BAB II

TINJAUAN PUSTAKA

A. Tinjauan Pustaka

Dengan perkembangan teknologi *multi*core prosesor, berbagai sistem dapat melakukan prosesnya dengan lebih cepat, namun juga ada berbagai cara melakukan optimalisasi dengan memanfaatkan sebagian besar daya prosesor, Chen dkk (2009) melakukan penelitian untuk mengoptimalkan kinerja suatu perulangan dengan memanfaatkan perkembangan teknologi *multi*core ini, idenya adalah dengan melakukan efisiensi pada spekulasi saat terjadi paralelisasi di perulangan. Paralelisasi dalam kode program dapat diwujudkan salah satunya adalah dengan menggunakan *multithread*, penelitian ini pernah dilakukan juga oleh Soliman dkk(2009) , dari penelitiannya dia menyatakan bahwa pada intel *multi-core*, tiga bentuk tingkatan paralelisme (instruksi, data, *thread*) dapat dimanfaatkan untukmemberikan peningkatan kinerja menggunakaneksekusi superskalar, komputasi *multi-threading*, dan streaming SIMD(*Single Instruction Multi Data*) ekstensi. Ivanov dkk(2009) juga melakukan penelitian mengenai *multithreading* yang diterapkan pada *multi-core* prosesor, dia membandingkan metode paralelisasi lain yaitu menggunakan OpenMP,dan hasilnya menunjukkan tidak jauh beda dengan menggunakan *multithread*.

Penelitian mengenai *multiprocessor* juga pernah dikemukakan oleh Samaneh dkk(2009), Samaneh meneliti tentang bagaimana mengakses antrian

prosesor menggunakan algoritma *ant*. Samaneh menyatakan bahwa cara untuk melakukan efisiensi terhadap kinerja prosesor adalah dengan memanfaatkan metode antriannya. Pada penelitian-penelitian selanjutnya dilakukan penelitian mengenai eksploitasi sumber daya dalam arsitektur *multi*core ini, seperti Hadjidoukas dkk(2009) yang melakukan penelitian mengenai pengembangan *runtimethread*ing pada paralisme *fine-grain* yang bisa sesuai untuk bentuk arsitektur *multi*-core. Hadjidoukas membangun semacam library baru untuk suatu *threading* yang diintegrasikan dengan runtime teknologi openMP. Hasilnya library ini mampu mengefektifkan paralelisme *fine grain* dengan jumlah *thread* yang banyak. Namun penelitian ini belum mempertimbangkan teknologi NUM(*Non-Uniform Memory Access*) yang digunakan pada sistem CPU saat ini. Penelitian yang hampir sama tentang melakukan eksploitasi arsitektur sistem *multi*core juga dilakukan oleh Schiller dkk(2010) dimana ia memanfaatkan teknologi CellSs sebagai pemrograman untuk *multithread* pada arsitektur *multi*core. Pada penelitian ini dikatakan bahwa pada kasus jumlah *thread* kurang dari 4 *thread*, maka teknologi *CellSs* hasilnya akan menjadi lebih cepat dibanding menggunakan *openMP*. Selain itu Lewis dkk(2011) juga melakukan penelitian menyangkut perkembangan teknologi *multi*core ini yaitu dengan memperkuat perhitungan komputasi dengan mengekspolitasi arsitektur dari sistem prosesor yang modern menggunakan algoritma Paralel QBF Solver. Berkembangannya teknologi *multi*core juga berpengaruh terhadap struktur data, dimana struktur data seperti queue dan stack diberlakukan modifikasi baru agar bisa lebih mengoptimalkan fungsi dari arsitektur prosesor *multi*core(Nir,2011).

Dalam penilitian lainnya Abo-Hamad dkk(2011) melakukan review dari 100 lebih tulisan mengenai optimalisasi khususnya pada kasus rantai pasok. Dia menyatakan bahwa dalam melakukan klasifikasi teknik optimalisasi dalam rangka pengambilan keputusan untuk rantai pasok perlu dipertimbangkan mekanisme optimalisasi, jenis variabel keputusan dan ruang untuk pencarian. Dalam pemetaan yang dibuat, teknik optimalisasi dibagi menjadi 2 bagian yaitu metode pencarian langsung dan metode matematis.

