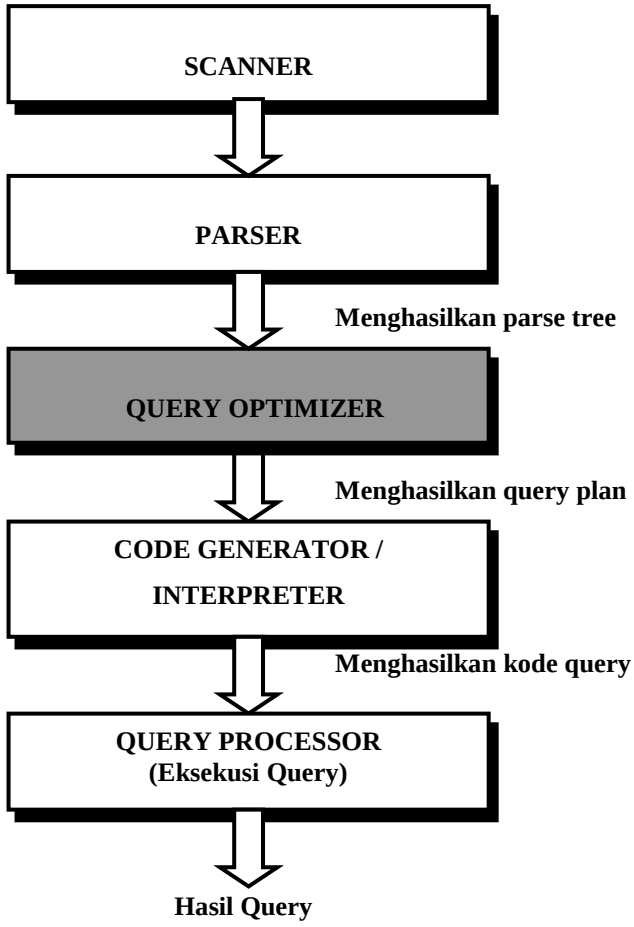
PENGENALAN PROSES DAN OPTIMISASI QUERY

Database Manajemen Sistem (DBMS) adalah kumpulan dari program-program yang membolehkan user untuk menciptakan dan memelihara sebuah database. DBMS sudah menjadi peralatan standar untuk melindungi pengguna komputer dari bagian-bagian kecil dalam pengelolaan *secondary storage (hard disk).* DBMS didesain untuk meningkatkan produktivitas dari aplikasi para programmer dan untuk memberikan kemudahan pengaksesan data oleh komputer.

Pengaksesan data di dalam DBMS dapat dilakukan dengan berbagai macam cara. Dan tentunya dalam melakukan pengaksesan data ada hal-hal yang perlu diperhatikan seperti ketepatgunaan implementasi dari data itu sendiri serta waktu prosesnya. Ada banyak *plan* (rencana) yang dapat diikuti oleh database manajemen sistem dalam memproses dan menghasilkan jawaban sebuah query. Semua plan pada akhirnya akan menghasilkan jawaban (output) yang sama tetapi pasti mempunyai harga yang berbeda-beda, seperti misalnya total waktu yang diperlukan untuk menjalankan sebuah query.

Optimisasi query mencoba memberikan suatu pemecahan untuk menangani masalah tersebut dengan cara menggabungkan sejumlah besar teknik-teknik dan strategi, yang meliputi transformasi-transformasi logika dari query-query untuk mengoptimisasi jalan akses dan penyimpanan data pada sistem file. Setelah ditransformasikan, sebuah query harus dipetakan ke dalam sebuah urut-urutan operasi untuk menghasilkan data-data yang diminta.

