Optimasi Query Database Menggunakan Teknik Heuristic

Slamet Sudaryanto N

Abstract : *Query Optimization is process of selecting the most efficient query-evaluation plan from among the many strategies usually possible for processing a given query, especially if the query is complex. We do not expect users to write their queries so that they can be processed efficiently. Rather, we expect the system to construct a query-evaluation plan that minimizes the cost of query evaluation. This is were query optimization comes into play. One Aspect of optimization occurs at the relation algebra level, where the system attemts to find an expression that is equivalent to the given expression, but more efficient execute. Another aspect is selecting a detailed strategy for processing the query, such as choosing the algoritm to use for executing an operation, choosing the specifict indices to use, and so on.*
Kata Kunci : Heuristic, logical query plan, query tree, Optimizer

PENDAHULUAN

Fungsi atau peranan kompter dewasa ini telah mentransformasi atau merabah ke berbagai bidang kehidupan, baik untuk kepentingan bisnis, teknik, hukum, kesehatan, pemerintahan, pendidikan bahkan sampai pada pendukung entertain atau hiburan seperti game dan multi media lainnya. Dengan demikian teknologi komputer memiliki peranan yang penting dalam segi kehidupan. Salah satu peranan yang sangat penting dari teknologi komputer itu sendiri adalah kemampuannya untuk mengelola dan mengolah data dalam rangka menghasilkan informasi yang diharapkan dengan baik, benar, cepat dan tepat. Sehingga unsur-unsur tersebut merupakan keunggulan dalam mencapai derajat kualitas efektive dan efisien dalam system infrormasi yang terintegrasi. Dari unsur pendukung pengelolanan dan pengloaha data dalam rangka menghasilkan informasi tersebut sudah barang tentu secara sepesifik peranan penyimpanan data sangat penting. Komputer tidak hanya mampu menyimpan data, tetapi juga mampu mengambil atau mendapatkan kembali data tersebut. Sifat menyimpan dan mengambil kembali data yang sudah disimpan tersebut membutuhkan pengelolaan yang baik, sehingga didapatkan suatu kemampuan menyimpan dan mendapatkan kembali secara efektif dan efisien. Kemampuan mengelola data tersebut biasa disebuat sebagai suatu database system. Untuk mendukung peranan penyimpanan dan penampilan data tersebut (database system) komputer membutuhkan softaware yang mengatur jalannya data (keluar masuknya data) yang biasa disebut dengan database management system (DBMS). Fasilitas dari DBMS adalah dapat mengakses sebuah database tunggal secara bersamaan oleh banyak user, dapat mengakses data secara terbatas hanya untuk

Slamet Sudaryanto N adalah Dosen Fakultas Ilmu Komputer UDINUS Semarang

user yang berhak dan mengganti kegagalan dari sistem-sistem tanpa kehilangan keutuhan data. Umumnya, interface yang pertama (pokok) untuk sebuah DMBS adalah sebuah high level query atau *data manipulation language* yang dapat dengan mudah digunakan. Contoh dari high level query adalah SQL (Structure Query Language).

Statement – statement (pernyataan-pernyataan) dalam SQL dapat dihasilkan secara langsung oleh user dengan menggunakan command-command (perintah-perintah) interface atau dengan sebuah program aplikasi.

Kehandalan dari suatu sistem database atau DBMS dapat diketahui dari cara kerja optimizer-nya dalam memproses statement-statement SQL yang dibuat oleh user maupun program-program aplikasinya. Di dalam optimizer, statement-statement yang ada diproses dengan salah satu cara dari banyak cara yang ada untuk mendapatkan perencanaan query yang paling optimal sehingga pada akhirnya akan didapatkan jawaban query dengan waktu akses yang paling minimum. Proses untuk mencari perencanaan eksekusi query yang terbaik inilah yang disebut dengan *proses optimisasi query*.

Query optimisasi merupakan sebuah proses memilih query plan yang plaing efisien dari banyak startegi pengaksesan query, khususnya untuk query yang kompleks. Dalam mendapatkan query plan yang efisien pemeriksaan dilakukan mulai dari pemeriksaaan akses path yang mungin (primary index akases, secondary indeks aksess, dan full file scan), dan juga variasi teknik join tabel relasional. Tujuan dari optimasisi adalah untuk mengurangi sebanyak mungkin tuple atau baris yang tidak dibutuhkan.

Jika kita sudah memiliki query, maka kita bisa mengoptimasi query dengan cara mentransform query tersebut. Untuk memeriksa apakah query yang dioptimasi hasilnya sama dengan query yang biasa, ada beberapa rules equivalence. Rule equivalence ini juga bisa sebagai langkah awal untuk mentransform query sehingga lebih efisien.

PEMBAHASAN

Berikut adalah beberapa rule equivalence yang bisa digunakan:

1. Operasi seleksi konjungtif dapat didekonstruksi menjadi sekumpulan seleksi individual.
2. Operasi Seleksi adalah kmutatif
3. Hanya operasi final dari operasi squence proyeksi yang dibutuhkan, yang lainnya dapat diabaikan.
4. Operasi dapat dikombinasikan dengan cartesian product dan theta join.
5. Operasi theta join komutatif
6. Operasi seleksi dapat didistribusikan diantara operasi uniun intersection dan set difference.
7. Sekumpulan union dan intersection asosiatif.
8. Operasi join didistribusikan.