Proses optimalisasi dalam prakteknya bisa menggunakan model matematika salah satunya adalah dengan menggunakan pemrograman linier. McAllister dkk(2008) melakukan penelitian mengenai penggunaan pemrograman linier integer untuk mengatasi masalah optimalisasi pada komputasi biologi sehingga menghasilkan program yang lebih efisien.Pemrograman linier juga digunakan untuk masalah multi obyektif dengan mengubah permasalahan multikriteria ke masalah global optimum(Maimos dkk,2009). Pada perkembangannya teknik pemrograman linierini sering dikombinasikandengan teknik lain seperti algoritma genetik (Hagh dan Galvani, 2009) atau partikel swarm(Cisty,2009). Menggabungkan dua buah algoritma ini bukan berarti mengubahnya namun lebih pada mengkombinasikannya, dimana kedua algoritma ini akan saling melengkapi untuk memecahkan masalah suatu kasus.

Dalam pengembangan optimalisasi suatu sistem kadang-kadang terjadi ada 2 buah obyek yang saling berlawanan, ketika suatu obyek ditingkatkan kemampuan maka ada obyek lain yang berkurang kemampuannya. Kasus seperti ini dipecahkan menggunakan metode optimalisasi *multi*obyektif seperti yang

dilakukan oleh Vergidis dkk(2006) dan Luque dkk(2008). Duvivier dkk(2007) melakukan penelitan pada kasus yang sama pada pengembangan perangkat lunak untuk bagian industri, penelitiannya dihadapkan pada kepentingan untuk melakukan penggunaan kembali dan kemampuan memodulasi fungsi-fungsi.

Optimalisasi-optimalisasi yang ada tidak hanya terjadi pada algoritma suatu program namun pada query basis data. Jananto(2006), melakukan penelitian mengenai optimalisasi basis data dengan menggunakan teknik optimalisasi Rushmore . Teknologi Optimalisasi Query Rushmore sebagai salah satu teknologi untuk meningkatkan kecepatan akses data. Penggunaan *cache* memory untuk menampung sementara hasil query dilakukan karena ada keadaaan dimana sistem mengirimkan query yang sama pada satu atau banyak pengguna pada waktu yang berbeda(Gour1 dkk, 2010) (Selvara dkk, 2006). Selain hanya melakukan optimalisasi, diperlukan juga pengukuran kemampuan eksekusi suatu query secara effisien dan keakurasian sangatlah penting guna membantu sistem pengambil keputusan dan data mining (Surajit dkk, 2007). Melakukan optimalisasi pada suatu query bisa juga dilakukan dengan melakukan paralelisasi. Suri dkk(2008) melakukan penelitian mengenai mekanisme untuk paralel query. Tujuan utama dari paralel query adalah untuk meningkatkan kecepatan, memperbesar dan keluaran yang tinggi. Suri dkk mengembangkan *single* operator paralelisasi untuk proses paralelisasi. Le dkk(2012) juga melakukan penelitian untuk paralelisasi tapi lebih ke *multiquery* pada RDF/SPARQL. Dia mengusulkan input algoritma heuristik dimana bagian dari input batch query menjadi kelompok-kelompok sehingga setiap kelompok query dapat dioptimalkan. Dari

hasil eksperimen menyatakan bahwa teknik ini efektif, efisien, dan terukur. Suatu teknik optimalisasi untuk basis data tidak seterusnya bisa memiliki performa baik pada seluruh lingkungan. Jadi lebih baik mengembangkan teknik adaptif untuk suatu konteks dari pada mengembangkan secara umum(Deshpande dkk, 2007)(Mahajan dkk, 2010).