Query Language (SQL)



Gambar 2.1
Tahapan proses sebuah query

Sebuah query yang diekspresikan dalam sebuah bahasa query tingkat tinggi seperti SQL mula-mula harus dibaca, diuraikan dan disahkan (*scanning*, *parsing*, *validating*)[1]. Query tersebut kemudian dibentuk menjadi sebuah struktur data yang biasa disebut dengan *query tree*. Dan kemudian DBMS (Database Manajemen Sistem) harus merencanakan sebuah strategi eksekusi untuk mendapatkan kembali hasil dari query dari file-file database. Tahapan-tahapan proses dari sebuah query di

[1] http://cisnet.baruch.cuny.edu/holowczak/classes/9440/queryprocessing/

dalam sebuah sistem database ditunjukkan pada gambar 2.1. Berikut penjelasan dari masing-masing tahapan :

- *Scanner* melakukan identifikasi (pengenalan) token-token seperti *SQL keywords*, *attribute*, dan *relation name*. Proses ini disebut dengan *scanning*.
- *Query Parser* mengecek kevalidan query dan kemudian menterjemahkannya ke dalam sebuah bentuk internal yaitu ekspresi relasi aljabar atau parse tree. Proses ini disebut dengan *parsing*.
- *Query Optimizer* memeriksa semua ekspresi-ekspresi aljabar yang sama untuk query yang diberikan dan memilih salah satu dari ekspresi tersebut yang terbaik yang memiliki perkiraan termurah. Dengan kata lain, tugas dari query optimizer adalah menghasilkan sebuah rencana eksekusi. Proses ini disebut dengan *optimisasi query*.
- *Code Generator* atau *Interpreter* mentransformasikan rencana akses yang dihasilkan oleh optimizer ke dalam kode-kode. Setelah itu, kode-kode tersebut dikirimkan ke dalam query processor untuk dijalankan.
- *Query Processor* melakukan eksekusi query untuk mendapatkan hasil query yang diinginkan.

Bagian yang diarsir pada gambar 2.1 adalah merupakan komponen utama yang berperan penting dalam proses optimisasi query yang menjadi topik utama dari seluruh pembahasan ini.

Alasan dibahasnya tahapan proses query pada gambar 2.1 adalah semata-mata untuk memberikan gambaran yang jelas tentang bagaimana pada umumnya sebuah query diproses di dalam sebuah Database Manajemen Sistem (DBMS).

2.1 Pengenalan Tentang Query

Sebuah *query* adalah sebuah ekspresi bahasa yang menggambarkan data yang akan didapatkan kembali dari sebuah database. Dalam hubungannya dengan optimisasi query, seringkali diasumsikan bahwa query-query tersebut dinyatakan dalam sebuah dasar-dasar isi dan sekumpulan cara orientasi, yang memberikan optimizer pilihan-pilihan diantara alternatif prosedur-prosedur evaluasi.

Query dapat digunakan pada beberapa keadaan. Kebanyakan aplikasi nyatanya adalah permintaan-permintaan secara langsung dari *user* yang memerlukan informasi tentang bentuk maupun isi dari database. Apabila permintaan user terbatas pada sekumpulan query-query standar, maka query-query tersebut dapat dioptimisasi secara manual oleh pemrograman prosedur-prosedur pencarian gabungan dan membatasi input dari user pada sebuah ukuran menu. Tetapi bagaimanapun juga, sebuah sistem optimisasi query otomatis menjadi penting apabila query-query khusus ditanyakan dengan menggunakan bahasa query yang digunakan secara umum seperti SQL.

Aplikasi yang kedua dari query terjadi pada transaksi-transaksi yang mengubah data yang disimpan berdasarkan nilainya saat itu. Pada akhirnya, query seperti ekspresi-ekspresi dapat digunakan secara internal dalam sebuah DBMS, sebagai contoh adalah untuk mengecek kebenaran akses dan menyamakan kebenaran akses-akses yang terjadi.

Membicarakan tentang query, sangat erat hubungannya dengan cara penulisan query tersebut ke dalam sebuah bentuk bahasa yang mudah dimengerti. Pada umumnya, bahasa query yang digunakan untuk mengekspresikan sebuah pernyataan dari query adalah SQL (Structure Query Language).

SQL adalah sebuah bahasa database yang luas yang memiliki *statement-statement* (pernyataan) untuk definisi data, query dan *update* data (memperbaharui data). SQL mempunyai satu statement dasar untuk mendapatkan kembali informasi dari sebuah database. Statement dasar dari SQL adalah SELECT.