A. Teknik Optimasi Query Heuristic

Ada bermacam-macam teknik yang dapat digunakan untuk melakukan optimisasi query. Masing-masing teknik mempunyai cara sendiri-sendiri dalam mengoptimisasi query. Teknik optimisasi query dapat juga dikatakan

sebagai tahapan-tahapan proses yang dilakukan untuk membuat sebuah query tree menjadi lebih optimal. Ada bermacam-macam teknik yang digunakan untuk mengoptimisasi query, tetapi pada dasarnya ada dua teknik utama yang umumnya digunakan dalam proses optimisasi query. Dua teknik tersebut adalah *Heuristic Optimization* dan *Cost Based optimization*. Pembahasan secara lebih detail pada pembahasan ini adalah dengan menggunakan teknik heuristic optimization.

Heuristic Optimization atau yang biasanya disebut dengan rule based optimization adalah optimisasi query dengan menggunakan aturan-aturan heuristik dan dijalankan pada *logical query plan* (rencana query secara logika) yang terdiri dari urutan operasi-operasi relasional yang biasanya digambarkan sebagai *query tree*. Query Optimizer mendapatkan sebuah inisial plan dari parser dan menggunakan aturan-aturan heuristik untuk mentransformasikan sebuah query ke dalam sebuah bentuk yang sama sehingga dapat diproses dengan lebih efisien. Adapun tujuan dari transformasi tersebut adalah :

- Standarisasi, yaitu mentransformasikan sebuah query ke dalam sebuah bentuk standar tanpa optimisasi.
- Simplifikasi, yaitu mengeliminasi kelebihan dalam sebuah query.
- Ameliorasi, yaitu menyusun ekspresi-ekspresi yang sudah dihasilkan dengan baik untuk mengevaluasi bentuk.

Ada banyak aturan untuk mentransformasikan operasi-operasi relasi aljabar ke dalam suatu persamaan. Berikut ini adalah beberapa aturan-aturan transformasi untuk operasi-operasi relasi aljabar :

1. Pengurutan σ : sebuah pilihan kondisi konjungtif dapat dipisahkan ke dalam sebuah urut-urutan dari operasi-operasi σ tersendiri :
   $^{\sigma}$c1 AND c2 AND … AND cn $^{(R)\equiv}$ $\sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$
2. Perubahan σ : Operasi σ dirubah menjadi :
   $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$
3. Pengurutan π : Dalam sebuah urutan dari operasi-operasi π, semuanya, tetapi yang terakhir dapat diabaikan :
   $\pi_{List1}(\pi_{List2}(\ldots(\pi_{Listn}(R))\ldots)) \equiv \pi_{List1}(R)$
4. Merubah σ dengan π : Jika kondisi pilihan c hanya meliputi attribute-attribute A1,…,An dalam daftar proyeksi, maka kedua operasi dapat dirubah menjadi :
   $\pi_{A1,A2,\ldots,An}(\sigma_c(R)) \equiv \sigma_c(\pi_{A1,A2,\ldots,An}(R))$
5. Perubahan dari $\bowtie$ (dan x) : operasi $\bowtie$ dirubah sebagaimana adanya operasi x:
   $R \bowtie_c S \equiv S \bowtie_c R$
   $R \times S \equiv S \times R$
   Perlu diperhatikan bahwa meskipun urutan dari attribute-attribute mungkin tidak sama dalam relasi yang dihasilkan dari kedua join (atau kedua cartesian product) tetapi artinya adalah sama karena urutan dari attribute-attribute tidaklah penting dalam definisi pilihan dari relasi.
6. Merubah σ dengan $\bowtie$ (atau x) : Jika semua attribute-attribute dalam pilihan kondisi c hanyalah meliputi attribute-attribute dari satu relasi yang digabungkan (misalnya R), maka kedua operasi-operasi tersebut dapat dirubah seperti berikut ini :
   $\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$

Sebagai alternatif, apabila pilihan kondisi c dapat dituliskan sebagai (c1 dan c2), dimana kondisi c1 hanya meliputi attribute-attribute dari R dan kondisi c2 hanya meliputi attribute-attribute dari S, maka operasi-operasi dirubah seperti berikut :

$\sigma_c (R \bowtie S) \equiv (\sigma_{c1} (R)) \bowtie (\sigma_{c2} (S))$

Aturan yang sama dipakai apabila $\bowtie$ digantikan oleh operasi x.

7. Merubah π dengan $\bowtie$ (atau x) : Anggap bahwa daftar proyeksi adalah $L = \{A_1,\ldots, A_n, B_1,\ldots, B_m\}$ di mana $A_1,\ldots, A_n$ adalah attribute dari R dan $B_1,\ldots, B_m$ adalah attribute dari S. Apabila kondisi gabungan c hanya meliputi attribute pada L, maka kedua operasi dapat dirubah sebagai berikut :

   $\pi_L (R \bowtie_c S) \equiv ((\pi_{A1,\ldots,An,\ An+1,\ldots,An+k} (R)) \bowtie_c (\pi_{B1,\ldots,Bn,\ Bn+1,\ldots,Bn+k} (S)))$