Dalam mengembangkan suatu teknik optimalisasi perlu adanya suatu analisis yang mendalam mengenai algoritma yang lama dengan algoritma yang dikembangkan. Ada 3 bentuk analisis algoritma yang sering digunakan yaitu matematis, empiris, dan visualisasi algoritma. Analisis dengan menggunakan metode ini pernah dilakukan oleh Lin dan Ke (2011) untuk analisis manfaat dan performa sistem antrian menggunakan algoritma genetik. Lin dan Ke menganalisis metode algoritma genetik ke dalam persamaan matematika, kemudian membuat data empiris dan visualisasi numeris untuk menjelaskan hasil yang diperoleh. Tujuan dari analisis yang mereka lakukan adalah untuk menjelaskan bagaimana algoritma tersebut bisa mencapai titik maksimal dari permasalahan yang ada. Sementara itu Sung dan Yuen (2011) juga melakukan analisis dengan metode persamaan matematika, data empiris dalam bentuk grafik, dan visualisasi hasil untuk algoritma pencarian acak. Objek sasaran analisis penelitian Sung dan Yuen adalah dampak penggunaan memori dengan waktu eksekusi dan seberapa cepat algoritma mencapai titik optimal. Penggunaan ke tiga metode sekaligus untuk menganalisis suatu algoritma pun juga dipakai oleh Wang dkk (2009) untuk pengembangan refisi margin algoritma adaboost yang disebut equilibrium margin. Tujuan Wang dkk melakukan analisis dari algoritma adaboost

ini adalah untuk melakukan pembuktian dan pembenahan kesalahan dari performa algoritma.

Pada penelitian kali ini akan diteliti penggunaan *multithread* untuk mengolah data dalam jumlah besar, dengan mempertimbangkan kecepatan read/write suatu hardisk sehingga meminimalisir efek *bottleneck*. Efek *bottleneck* di sini adalah efek yang menyatakan bahwa proses yang mengantri banyak namun pemrosesnya hanya sedikit seperti pada bentuk botol yang lehernya cenderung lebih kecil dari pada badan botol itu sendiri, sehingga kecepatan eksekusi proses menjadi tidak maksimal.Sesuai dengan pendapat Despandhe dkk(2007) dan Marhajan dkk(2010), pengembangan teknik optimalisasi yang dilakukan dalam penelitian ini bukan untuk kasus-kasus general suatu optimalisasi, namun teknik yang disesuaikan dengan batasan-batasan kondisi permasalahan yang ada pada penilitian ini.Dalam pengembangan teknik optimalisasi ini dilakukan analisis persamaan waktu secara matematis, pengambilan data secara empiris dan pemodelan bentuk teknik paralelisasi yang dilakukan. Tujuan dari analisis ini adalah untuk menjelaskan bagaimana proses optimalisasi bekerja, dan membandingkan antara metode-metode optimalisasi yang digunakan.

B. Landasan Teori

1. Optimalisasi

Optimalisasi adalah proses untuk memaksimalkan atau meminimalkan fungsi objektif yang diinginkan sementara memenuhi batasan-batasan yang berlaku (Belegundi,1999). Algoritma optimalisasi dapat didefinisikan sebagai algoritma

atau metode numerik untuk menemukan nilai x sedemikian hingga menghasilkan f(x) yang bernilai sekecil/sebesar mungkin untuk suatu fungsi f yang diberikan, yang mungkin disertai dengan beberapa batasan pada x(Suyanto,2010). Optimalisasi berdasarkan kecepatan dan ketepatan dibagi menjadi 2 yaitu optimalisasi online dan optimalisasi offline. Optimalisasi online ditujukan untuk permasalahan yang membutuhkan solusi dalam waktu cepat dan biasanya permasalahan tersebut terjadi secara berulang-ulang.Optimalisasi offline ditujukan untuk permasalahan yang membutuhkan solusi tidak dalam waktu cepat dan biasanya terjadi dalam periode waktu yang lama(Suyanto,2010)