Bentuk dasar dari statement SELECT biasa disebut dengan blok *select from where* yang terbentuk dari tiga macam klausa yaitu SELECT, FROM dan WHERE yang mempunyai bentuk sebagai berikut :

```
SELECT    <daftar Attribute>
FROM      <daftar Tabel>
WHERE     <kondisi>
```

Dimana <*daftar attribute*> adalah sebuah daftar dari nama-nama attribute yang nilai-nilainya didapatkan oleh query. Sedangkan <*daftar tabel*> adalah sebuah daftar dari nama-nama relasi yang diperlukan oleh proses sebuah query. <*kondisi*> adalah sebuah kondisi ekspresi boolean yang mengidentifikasikan tuple-tuple yang akan dikembalikan oleh query.

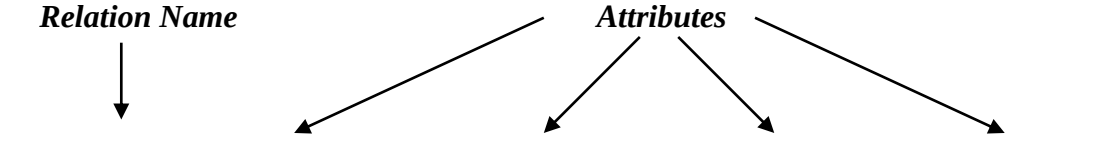
Selanjutnya, statement Select-From-Where akan selalu digunakan dalam pembahasan mengenai proses dan teknik optimisasi query nanti dengan tujuan untuk lebih memudahkan pemahaman tentang proses dan teknik optimisasi query karena statement tersebut merupakan statement dasar yang mudah dimengerti dan umum digunakan.

2.2 Hubungan Database Dengan Pemrosesan Query

Database adalah kumpulan dari data-data yang berhubungan satu sama lainnya yang digunakan untuk pencarian suatu data tertentu pada saat SQL query dijalankan. Sebuah database dirancang, dibuat dan ditempati oleh data dengan

tujuan tertentu. Di dalam sistem database relasional, tabel-tabel dari database saling berhubungan satu sama lainnya. Dan sebuah tabel database akan selalu memiliki *attribute names* (nama-nama attribute), *relation names* (nama-nama relasi), dan *tuples* (record-record).

Untuk lebih jelasnya, pada gambar 2.2 diberikan sebuah tabel database yang lengkap dengan attribute, relation name dan tuples.



STUDENT	Name	Age	Home Phone	Address
	Benyamin	19	373559	Klipang
Tuples	Barbara	23	578490	Banyumanik
	Katherine	21	871009	Purwodadi

Gambar 2.2
Attributes dan tuples dari relasi STUDENT

Attribute digunakan untuk mengidentifikasikan sebuah nama yang diikutsertakan dalam relasi dan menspesifikasikan domain (tipe data sederhana yang menentukan sebuah pemisahan data). Sedangkan *tuples* adalah kumpulan dari record-record di dalam sebuah database. *Relation name* mendefinisikan attribute-attribute yang diperlukan dalam predikat dan mendefinisikan arti dari predikat tersebut.

2.3 Sistem Katalog Dalam Database Manajemen Sistem (DBMS)

Sistem katalog adalah sebuah sistem yang digunakan untuk menyimpan informasi-informasi mengenai suatu database. Informasi yang disimpan dalam

sebuah katalog dari sebuah relasional database manajemen sistem meliputi relation names, attribute names, dan attributes domains (tipe-tipe data), gambaran dari batasan-batasan (primary key, secondary key, foreign key, NULL/NOT NULL dan tipe-tipe dari batasan-batasan lainnya), dan bentuk-bentuk penyimpanan dan index-index. Query optimizer melakukan akses pada katalog untuk jalan akses, implementasi (pelaksanaan) informasi, dan data statistik untuk menentukan jalan terbaik untuk menjalankan sebuah query.