   Untuk x, tidak ada kondisi c, jadi aturan transformasi yang pertama adalah selalu menggunakan penggantian $\bowtie_c$ dengan x.
8. Perubahan sekumpulan operasi-operasi : Kumpulan operasi $\cup$ dan $\cap$ adalah perubahan, tetapi $-$ adalah bukan perubahan.
9. Penggabungan $\bowtie$, x, $\cup$ dan $\cap$: Keempat operasi ini adalah gabungan dari individu. Maka dari itu, apabila θ berdiri untuk salah satu dari keempat operasi tersebut (sepanjang ekspresi) maka : $(R \theta S) \theta T \equiv R \theta (S \theta T)$
10. Merubah σ dengan sekumpulan operasi-operasi : operasi σ dirubah dengan $\cup$, $\cap$, dan $-$. Apabila θ berdiri untuk salah satu dari ketiga operasi tersebut (sepanjang ekspresi), maka : $\pi_c (R \theta S) \equiv (\pi_c (R)) \theta (\pi_c (S))$
11. Operasi π dirubah dengan $\cup$: $\pi_L (R \cup S) \equiv (\pi_L (R)) \cup (\pi_L (S))$
12. Mengkonversikan sebuah urutan (σ, x) ke dalam $\bowtie$: Jika kondisi c dari sebuah σ yang mengikuti sebuah x cocok untuk sebuah kondisi join, maka urutan (σ, x) dikonversikan ke dalam sebuah $\bowtie$ sebagai berikut :

    $\sigma_c (R \times S)) \equiv (R \bowtie_c S)$.

Untuk melakukan transformasi-transformasi tersebut, query optimizer harus mengetahui transformasi mana yang sah yaitu yang menghasilkan sebuah hasil yang sama. Di samping transformasi-transformasi yang sah, sebuah optimizer juga harus mengetahui bilamana aturan-aturan tersebut digunakan untuk query. Setelah transformasi-transformasi tersebut, optimizer menaruh kembali operasi-operasi relasi pada query tree dengan operasi-operasi fisik yang dapat dipakai untuk membuat rencana eksekusi. Garis besar dari algoritma optimisasi aljabar heuristik adalah menggunakan beberapa aturan-aturan transformasi relasi aljabar untuk mentransformasikan sebuah inisial query tree (bentuk query tree yang belum dioptimisasi) ke dalam sebuah tree yang optimal dan yang lebih efisien untuk dijalankan. Adapun algoritma dari optimisasi heuristik secara umum adalah :

Langkah 1 : Dengan menggunakan aturan transformasi 1, pisahkan beberapa operasi SELECT dengan kondisi-kondisi konjungtif ke dalam uraian dari operasi-operasi SELECT.

Langkah 2 : Dengan menggunakan aturan transformasi 2, 4, 6 dan 10 perhatikan perubahan dari SELECT dengan operasi-operasi lainnya, pindahkan tiap operasi SELECT sejauh mungkin ke bawah query tree selama diperbolehkan oleh attribute-attribute yang rumit dalam kondisi SELECT.

Langkah 3 : Dengan menggunakan aturan tranformasi 5 dan 9, merubah dan mengumpulkan operasi-operasi binary, susun kembali node-node leaf dari tree menggunakan kriteria-kriteria sebagai berikut. Pertama, tempatkan relasi-relasi node leaf dengan sebagian besar batasan operasi-operasi

SELECT sehingga relasi-relasi node leaf dengan sebagian besar batasan operasi-operasi SELECT dieksekusi terlebih dahulu ke dalam representasi query tree. Definisi dari sebagian besar batasan operasi SELECT salah satunya dapat juga berarti yang menghasilkan sebuah relasi dengan tuple-tuple yang paling sedikit atau dengan ukuran yang mutlak. Kemungkinan lainnya adalah untuk menetapkan sebagian besar batasan SELECT sebagai salah satunya dengan selectivity yang terkecil. Hal ini adalah lebih praktis karena perkiraan dari selectivity-selectivity biasanya tersedia dalam katalog DBMS. Yang kedua, pastikan bahwa urutan dari node-node leaf tidak menyebabkan operasi-operasi CARTESIAN PRODUCT. Sebagai contoh, apabila dua relasi dengan sebagian besar batasan SELECT tidak mempunyai kondisi join secara langsung diantara keduanya, maka diperlukan sekali untuk merubah urutan dari node-node leaf untuk menghindari cartesian product.

Langkah 4: Dengan menggunakan aturan transformasi 12, kombinasikan operasi CARTESIAN PRODUCT dengan sebuah operasi SELECT yang berikutnya pada tree ke dalam sebuah operasi JOIN, apabila kondisi menggambarkan sebuah kondisi join.

Langkah 5 : Dengan menggunakan aturan transformasi 3, 4, 7, 11 perhatikan uraian dari PROJECT dengan operasi-operasi lain, pisahkan dan pindahkan daftar-daftar proyeksi attribute-attribute ke bawah tree sejauh mungkin dengan membentuk operasi-operasi PROJECT yang baru sesuai dengan keperluan. Hanya attribute-attribute itu yang diperlukan dalam hasil query dan dalam operasi-operasi berikutnya pada query tree harus disimpan setelah masing-masing operasi PROJECT.

Langkah 6: Sebagai langkah terakhir, identifikasikan subtree–subtree yang menggambarkan kelompok-kelompok dari operasi-operasi yang dapat dieksekusi dengan menggunakan algoritma tunggal.

B. Notasi untuk Query Tree dan Query Graph

Sebuah query tree adalah sebuah struktur data tree yang sesuai untuk sebuah ekspresi relasi aljabar. Query tree menggambarkan hubungan-hubungan input query sebagai node-node leaf dan tree dan menggambarkan hubungan operasi-operasi aljabar sebagai node-node internal. Sebuah eksekusi dari query tree terdiri dari pelaksanaan sebuah operasi internal node bilamana operand-operand dari query tree tersedia dan kemudian menggantikan internal node tersebut dengan hubungan yang menghasilkan pelaksanaan operasi. Pelaksanaan akan diakhiri apabila root node dijalankan dan menghasilkan hasil relasi untuk query.