2. Optimalisasi Query

Optimalisasi Query adalah suatu proses menganalisis query untuk menentukan sumber-sumber apa saja yang digunakan oleh query tersebut dan apakah penggunaan dari sumber tersebut dapat dikurangi tanpa mengubah output. Bisa juga dikatakan bahwa optimalisasi query adalah sebuah prosedur untuk meningkatkan strategi evaluasi dari suatu query untuk membuat evaluasi tersebut menjadi lebih efektif. Optimalisasi query mencakup beberapa teknik seperti transformasi query ke dalam bentuk logika yang sama, memilih jalan akses yang optimal dan mengoptimumkan penyimpanan data. Tujuan dari optimalisasi query adalah menemukan jalan akses yang termurah untuk meminimumkan total waktu pada saat proses sebuah query(Siallagan dkk, 2008).

3. Analisis Algoritma

Analisis algoritma merupakan suatu cara untuk melakukan investigasi sebuah efisiensi algoritma dengan dua sumber yaitu waktu eksekusi dan ruang

penyimpanan. Macam efisiensi sendiri dalam konteks algoritma ada dua yaitu efisiensi waktu dan efisiensi ruang penyimpanan. Efisiensi waktu menunjukkan seberapa cepat algoritma berjalan, efisiensi ruang berhubungan dengan seberapa hemat ruang penyimpanan yang dibutuhkan. Efisiensi waktu algoritma pada dasarnya diukur sebagai fungsi ukuran input dengan menghitung berapa kali operasi dasar dijalankan. Operasi dasar sendiri adalah operasi yang paling berkontribusi terhadap waktu eksekusi. Efisiensi dari sejumlah besar algoritma masuk ke dalam beberapa kelas: konstanta, logaritmik, linier, n log n, kuadratik, kubik, dan eksponensial. Analisis algoritma sendiri dibagi menjadi 3 jenis yaitu:

a. Analisis Matematis

   Merupakan analisis menggunakan pemodelan matematika untuk fungsi kompleksitas suatu algoritma, dan bisa diikuti dengan mencari kasus terbaik, kasus terburuk dan kasus rata-rata untuk kompleksitasnya.

b. Analisis Empiris

   Merupakan analisis pada suatu algoritma yang dilakukan dengan menjalankan algoritma pada sampel input dan menganalisis data yang diamati. Kemampuannya untuk digunakan pada setiap algoritma merupakan kekuaran utama dari pendekatan ini dan contoh sampel merupakan kelemahan utamanya

c. Visualisasi Algoritma

Penggunaan gambar untuk menyampaikan informasi yang berguna tentang algoritma. Variasinya adalah visualisasi statis dan visualisasi dinamis(Jogiyanto, 2005).

4. Hukum Amdhal

Hukum Amdhal mengkarakterisasi kecepatan maximum (S) yang bisa di dapat oleh n prosesor yang berkolaborasi pada sebuah aplikasi dimana p adalah fraksi dari komputasi yang bisa dieksekusi secara paralel. Asumsikan, untuk menyederhanakan, menggunakan (dinormalkan) waktu 1 untuk sebuah *single* prosesor menyelesaikan komputasi. Dengan n prosesor yang berkaitan, bagian paralel membutuhkan waktu p/n, dan bagian sambungannya menggunakan waktu 1-p. Secara keseluruhan, komputasi yang dipararelisasikan membutuhkan waktu $1 - p + \frac{p}{n}$. Hukum amdahl mengatakan peningkatan kecepatan adalah ratio diantara rentetan waktu pada *single* prosesor dengan waktu paralel: $s = \frac{1}{1-p+\frac{p}{n}}$ dengan kata lain s tidak berbanding lurus dengan n. Misalnya ada sebuah aplikasi dimana 90% proses nya dilakukan paralelisasi dengan menggunakan 10 prosesor dan 10% nya tidak. Dan kenaikan kecepatanya hanya berakhir pada 5 kali lebih cepat. Kemudian jumlah prosesornya dinaikan 2 kali lipat menjadi 20, dan hal ini hanya menyebabkan kenaikannya menjadi 7 kali lipat. Jadi 10% bagian ini membuat kecepatan proses yang dilakukan menurun setengah. 10%yang membuat sulit untuk diparalelkan itu adalah bagian-bagian dari program yang melibatkan interaksi dan koordinasi inter*thread*, yang dalam mesin *multi-core*dilakukan oleh pengaksesan secara bersama-sama data yang di *share*. Lebih baik memfokuskan pada paralelisasi pada 10% ini (Shavit, 2011) .