Tabel 2.1
Contoh katalog yang menyimpan informasi tentang database

TABLE_NAME	NUM_ROWS	BLOCK
PROJECT	2000	100
DEPARTMENT	50	5
EMPLOYEE	10000	2000

Sebagai contoh, optimizer mengakses katalog untuk mengecek field-field mana dari sebuah relasi yang memiliki akses hash atau akses index-index, sebelum memutuskan bagaimana untuk menjalankan sebuah kondisi pilihan ataupun kondisi join pada relasi. Contoh dari sebuah katalog yang menyimpan informasi tentang database dapat dilihat pada tabel 2.1

2.4 Optimisasi Query

Optimisasi Query adalah suatu proses untuk menganalisa query untuk menentukan sumber-sumber apa saja yang digunakan oleh query tersebut dan apakah penggunaan dari sumber tersebut dapat dikurangi tanpa merubah output. Atau bisa juga dikatakan bahwa optimisasi query adalah sebuah prosedur untuk meningkatkan strategi evaluasi dari suatu query untuk membuat evaluasi tersebut menjadi lebih efektif. Optimisasi query mencakup beberapa teknik seperti

transformasi query ke dalam bentuk logika yang sama, memilih jalan akses yang optimal dan mengoptimumkan penyimpanan data.

Optimisasi query merupakan bagian dasar dari sebuah sistem database dan juga merupakan suatu proses untuk menghasilkan rencana akses yang efisien dari sebuah query di dalam sebuah database. Secara tidak langsung, sebuah rencana akses merupakan sebuah strategi yang nantinya akan dijalankan untuk sebuah query, untuk mendapatkan kembali operasi-operasi yang apabila dijalankan akan menghasilkan database record query. Ada tiga aspek dasar yang ditetapkan dan mempengaruhi optimisasi query, yaitu : *search space, cost model* dan *search strategy*.

Search space adalah sekumpulan rencana-rencana akses yang sama secara logika yang dapat digunakan untuk mengevaluasi sebuah query. Semua rencana-rencana dalam search space query mengembalikan hasil yang sama biarpun beberapa rencana lebih efisien dibandingkan dengan rencana yang lainnya.

Cost model menandakan sebuah harga untuk tiap rencana dalam search space. Harga dari rencana tersebut adalah sebuah perkiraan dari sumber-sumber yang digunakan pada saat rencana dijalankan, dimana harga yang lebih rendah, merupakan yang terbaik dari rencana-rencana yang ada.

Search strategy adalah sebuah perincian dari rencana-rencana mana dalam search space yang akan diperiksa. Apabila search space-nya kecil, maka strategi yang dapat diteruskan adalah menghitung dan mengevaluasi setiap rencana. Meskipun kebanyakan search space bahkan untuk query-query yang sederhana adalah sangat besar, akan tetapi query optimizer selalu memerlukan aturan heuristik untuk mengontrol nomer dari rencana-rencana yang akan diperiksa.

2.4.1 Sejarah Singkat Optimisasi Query

Optimisasi query lahir sejak sebelum tahun 1970. Pada saat itu yang dikenal dengan jaman kegelapan (dark age), optimisasi query masih dilakukan secara manual oleh manusia. Selain itu, pada jaman tersebut database relasional juga belum dikenal sehingga diperlukan seseorang yang benar-benar ahli dalam database untuk melakukan optimisasi query. Jadi pada jaman kegelapan hingga tahun 1970-an, optimisasi query masih dilakukan oleh manusia dan masih menggunakan cara yang benar-benar kuno.

Seiring dengan perkembangan jaman di mana teknologi menjadi semakin maju, maka sekitar pertengahan tahun 1970-an sampai pada pertengahan tahun 1980-an optimisasi query tidak lagi dilakukan secara manual, tetapi dilakukan oleh suatu sistem yang dikenal dengan *sistem R* yang kemudian menjadi berkembang pada optimisasi perintah JOIN, sekumpulan tahapan untuk optimisasi query selanjutnya. Hingga saat ini, optimisasi query terus berkembang bersamaan dengan bermunculannya teknik-teknik baru yang digunakan dalam proses optimisasi query meskipun pada dasarnya hanya ada dua macam teknik utama yang biasanya digunakan dalam mengoptimisasi sebuah query. Bisa dikatakan bahwa optimisasi query merupakan tulang punggung dari sebuah sistem karena ketepatgunaan suatu sistem sangat tergantung dari optimizernya.

Hampir semua DBMS menggunakan sistem optimisasi query untuk mengurangi waktu eksekusi query sehingga kerja sistem dapat dioptimalkan dan penggunaan sumber-sumber dapat diminimumkan. Selain itu, akes dari disk yang sangat lambat dibandingkan dengan akses memory membuat optimisasi query menjadi semakin penting.