Contoh Kasus Optimasi Query :

Q2 : For every project located in 'Stafford', retrieve the project number, the controlling department number, and department manager's last name, address, and birthdate.
Ekspresi relasi aljabamya sbb:

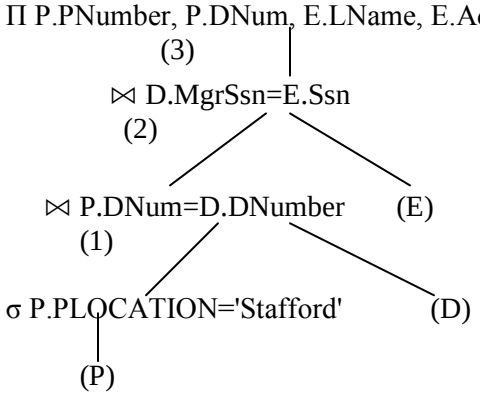
ΠPNumber, DNum, LName, Address, BDate (((σPLocation=’Stafford’(PROJECT))
⋈ DNum=DNumber(DEPARTEMENT) ⋈ MgrSsn=Ssn (EMPLOYEE))

Persamaan ini mengikuti SQL query berikut:

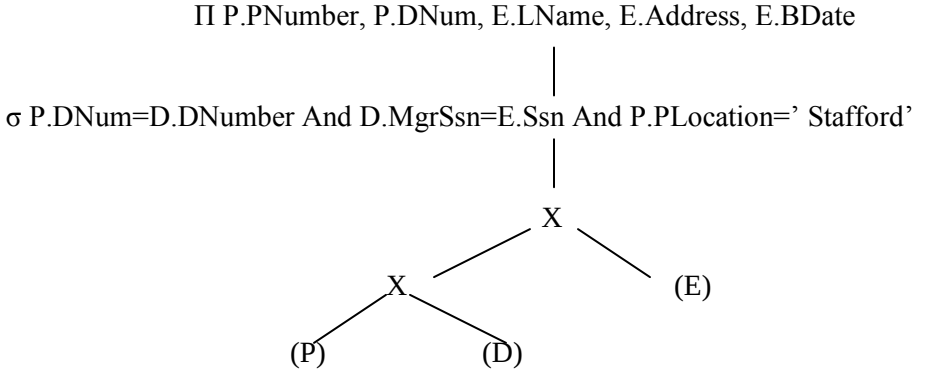
```
SELECT     P.Number, P.DNum, E.LName, E.Adress, E.BDate
FROM       PROJECT AS P, DEPARTMENT AS D, EMPLOYEE AS E
WHERE      P.DNum=D.DNum AND
           D.MgrSsn=E.Sssn AND   P.Location=’Stafford’;
```

Maka Proses Parsing nya akan memiliki beberapa kemungkinan sebagai berikut :

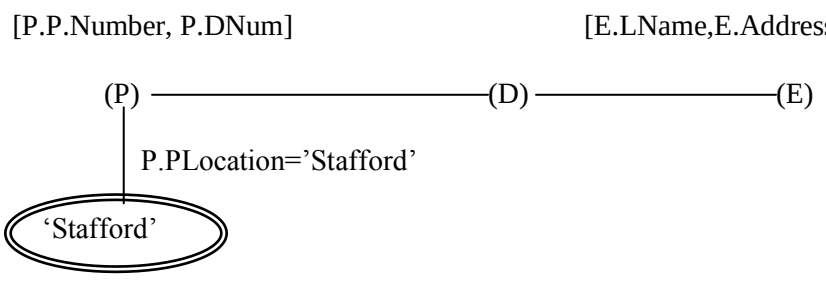
- (a). Query tree yang sesuai dengan ekspresi relasi aljabar untuk Q2



- (b). Inisial (Canonical) query tree untuk SQL query Q2



-(c). Query graph untuk Q2



Pada gambar di atas (a) relasi-relasi tree PROJECT, DEPARTMENT, dan EMPLOYEE digambarkan oleh leaf node P, D, dan E, sementara operasi-operasi relasi aljabar digambarkan oleh internal tree node. Pada saat query tree tersebut dieksekusi, node marked (1) pada gambar (a) harus mulai melakukan eksekusi sebelum node (2) karena beberapa hasil tuple dari operasi (1) harus tersedia sebelum dilakukan operasi eksekusi (2). Dengan cara yang sama, node (2) harus mulai dieksekusi dan menghasilkan hasil sebelum node (3) dapat mulai dieksekusi, dan begitu seterusnya.

Seperti yang dapat dilihat, query tree menggambarkan sebuah perintah khusus dari operasi-operasi untuk mengeksekusi sebuah query. Sebuah gambaran mumi dari sebuah query adalah notasi query graph. Gambar (c) menunjukkan query graph untuk Q2. Hubungan-hubungan dalam query digambarkan oleh relation node yang ditunjukkan dalam sebuah lingkaran. Nilai konstan khususnya dari kondisi-kondisi pilihan query digambarkan oleh constant nodes yang ditunjukkan oleh lingkaran ganda. Kondisi-kondisi pemilihan dan penggabungan digambarkan oleh graph edges, seperti yang terlihat pada gambar (c). Terakhir, attribute-attribute yang akan didapatkan kembali dari tiap relasi ditunjukkan dalam bentuk kurung siku di atas tiap relasi.