5. Arsitektur *Multi-core* Prosesor(MCA)

Arsitektur *multi-core* prosesor sekarang sudah banyak diimplementasi pada prosesor dimana-mana, dengan menawarkan layanan yang sangat aksesibel yang berarti mampu meningkatkan kemampuan suatu aplikasi. Melakukan pemrograman secara efisien dalam era *multi-core* prosesor saat ini sangatlah dibutuhkan. Para pemrogram saat ini dihadapkan pada tantangan *high performance computing*(HPC). MCA saat ini bisa dipandang sebagai sebuah sistem simetrik *multi*prosesor(SMP). Secara Tipikal MCA melakukan melakukan *share* terhadap memory dan L2 *cache*. L2 *cache* pada MCA lebih kecil dari pada yang dimiliki oleh SMP, namun justru dengan *cache* memori yang lebih kecil ini konflik pada *cache* dapat lebih mudah terjadi dan mampu berdampak negatif pada performa suatu aplikasi.

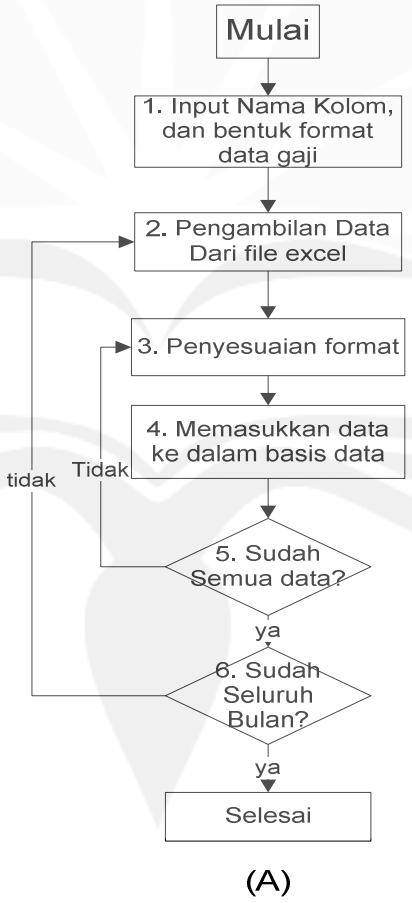
Dalam melakukan HPC diperlukan perhatian pada beberapa karakteristik pada MCA yang mampu menghasilkan dan mengatur efisiensi:

1. *Load Balancing*, merupakan syarat untuk komputasi dengan performa tinggi. Hal ini akan menjadi masalah yang harus diperhatikan seiring munculnya banyaknya inti prosesor yang ada. Misal, suatu eksekusi *nested paralelism* bisa menghabiskan banyak waktu jika tidak memperhatikan keseimbangan beban kerja secara tepat.
2. Besarnya Suatu paralelisme. Semakin banyak *low-overhead* pada *thread* pada beberapa core dapat membantu untuk menyeimbangkan beban kerja. Namun beberapa *thread* yang kurang berkualitas dengan beberapa core dapat membuat ketidak seimbangan beban kerja.