2.4.2 Tujuan Optimisasi Query

Prinsip ekonomi yang diperlukan untuk sebuah query adalah mengoptimisasi prosedur-prosedur, mencoba untuk memaksimumkan output dari sejumlah sumber-sumber yang diberikan ataupun untuk meminimumkan penggunaan sumber untuk memberikan output.

Tujuan dari optimisasi query adalah berbeda-beda untuk setiap sistem. Ada yang menggunakan optimisasi query untuk meminimumkan waktu proses sedangkan pada situasi lain bisa juga optimisasi query diperlukan untuk waktu respon, meminimumkan I/O dan meminimumkan penggunaan memory. Tetapi pada dasarnya, tujuan dari optimisasi query adalah menemukan jalan akses yang termurah untuk meminimumkan total waktu pada saat proses sebuah query. Untuk mencapai tujuan tersebut, maka diperlukan *optimizer* untuk melakukan analisa query dan untuk melakukan pencarian jalan akses.

2.5 Operasi-operasi Dasar Aljabar Relasional

Untuk menetapkan bentuk dan batasan-batasan database, sebuah model data harus memasukkan sekumpulan operasi-operasi untuk mengontrol data. Sekumpulan model operasi-operasi relasional standar merupakan aljabar relasional. Operasi-operasi ini membolehkan user untuk menentukan dasar pencarian permintaan. Hasil dari sebuah pencarian adalah relasi yang baru, di mana mungkin saja dibentuk dari satu atau lebih relasi. Oleh sebab itu, operasi-operasi aljabar menghasilkan relasi-relasi baru yang dapat menjadi lebih berguna dengan menggunakan operasi-operasi aljabar yang sama. Urutan dari operasi-operasi aljabar relasional membentuk sebuah ekspresi relasi aljabar yang hasilnya juga berupa sebuah relasi.

Operasi-operasi relasi aljabar umumnya dibagi ke dalam dua kelompok. Kelompok pertama, termasuk sekumpulan operasi-operasi dari sekumpulan teori matematika yang dapat dipakai karena tiap-tiap relasi ditetapkan menjadi sekumpulan tuple-tuple. Sekumpulan operasi-operasi tersebut termasuk UNION, INTERSECTION, SET DIFFERENCE dan CARTESIAN PRODUCT.

Kelompok yang kedua terdiri dari operasi-operasi yang khusus dibuat untuk database-database relasional. Yang termasuk dalam kelompok yang kedua adalah SELECT, PROJECT dan JOIN.

2.5.1 Operasi SELECT

Operasi SELECT digunakan untuk memilih sebuah subset tuple-tuple dari sebuah relasi yang memenuhi pilihan. Pada umumnya, operasi SELECT ditunjukkan oleh :

σ<kondisi pilihan> (R)

dimana **"σ"** (sigma) digunakan untuk menetapkan operator SELECT dan <kondisi pilihan> adalah berupa ekspresi boolean yang ditetapkan pada attribute-attribute dari relasi R. Perlu diingat bahwa R pada umumnya adalah sebuah ekspresi aljabar relasional yang hasilnya adalah sebuah relasi. Ekspresi yang paling sederhana dari operasi SELECT adalah sebuah nama dari sebuah relasi database. Relasi yang dihasilkan dari operasi SELECT mempunyai attribute yang sama, sebagai R. Ekspresi boolean yang ditetapkan dalam <kondisi pilihan> terbentuk dari sejumlah klausa-klausa dengan bentuk sebagai berikut :

<nama attribute><operator pembanding><nilai konstan>, atau

<nama attribute><operator pembanding><nama attribute>

dimana <nama attribute> adalah nama sebuah attribute dari R, <operator pembanding> biasanya adalah salah satu dari operator-operator seperti "=","<",">=", "≤" dan "≠", dan <nilai konstan> adalah sebuah nilai konstan dari attribute domain. Klausa-klausa dapat mempunyai hubungan yang berbeda-beda dengan operator-operator boolean AND, OR, dan NOT untuk membentuk kondisi pilihan yang umum.