Gambar query graph tidak menunjukkan sebuah urutan operasi-operasi yang mula-mula akan dibentuk. Hanya ada sebuah graph tunggal yang sesuai untuk tiap query.

Meskipun beberapa teknik optimisasi berdasarkan pada query graph, tapi pada kenyataannya query tree adalah lebih baik karena dalam penggunaannya, query optimizer perlu untuk menunjukkan perintah-perintah untuk eksekusi query yang tidak mungkin dilakukan dalam query graph.

C. Heuristic Optimization Query Tree

Secara umum, banyak ekspresi-eksprcsi relasi aljabar yang berbeda-beda, karena itu ada banyak query tree yang dapat ekuivalen yaitu dapat sesuai dengan query yang sama. Query parser khusus akan menghasilkan sebuah inisial query tree yang standar untuk mencocokkannya pada sebuah SQL query, tanpa melakukan beberapa optimisasi. Sebagai contoh, untuk sebuah select-project-join query seperti Q2, yang inisial tree-nya ditunjukkan pada gambar (b), CARTESIAN PRODUCT dari relasi-relasi ditentukan dalam klausa FROM yang terlebih dahulu digunakan dan kemudian kondisi-kondisi selection dan join dari klausa WHERE yang digunakan, diikuti oleh proyeksi pada attribute-attribute klausa SELECT. Sebagai sebuah canonical query tree yang menggambarkan sebuah ekspresi relasi aljabar adalah sangat tidak efisien apabila menjalankannya secara langsung, karena operasi-operasi CARTESIAN PRODUCT (X). Sebagai contoh, apabila relasi-relasi seperti PROJECT, DEPARTEMENT dan EMPLOYEE mempunyai ukuran record 100 byte, 50 byte, dan 150 byte dan mengandung 100 tuple, 20 tuple, dan 5000 tuple, berturut-turut, maka hasil dari CARTESIAN PRODUCT akan mengandung 10 milyar tuple dari masing-masing ukuran record 300 byte. Bagaimanapun juga, query tree yang ditunjukkan pada gambar (b) adalah merupakan bentuk standar yang dapat dibentuk dengan mudah. Dan tugas dari heuristic query optimizer adalah mentransformasikan inisial query tree tersebut kedalam query tree akhir yang dapat dieksekusi dengan lebih efisien.

Optimizer harus memasukkan aturan-aturan untuk persamaan diantara ekspresi-ekspresi relasi aljabar yang nantinya dapat dipakai untuk inisial tree. Dan kemudian aturan-aturan Heuristic query optimization memanfaatkan persamaan ekspresi-ekspresi tersebut untuk mentransformasikan inisial tree ke dalam bentuk akhir, yaitu query tree yang sudah dioptimisasi. Berikut ini, akan dijelaskan tentang bagaimana sebuah query tree ditransformasikan dengan menggunakan heuristic.

Diberikan contoh dari transformasi sebuah query Q yang bunyinya sbb:

"Find the last names of employees born after 1957 who work on a project named 'Aquarius'."

Query di atas dapat dispesifikasikan ke dalam SQL seperti berikut ini:

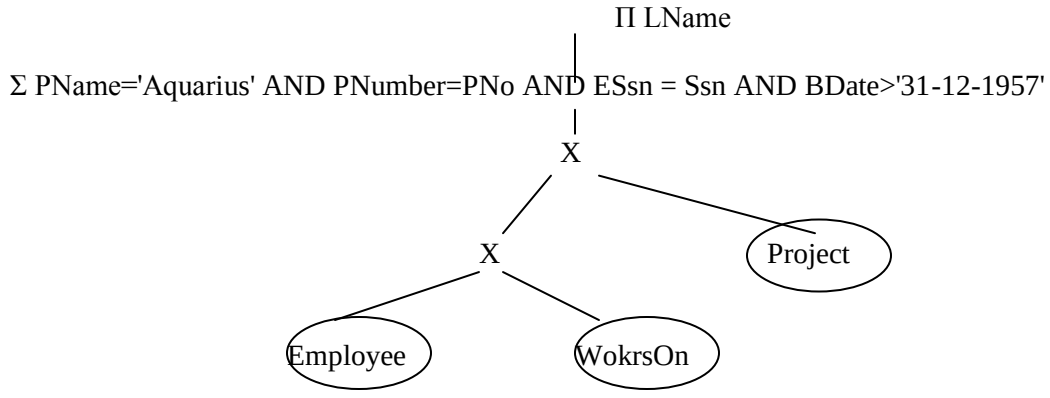
```
Q:  SELECT    LNAME
    FROM      EMPLOYEE, WORKS_ON, PROJECT
    WHERE     PNAME = 'Aquarius' AND PNUMBER = PNO
              AND   ESSN=SSN AND
              BDATE = '31-12-1957';
```

Inisial query tree untuk Q akan ditunjukkan pada gambar (a). Menjalankan tree ini secara langsung mula-mula membentuk sebuah file yang sangat besar yang berisi CARTESIAN PRODUCT dari keseluruhan file-file EMPLOYEE, WORKS_ON, dan PROJECT. Bagaimanapun juga query ini hanya memerlukan satu record dari relasi PROJECT untuk proyek 'Aquarius' dan hanya record EMPLOYEE untuk yang tanggal lahimya setelah '31-12-1957'. Gambar (b) akan menunjukkan perbaikan query tree yang mula-mula menggunakan operasi-operasi SELECT untuk mengurangi banyaknya tuple yang tampak dalam CARTESIAN PRODUCT.