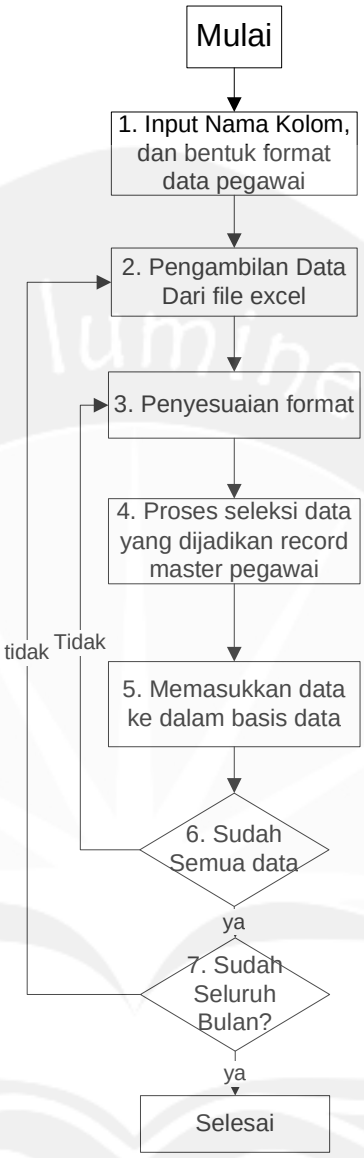
3. Penjadwalan yang efektif. Faktor Akses memori yang tidak seragam akan membantu, khususnya jika jumlah inti prosesor semakin banyak.(Hadjidoukas dkk,2009).

6. Gambaran Umum Sistem

Sistem pengolahan data perpajakan PNS merupakan sebuah sistem untuk mengolah data gaji dan pegawai PNS dalam kurun waktu 1 tahun ditambah dengan gaji ke 13 untuk menghasilkan laporan data perpajakan PNS dalam bentuk rekap 1 tahun dan statistik. Berikut merupakan flowchart dari proses utama dalam system pengolahan data perpajakan PNS ini:



Gambar 1. Proses memasukkan data gaji

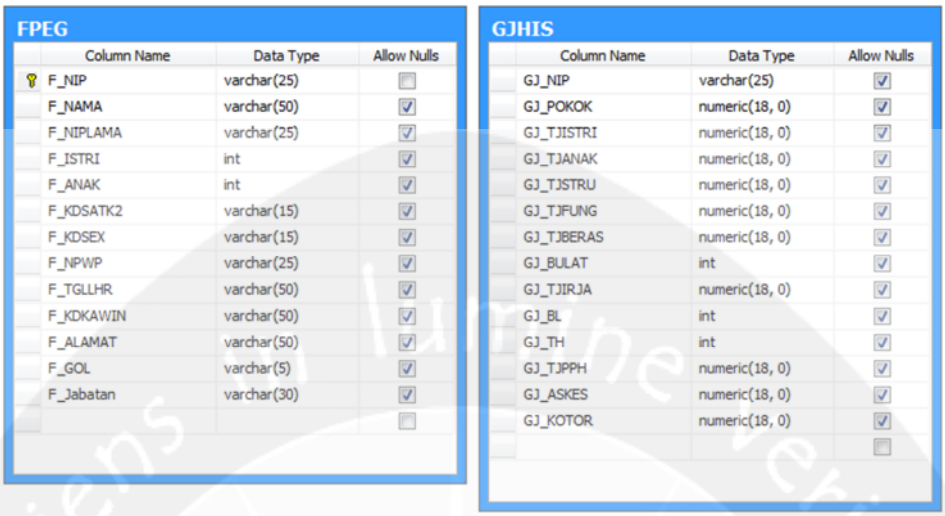


Gambar 2. Proses memasukkan data pegawai

Secara umum proses input data dalam sistem ini merupakan proses yang memakan sebagian besar proses pengolahan data gaji. Proses input data disini terdiri dari input data gaji (Gambar 1) dan input data pegawai (Gambar 2). Proses input data gaji dimulai dengan proses input nama kolom dan bentuk format data gaji. Input nama kolom disini adalah memasukkan nama kolom yang berkaitan

