMOBY,TBMJEN,TBMKEL,TBMSTRUK,TBMROBY
WHERE TBMROBY.OBYID=TBMOBY.OBYID AND
TBMOBY.JENID=TBMJEN.JENID AND TBMJEN.KELID=TBMKEL.KELID
AND TBMKEL.STRUKID=TBMSTRUK.STRUKID
```

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

Query Nested Join Scalar yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMROBY WHERE TBMROBY.OBYID IN(SELECT OBYID FROM
TBMOBY WHERE TBMOBY.JENID IN(SELECT JENID FROM TBMJEN WHERE
TBMJEN.KELID IN(SELECT KELID FROM TBMKEL WHERE TBMKEL.STRUKID
IN(SELECT STRUKID FROM TBMSTRUK))))
```

Query Nested Join Correlated yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMROBY WHERE TBMROBY.OBYID IN(SELECT OBYID FROM
TBMOBY WHERE TBMROBY.OBYID=TBMOBY.OBYID AND TBMOBY.JENID
IN(SELECT JENID FROM TBMJEN WHERE TBMOBY.JENID=TBMJEN.JENID AND
TBMJEN.KELID IN(SELECT KELID FROM TBMKEL WHERE
TBMJEN.KELID=TBMKEL.KELID AND TBMKEL.STRUKID IN(SELECT STRUKID
FROM TBMSTRUK WHERE TBMKEL.STRUKID=TBMSTRUK.STRUKID))))
```

Ketiga query diatas memilik output informasi yang sama yaitu menampilkan satu tabel secara penuh yaitu tabel TBMROBY. Pengujian query-query tersebut diuji berdasarkan banyaknya jumlah data mulai dari data yang kecil sampai data yang besar dan mengeksekusi query tersebut secara bergiliran.

*Query Hash Join*pada kelompok data kecil tidak ada perbedaan relasi sebelumnya tidak ada perubahan waktu mengakses data dan kelompok data menengah juga sama tidak ada perubahan waktu untuk mengakses data akan tetapi kelompok data besar waktu yang dibutuhkan cukup besar dibandingkan relasi sebelumnya.