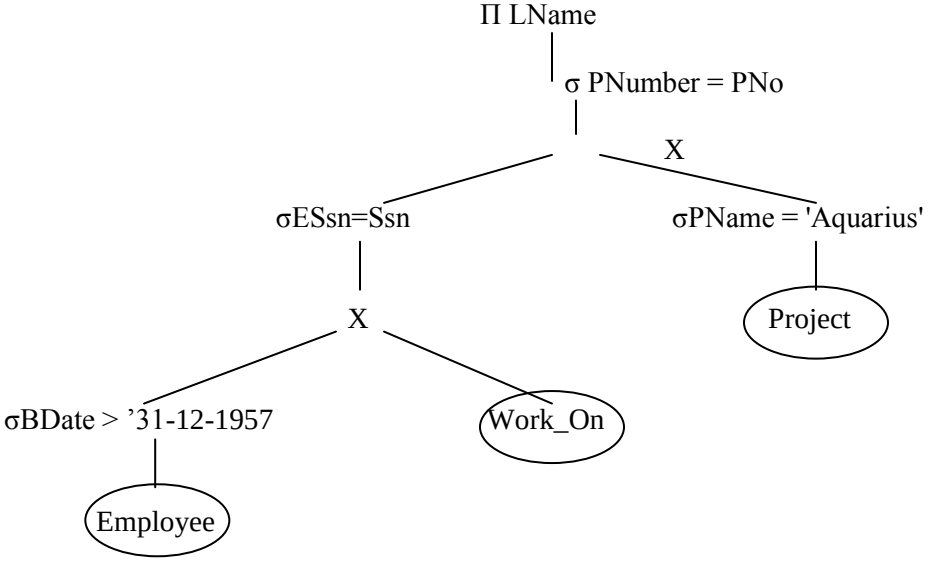
Selanjutnya perbaikan dicapai dengan menukar posisi-posisi dari relasi-relasi EMPLOYEE dan PROJECT dalam tree, seperti yang ditunjukkan pada gambar (c) yang menggunakan informasi bahwa PNUMBER adalah key attribute dari relasi proyek dan oleh sebab itu operasi SELECT pada relasi PROJECT akan mendapatkan kembali hanya sebuah record tunggal. Selanjutnya query tree dapat diperbaiki dengan cara mengembalikan beberapa operasi CARTESIAN PRODUCT yang diikuti dengan sebuah kondisi join dengan sebuah operasi JOIN seperti yang ditunjukkan pada gambar (d).

Perbaikan lainnya adalah hanya menyimpan attribute-attribute yang diperiukan oleh operasi-operasi berikutnya dalam relasi-relasi menengah, dengan memasukkan operasi-operasi PROJECT (Π) dalam query tree seperti yang ditunjukkan pada gambar (e). Hal ini akan mengurangi attribute-attribute (kolom-kolom) dari relasi-relasi menengah sedangkan operasi-operasi SELECT mengurangi nomor tuple (record). Langkah-langkah pengkonversian sebuah query tree selama proses optimisasi heuristic sbb :