dengan data gaji pada kolom di file excel. Sementara bentuk format file yang ditangani saat ini untuk data gaji terdiri dari 3 bentuk yaitu 1 file gaji yang terdiri dari 13 bulan dalam 13 sheet, 1 file gaji yang terdiri dari 13 bulan dalam 1 sheet, dan 13 file gaji yang terpisah. Setelah proses pengisian ini selesai system akan melakukan import data dari file excel yang sudah didefinisikan tadi, menyesuaikan formatnya sesuai dengan format dari system kemudian memasukkannya ke dalam basis data. Lalu proses akan berulang lagi hingga semua data gaji bulanan telah diproses. Hal yang hampir sama juga terjadi pada proses input data gaji pegawai namun untuk data gaji pegawai sumber data lebih sering menggunakan sumber yang sama dengan data gaji sehingga data sering terjadi duplikasi data pegawai, sehingga perlu melalui proses seleksi data agar data yang berada pada basis data tidak berulang dan untuk proses input data dilakukan dari data gaji bulan ke 12 ke data gaji bulan ke 1 agar data yang dipakai adalah data pegawai terbaru.

Dalam penelitian ini seperti dapat dilihat pada gambar 1 terdapat 2 macam input yang pertama merupakan input data gaji dan pegawai. Dalam sistem ini basis data dapat dibagi kedalam 2 buah tabel yang bersesuaian dengan data yang akan diolah yaitu tabel gaji dan tabel pegawai seperti yang tampak dalam gambar berikut:



FPEG

Column Name	Data Type	Allow Nulls
F_NIP	varchar(25)	☐
F_NAMA	varchar(50)	☑
F_NIPLAMA	varchar(25)	☑
F_ISTRI	int	☑
F_ANAK	int	☑
F_KDSATK2	varchar(15)	☑
F_KDSEX	varchar(15)	☑
F_NPWP	varchar(25)	☑
F_TGLLHR	varchar(50)	☑
F_KDKAWIN	varchar(50)	☑
F_ALAMAT	varchar(50)	☑
F_GOL	varchar(5)	☑
F_Jabatan	varchar(30)	☑
		☐

GJHIS

Column Name	Data Type	Allow Nulls
GJ_NIP	varchar(25)	☑
GJ_POKOK	numeric(18, 0)	☑
GJ_TJISTRI	numeric(18, 0)	☑
GJ_TJANAK	numeric(18, 0)	☑
GJ_TJSTRU	numeric(18, 0)	☑
GJ_TJFUNG	numeric(18, 0)	☑
GJ_TJBERAS	numeric(18, 0)	☑
GJ_BULAT	int	☑
GJ_TJIRJA	numeric(18, 0)	☑
GJ_BL	int	☑
GJ_TH	int	☑
GJ_TJPPH	numeric(18, 0)	☑
GJ_ASKES	numeric(18, 0)	☑
GJ_KOTOR	numeric(18, 0)	☑
		☐

Gambar 3. Struktur Tabel FPEG dan GJHIS padabasis data

Tabel FPEG merupakan tabel pegawai dan GJHIS merupakan tabel gaji. Kedua tabel ini tidak memiliki keterikatan satu sama lain untuk mempercepat proses penyimpanan. Relasi tabel yang berkurang tentunya mengurangi beban dari query saat dijalankan dibanding dengan tabel yang memiliki banyak relasi. Dapat dilihat pada tabel GJHIS tidak memiliki *primary key* dan tabel pada FPEG memiliki. Hal ini yang akan menjadi pembeda nantinya untuk proses optimalisasi dengan menggunakan memori primer dan *bulk insert*. Pada tabel FPEG memiliki dua buah NIP (Nomor Induk Pegawai) karena ada nip baru dan ada nip lama, dimana dalam sistem ini salah satunya saja yang menjadi *primary key*. Sehingga untuk pengecekan NIP dilakukan pada 2 kolom ini.

C. Hipotesis

Dalam penelitian mengenai optimalisasi algoritma query pada kasus eksploitasi sumber daya *multi-core* ini dapat diberikan beberapa hipotesis yaitu:

1. Sistem Pengolah data perpajakan PNS perkabupaten akan bisa dieksekusi dengan lebih cepat
2. Penggunaan prosesor akan maksimal saat eksekusi algoritma dan query optimalisasi yang mengeksplotasi sumber daya *multi-core* prosesor.