Query Nested Join Scalar ada kelompok data kecil waktu yang dibutuhkan tidak ada perubahan dari relasi sebelumnya untuk data jumlah data yang kecil tetap konsisten waktu dalam pengaksesan data akan tetapi ada perubahan pada kelompok data menengah dimana waktu lebih lama dibandingkan sebelumnya dan kelompok data besar juga hampir sama terjadi perubahan waktu lebih lama dibandingkan relasi sebelumnya.

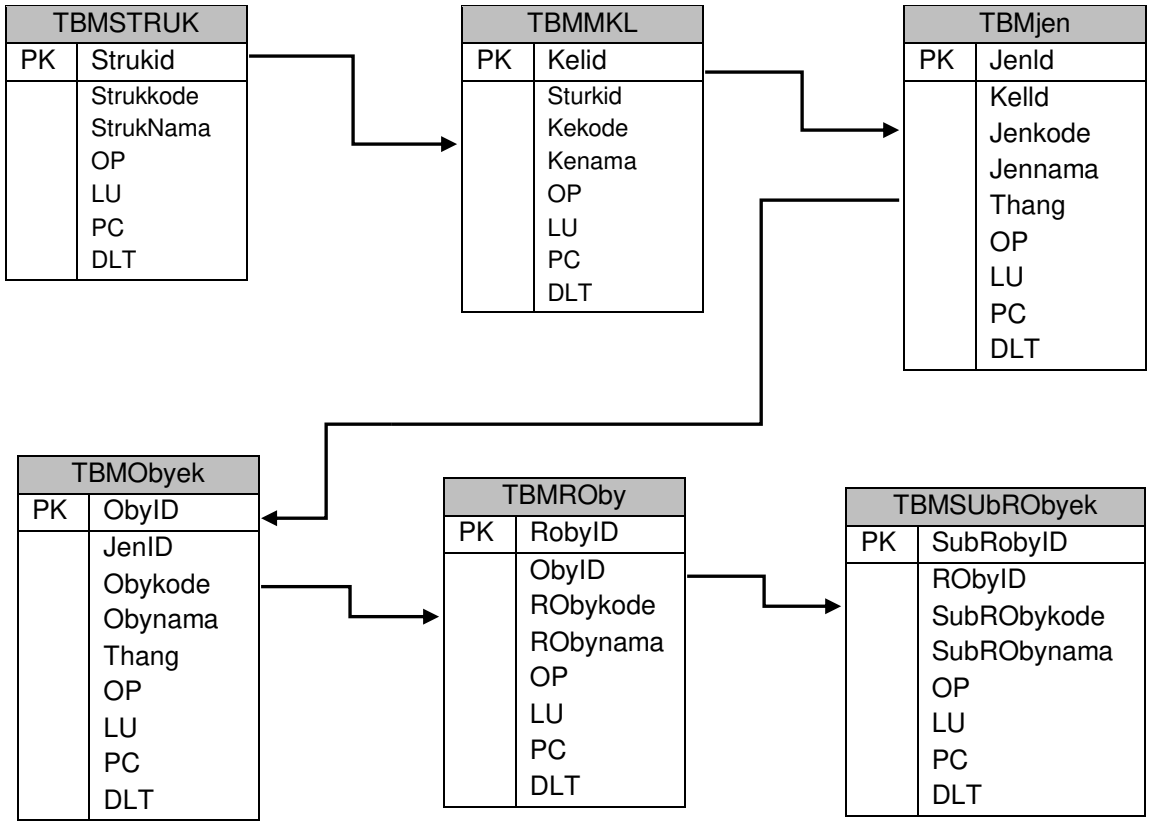
*Query Nested Join Correlated*pada kelompok lebih cepat dibandingkan relasi sebelumnya dan pada kelompok data menengah tidak ada perubahan waktu akan tetapi terjadi perubahan besar pada kelompok data yang besar waktu semakin lama dibandingkan dari relasi sebelumnya akan tetapi lebih baik dibandingkan query lainnya.

Berdasarkan data hasil percobaan diatas maka dapat disimpulkan sebagai berikut:

1. Kelompok data kecil ketiga query hampir sama dalam kecepatan waktu untuk mengakses data.
2. Kelompok data menengah query hash join lebih baik dibandigkan query nested join scalar dan nestd join correlated.
3. Kelompok data besar *query nested join correlated* lebih baik dibandingkan *nested join scalar* dan *hash join*.

11. Analisis Hasil Pengujian Query 5 Relasi

Tabel yang akan diuji dan direlasikan dapat dilihat sebagai berikut:



Gambar 25. Relasi antar 6 tabel

Query Hash Join yang dianalisa sebagai berikut:

```
SELECT TBMSUBROBY.* FROM
TBMOBY,TBMJEN,TBMKEL,TBMSTRUK,TBMROBY,TBMSUBROBY
WHERE TBMSUBROBY.ROBYID=TBMROBY.ROBYID AND
TBMROBY.OBYID=TBMOBY.OBYID AND TBMOBY.JENID=TBMJEN.JENID AND
TBMJEN.KELID=TBMKEL.KELID
AND TBMKEL.STRUKID=TBMSTRUK.STRUKID
```

Query Nested Join Scalar yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMSUBROBY WHERE TBMSUBROBY.ROBYID IN(SELECT
ROBYID FROM TBMROBY WHERE TBMROBY.OBYID IN(SELECT OBYID FROM
TBMOBY WHERE TBMROBY.OBYID=TBMOBY.OBYID AND TBMOBY.JENID
IN(SELECT JENID FROM TBMJEN WHERE TBMOBY.JENID=TBMJEN.JENID AND
TBMJEN.KELID IN(SELECT KELID FROM TBMKEL WHERE
TBMJEN.KELID=TBMKEL.KELID AND TBMKEL.STRUKID IN(SELECT STRUKID
FROM TBMSTRUK s)))))
```

Query Nested Join Correlated yang diuji dan analisa sebagai berikut:

```
SELECT*FROM TBMSUBROBY WHERE TBMSUBROBY.ROBYID IN(SELECT
ROBYID FROM TBMROBY WHERE TBMSUBROBY.ROBYID=TBMROBY.ROBYID
AND TBMROBY.OBYID IN(SELECT OBYID FROM TBMOBY WHERE
TBMROBY.OBYID=TBMOBY.OBYID AND TBMOBY.JENID IN(SELECT JENID
FROM TBMJEN WHERE TBMOBY.JENID=TBMJEN.JENID AND TBMJEN.KELID
```

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

```
IN(SELECT KELID FROM TBMKEL WHERE TBMJEN.KELID=TBMKEL.KELID AND
TBMKEL.STRUKID IN(SELECT STRUKID FROM TBMSTRUK WHERE
TBMKEL.STRUKID=TBMSTRUK.STRUKID)))))
```

Ketiga query diatas memilik output informasi yang sama yaitu menampilkan satu tabel secara penuh yaitu tabel TBMSUBOBY. Pengujian query-query tersebut diuji berdasarkan banyaknya jumlah data mulai dari data yang kecil sampai data yang besar dan mengeksekusi query tersebut secara bergiliran.

Query Hash Join pada kelompok data kecil tidak ada perubahan dari relasi sebelumnya tetap konsisten dari waktu mengakses data, kelompok data menengah juga hampir sama tidak ada perubahan waktu tetap stabil akan tetapi kelompok data besar lebih cepat dibandingkan relasi sebelumnya.

Query Nested Join Scalar pada kelompok data kecil terjadi perubahan dari relasi sebelumnya akan tetapi tidak terlalu besar, kelompok data menengah lebih lama dibandingkan relasi sebelumnya akan tetapi kelompok data besar terjadi perubahan lebih cepat dibandingkan relasi sebelumnya.

Query Nested Join Correlated pada kelompok data kecil dan menengah tetap konsisten seperti relasi sebelumnya tidak terjadi perubahan waktu, pada kelompok data besar waktu lebih cepat dibandingkan relasi sebelumnya.

Berdasarkan data hasil percobaan diatas maka dapat disimpulkan sebagai berikut:

1. Kelompok data kecil query hash join dan nested join hampir sama dalam kecepatan waktu untuk mengakses data lebih baik dibandingkan query nested join scalar.
2. Kelompok data menengah *query hash join* lebih baik dibandigkan query *nested join scalar* dan *nestd join correlated*.
3. Kelompok data besar *query nested join scalar* lebih baik dibandingkan *nested join correlated*dan *hash join*.

KESIMPULAN

Query adalah perintah SQL yang mempunyai peran penting dalam database manajemen sistem (DBMS), namun penggunaan *query* haruslah tepat. Faktor yang mempengaruhi terhadap kecepatan akses data tidak hanya pada perintah *Query* saja akan tetapi terhadap hal-hal lain yang berpengaruh, seperti aplikasi, jaringan komputer, struktur fisik penyimpanan database, *hardware*, desain logik database, sistem operasi, *cluster* atau index *database*.

Pencarian data menggunakan *Query* atau perintah SQL banyak algoritma atau *query* bisa dilakukan dengan output yang sama. Beberapa algoritma *query* yang banyak digunakan yaitu *Query Hash Join* dan *Query Nested Join*. Penggunaan *Query* bukan saja digunakan pada database manajemen sistem saja akan tetapi disebuah aplikasi dalam pemrosesan data atau pengolahan data menggunakan *query* akan lebih mudah dan cepat, setiap algoritma *query* memiliki *running time* atau kecepatan waktu melakukan respon data apalagi aplikasi berbasis *client server*.

Setelah penulis melakukan penelitian dengan pengujian *query Hash join* dan *Query Nested* dengan aplikasi yang dirancang menggunakan *Microsoft Visual Studi 2010* dan *Microsoft SQL Server 2008* sebagai database. Pengujian running time atau kecepatan waktu merespon data yang dilakukan untuk pemrosesan data khususnya pencarian data berdasarkan jumlah tabel yang direalisasikan, jumlah baris/record maka penulis membuat kesimpulan sebagai berikut:

Kelompok data kecil jumlah data antara 10-320 baris *Query Hash join* lebih baik dibandingkan *Nested Join*. Kelompok data besar jumlah data antara 40960-1310720 baris *query Nested join scalar* lebih baik dibandingkan *query Hash Join*.