- (a). Inisial (canonical) query tree untuk SQL query

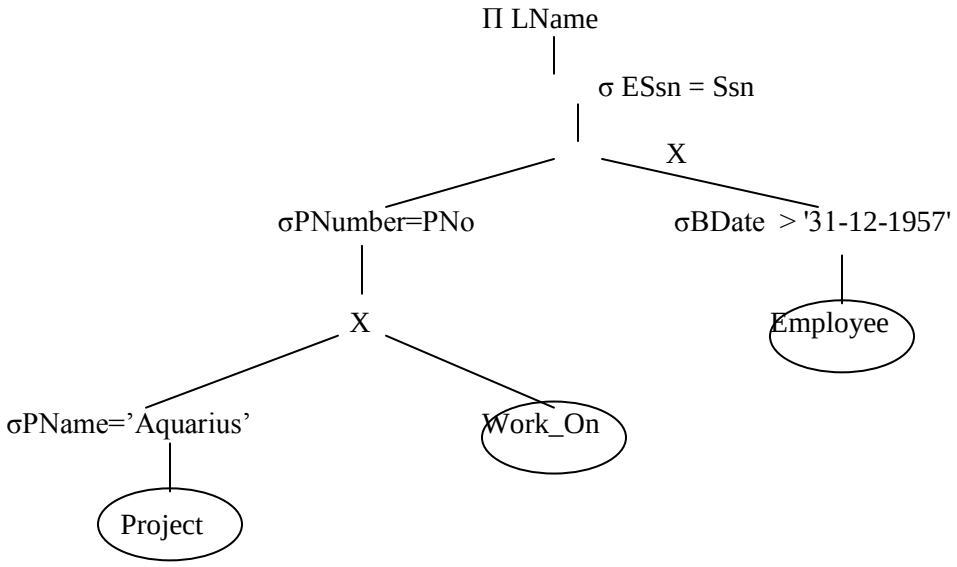


-(b). Memindahkan operasi-operasi SELECT ke dalam query tree



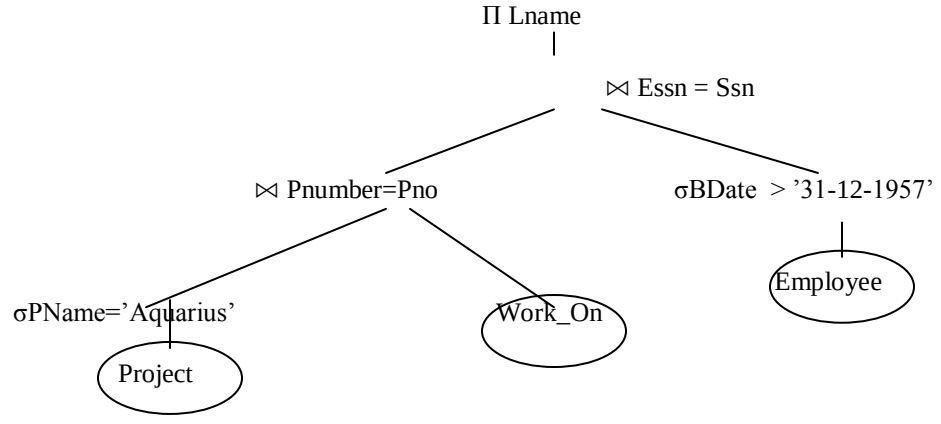
Techno.Com, Vol. 7 No. 2, Mei 2007

- **(c)** . Membatasi penggunaan operasi SELECT terlebih dahulu

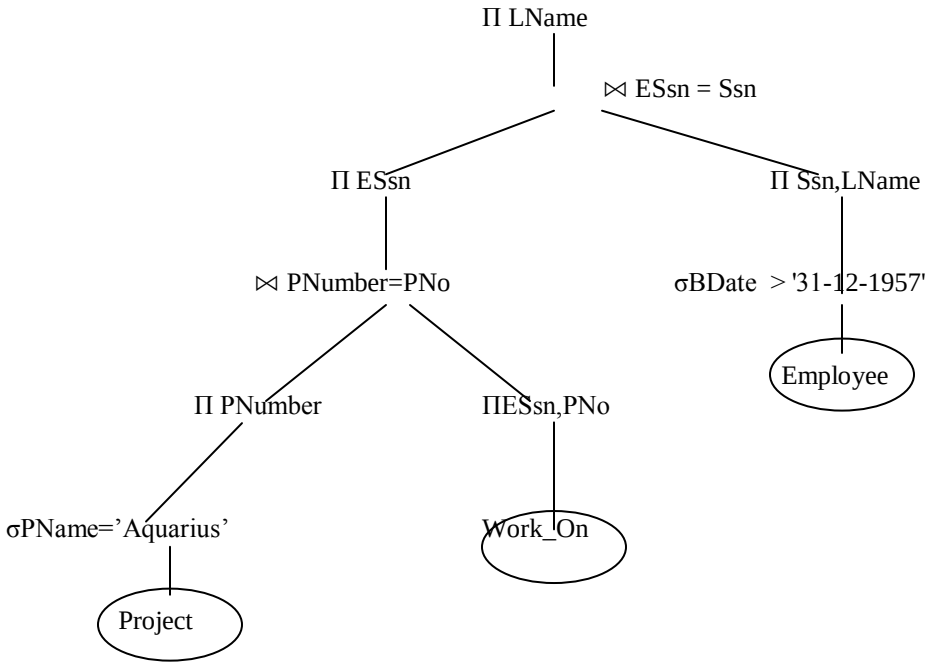


Seperti contoh diatas, sebuah query tree dapat dintransformasikan selangkah demi selangkah ke dalam query tree yang lainnya yang lebih efisien untuk dieksekusi. Bagaimanapun juga harus dipastikan terlebih dahulu bahwa langkah-langkah transformasi selalu berperan penting untuk sebuah query tree yang sama.

- **(d).** Menggantikan CARTESIAN PRODUCT dan SELECT dengan operasi –operasi JOIN



- (e). Memindahkan Operasi Project ke bawah Tree



Untuk melakukan transformasi-transformasi query tersebut, query optimizer harus mengetahui aturan-aturan transformasi mana yang mempertahankan persamaan ini. Dan aturan-aturan transformasi tersebut telah diuraikan di atas.

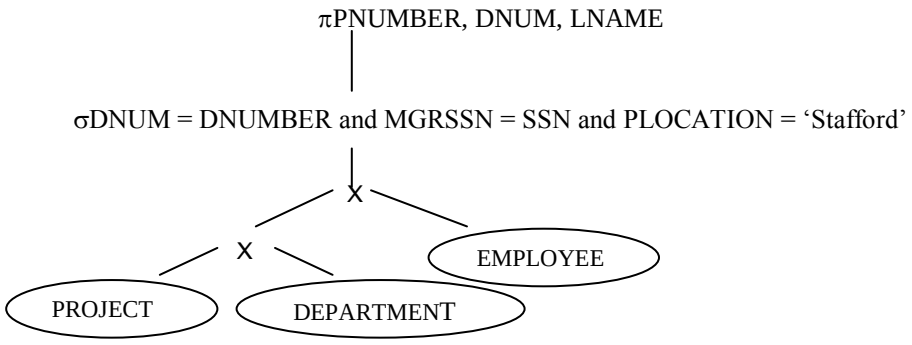
Diberikan sebuah SQL :

```
SELECT     PNUMBER, DNUM, LNAME
FROM       PROJECT, DEPARTMENT, EMPLOYEE
WHERE      DNUM = DNUMBER AND
           MGRSSN = SSN AND
           PLOCATION = "Stafford"
```