Sebagai contoh, dengan menggunakan tabel-tabel database pada lampiran A, untuk memilih tuple-tuple dari semua employee yang juga bekerja pada department 4 dan menghasilkan salary lebih dari $25.000 per-tahun, atau bekerja pada departement 5 dan menghasilkan $30.000 per-tahun, maka operasi SELECT yang dapat ditetapkan adalah :

$$\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ (DNO=5\ AND\ SALARY>30000)}(EMPLOYEE)$$

Umumnya, hasil dari sebuah operasi SELECT dapat ditentukan sebagai berikut. <selection condition> digunakan secara bebas untuk tiap-tiap tuple T dalam R. Hal ini dilakukan dengan mengganti tiap-tiap kejadian dari sebuah attribute A, dalam kondisi pilihan dengan nilainya pada tuple $t[A_i]$. Apabila kondisi dinilai benar, maka tuple t akan dipilih. Semua tuple-tuple yang dipilih akan tampak pada hasil dari operasi SELECT.

Operator SELECT adalah *unary*. Maka dari itu, operator SELECT digunakan untuk sebuah relasi tunggal. Lebih dari itu, operasi pemilihan digunakan untuk tiap-tiap tuple secara individu. Bahkan, kondisi-kondisi pilihan tidak dapat menggunakan lebih dari satu tuple. Jumlah tuple-tuple dari relasi yang dihasilkan selalu lebih kecil atau sama dengan jumlah tuple pada R. Maka dari itu, $|\sigma_c (R)| \leq |R|$ untuk setiap kondisi C. Bagian dari tuple-tuple yang dipilih oleh sebuah kondisi pilihan disebut sebagai **selectivity** dari kondisi.

2.5.2 Operasi PROJECT

Operasi PROJECT digunakan untuk memilih kolom-kolom tertentu dari tabel dan membuang kolom-kolom lainnya yang tidak diperlukan. Pada umumnya operasi PROJECT ditunjukkan oleh :

π<daftar attribute> (R)

di mana **"π"** (pi) adalah simbol yang digunakan untuk menggambarkan operasi PROJECT dan <daftar attribute> adalah sebuah daftar attribute dari attribute-attribute pada relasi R. Perlu diingat bahwa R umumnya adalah ekspresi aljabar relasional yang hasilnya adalah relasi, dimana dalam kasus yang paling sederhana R adalah nama dari sebuah relasi database. Hasil dari operasi PROJECT hanyalah attribute-attribute yang ditentukan dalam <attribute list> dan dalam urutan yang sama seperti yang terlihat pada list.

2.5.3 Operasi JOIN

Operasi JOIN dilambangkan dengan **"⋈"** dan digunakan untuk mengkombinasikan hubungan tuple-tuple dari dua relasi kedalam tuple tunggal. Pada umumnya operasi PROJECT pada dua relasi $R(A_1,A_2,\ldots A_n)$ dan $S(B_1,B_2,\ldots B_m)$ ditunjukkan oleh :

R⋈ <kondisi join> (S)

Hasil dari JOIN adalah sebuah relasi Q dengan n + m attribute $Q(A_1,A_2,\ldots A_n, B_1,B_2,\ldots B_m)$. Q mempunyai satu tuple untuk masing-masing kombinasi dari tuple, satu dari R dan satu dari S. Dalam JOIN, hanya kombinasi-kombinasi dari tuple-tuple yang memenuhi kondisi join yang akan tampak pada hasil. Kondisi JOIN ditentukan oleh attribute-attribute dari relasi R dan S dan evaluasi untuk tiap kombinasi dari tuple-tuple.

Bentuk dari kondisi JOIN secara umum adalah :

<condition> AND <condition> AND … AND<condition>

dimana tiap kondisi adalah bentuk dari $A_i\ \theta\ B_j$. A_i adalah sebuah attribute dari R dan B_j adalah sebuah attribute dari S. A_i dan B_j mempunyai domain yang sama, dan θ (theta) adalah salah satu dari operator-operator pembanding $\{<, \leq, \neq, \geq, >, =\}$. Operasi JOIN dengan sebuah kondisi join yang umum disebut dengan *theta join*.