Jurnal Teknovasi
Volume 01, Nomor 2, 2014, 71 –93
ISSN : 2355-701X

DAFTAR PUSTAKA

Aris A.b, (2008), *Optimisasi Query Pada Sistem Database Paralel*, Tersedia: http://xuyas.wordpress.com

Bratbergsengen K., (1984), *Hashing Methods and Relational Algebra Operations,* Proc. 10th VLDB.

DeWitt D.J., Katz R.H., Olken F., Shapiro L.D., Stonebraker M., Wood D., (1984), *Implementation Techniques for Main Memory Database Systems,* Proc. ACM SIGMOD Conference.

DeWitt D.J., Gerber R.H., (1985), *Multiprocessor Hash-Based Join Algorithms,* Proc. VLDB.

Gerber R. H., (1986), *Dataflow Query Processing Using Multiprocessor Hash-Partitioned Algorithms*, Dissertation, University of Wisconsin-Madison, Computer Sciences Technical Report.

Immanuel Chan, (2008), *Oracle Database Performance Tuning Guide, 10g Release 2 (10.2),* Redwood City: CA.

Kitsuregawa M., Tanaka H., Moto-oka T., (1983), *Application of Hash to Data BaseMachine and Its Architecture*, New Generation Computing 1, pp.

Korth, H.F, dan Silberschatz, A, (1991), *Database System Concepts*, McGraw Hill, Singapura.

Kusrini, (2006), *Optimasi Query Untuk Pencarian Data dengan Subset Query*, Bandung.

Metta Santiputri, Mira Chandra Kirani, Anni, (2010), *Perbandingan Cross-Product Dan Subset Query Pada Multiple Relasi Dengan Metode Cost-Based*, Seminar Nasional Informatika UPN Veteran, Yogyakarta.

Nielsen, Paul., Mike White, and Uttam Parui, (2009), *Microsoft SQL Server 2008 Bible*, Wiley Publishing, Inc.,

Shapiro L.D., (1986), *Join Processing in Large Database Systems with Large Main Memory*, ACM TODS 11,3.

Nakayama M., Kitsuregawa M., Takagi M., (1988), *Hash-Partitioned Join Method Using Dynamic Destaging Strategy*, Proc. 14th VLDB, pp.

Patrick Valduriez, Georges Gardarin, (1984), *Join and Semijoin Algorithms for a Multiprocessor Database Machine*, ACM TODS 9,1.

Pong M., (1988), *NonStop SQL Optimizer: Query Optimization and User Influence*, Tandem Systems Review 4,2 pp: Tandem Computers, Part. No. 13693

Richard Foote, (2009), *OPTIMIZER_INDEKS_CACHING* Parameter. Tersedia: http://richardfoote.wordpress.com.

Sagi Arsyad, (2008), *Pengenalan.NET dan Microsoft Innovation Center*, Universitas Indonesia, Jakarta.

Sandra Cheevers, (2006), *Oracle Database Product Familiy, An Oracle White Paper*, Redwood Shores, CA USA.

Schneider D.A., DeWitt D.J., (1989), *A Performance Evaluation of Four ParallelJoin Algorithms in a Shared-Nothing Multiprocessor Environment,* Proc. ACM SIGMOD Conference: Portland, Oregon.

Setiawan, M.A., (2004), *Optimasi SQL Query untuk Informasi Retrievel pada Aplikasi Berbasis Web*, Proceedings Seminar Nasional Aplikasi Teknologi Informasi UII, Yogyakarta.

Stonebraker M. et.al., (1989), *Parallelism in XPRS*, UC Berkeley, Electronics Research Laboratory, Report M89/16.

Tom Best dan M.J. Billings, (2005), *Oracle Database 10g: Administration Workshop I, Electronic Presentation*, Redwood Shores, California USA.