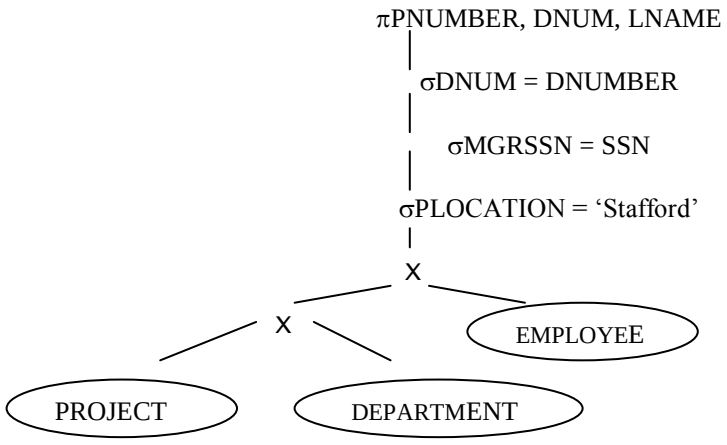
Maka Proses Optimisasi dengan Menggunakan Aturan Heuristik sbb :

Techno.Com, Vol. 7 No. 2, Mei 2007

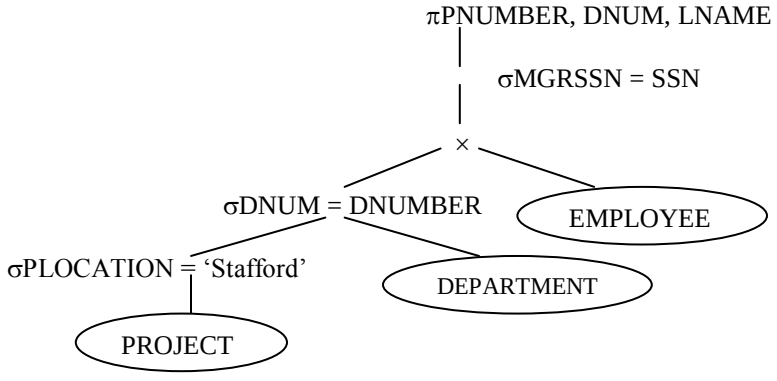
-(a). Inisial(canonical) Query Tree



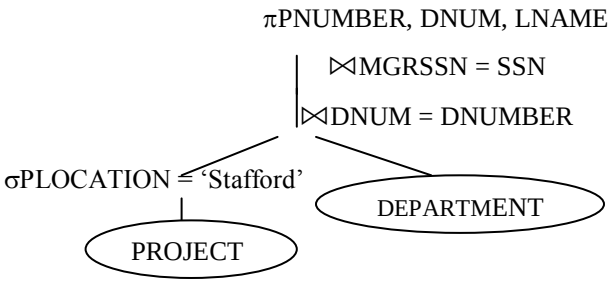
-(b). Menggunakan langkah 1 untuk memisahkan SELECT



-(c) Merubah operasi SELECT dengan Cross Product



-(d) Mengkombinasikan Cross Product dan SELECT ke dalam bentuk JOIN



KESIMPULAN

Dari hasil pengkajian mengenai proses dan teknik optimisasi query pada pembahasan diatas, maka dapat disimpulkan bahwa :

1. Optimisasi query berhubungan dengan teknik-teknik yang digunakan oleh DBMS (Database Manajemen Sistem) untuk memproses query agar diperoleh hasil query dengan waktu akses yang minimum.
2. Hasil cari optimasi dengan teknik heuristic optimization: Heuristic optimization merupakan urutan rencana-rencana dari query tunggal. Masing-masing rencana adalah lebih efisien dari rencana-rencana sebelumnya.
4. Terdapat 4 tahap yang harus dilakukan dalam optimisasi sebuah query, dimana tahapan-tahapan tersebut adalah :
   - Memasukkan query ke dalam representasi internal berdasarkan ekspresi aljabar yang sesuai.
   - Mengkonversikannya ke dalam bentuk canonical dengan cara mula-mula dengan menggunakan cartesian product dari klausa FROM, setelah itu menggabungkan dan memilih kondisi-kondisi dari klausa WHERE dan melakukan proyeksi-proyeksi dari klausa SELECT.
   - Memilih calon-calon prosedur low level, yaitu mempertimbangkan index-index atau jalan akses lainnya, membagi nilai-nilai penyimpanan data dari record-record untuk memilih satu atau lebih calon-calon prosedur untuk mengimplementasikan tiap-tiap operasi low level dalam query.
   - Menghasilkan rencana-rencana query dan memilih yang termurah, yaitu membuat sekumpulan calon rencana-rencana query dan kemudian memilih yang termurah.

DAFTAR PUSTAKA

1. Axmark, David; & Widenius, Michael. 2003. *MYSQL HELP 4.0.11.* Swedish.
2. Balling, Derek J.; & Zawodny, Jeremy. 2004. *High Performance MySQL*. California : O'Reilly Publishing.
3. England, Ken. 2001. *Microsoft SQL Server 2000 Performance Optimization and TuningHandbook*. USA: Digital Press.
4. Gulutzan, Peter; & Pelzer, Trudy. 2002. *SQL Performance Tuning*. USA: Addison Wesley.
5. Mata-toledo, Ramon A.; & Cushman, Pauline K. 2000. *Fundamentals of Relational*

6. *Databases*. : McGraw-Hill Companies.
7. Mishra, Sanjay; & Beaulieu, Alan. 2004. *Mastering Oracle SQL, 2nd Edition*. California: O'Reilly Media.
8. Silberschatz, Abraham, et. al. *Database System Concepts (Fourth Edition)*. : McGraw-Hill Companies.