2.5.4 Sekumpulan Operasi

Beberapa kumpulan operasi digunakan untuk menggabungkan elemen-elemen dari dua kelompok dalam cara yang bervariasi, termasuk UNION, INTERSECTION dan SET DIFFERENCE. Operasi-operasi ini adalah operasi-operasi binary yang masing-masing digunakan untuk dua kumpulan relasi.

Berikut definisi dari UNION, INTERSECTION, dan SET DIFFERENCE pada dua relasi R dan S .

- UNION : Hasil dari operasi ini ditunjukkan oleh $R \cup S$, yaitu sebuah relasi yang merupakan semua tuple-tuple yang ada alam R atau S atau keduanya. Tuple duplikat dihilangkan.
- INTERSECTION : Hasil dari operasi ini ditunjukkan oleh $R \cap S$, yaitu sebuah relasi yang merupakan semua tuple yang ada dalam R dan S.
- SET DIFFERENCE : Hasil dari operasi ini ditunjukkan oleh $R - S$, yaitu sebuah relasi yang merupakan semua tuple-tuple yang ada dalam R tetapi tidak ada dalam S.

Operasi CARTESIAN PRODUCT yang juga biasa disebut dengan CROSS PRODUCT atau CROSS JOIN dilambangkan dengan "X" yang juga merupakan sebuah kumpulan operasi binary. Operasi ini digunakan untuk mengkombinasikan tuple-tuple dari dua relasi dalam sebuah model gabungan. Umumnya, hasil dari $R(A_1, A_2,\ldots, A_n)$ X $S(B_1, B_2, \ldots, B_m)$ adalah relasi Q dengan n+m dari attribute-attribute $R(A_1, A_2,\ldots, A_n, B_1, B_2, \ldots, B_m)$ dalam urutannya. Hasil dari relasi Q mempunyai satu tuple untuk masing-masing kondisi dari tuple-tuple. Satu dari R dan satu dari S.

2.6 Algoritma-algoritma untuk Menjalankan Operasi-operasi Query

Macam-macam operasi query seperti Select, Join, Project dan *set operation* (sekumpulan operasi) dapat diimplementasikan dengan menggunakan satu atau lebih *access routines* yang berbeda. *Access Routines* adalah algoritma-algortima yang digunakan untuk mengakses dan mengelompokkan data di dalam sebuah database.

Berikut ini akan dijelaskan mengenai algortima-algoritma yang digunakan untuk menjalankan operasi-operasi query di dalam sebuah sistem database. Algoritma-algoritma yang akan dijelaskan berikut ini antara lain adalah algoritma algoritma untuk mengimplementasikan operasi SELECT, algoritma-algoritma untuk mengimlementasikan operasi JOIN, algoritma untuk mengimplementasikan operasi PROJECT dan algoritma untuk mengimplementasikan *Sekumpulan operasi*.

2.6.1 Algoritma untuk Mengimplementasikan Operasi SELECT

Ada bermacam-macam metode untuk mengimplementasikan operasi SELECT. Dan pengimplementasian operasi select tergantung pada keberadaannya dan tipe pengindeks-annya serta kombinasi *conjunctive* dan *disjunctive* dari pencarian kriteria.

Pengimplementasian dari operasi SELECT dibagi lagi menjadi dua, yaitu implementasi untuk *simple selection* (pilihan sederhana) dan implementasi untuk *complex selection* (pilihan kompleks). Berikut ini akan dibahas mengenai algoritma untuk implentasi kedua pilihan tersebut.

2.6.1.1 Metode untuk Simple Selection

Sejumlah algoritma *search* memungkinkan untuk menyeleksi record-record dari sebuah file. Algoritma-algoritma search juga dikenal sebagai *file scan* karena algoritma tersebut melakukan pembacaan record dari sebuah file untuk mencari dan mendapatkan kembali record-record yang memenuhi sebuah kondisi seleksi. Apabila algoritma search melibatkan penggunaan *index*, maka pencarian index disebut dengan *index scan*.

Berikut ini akan diberikan beberapa contoh dari *simple selection* yang data-datanya diambil dari file-file pada skema database COMPANY yang dapat dilihat pada lampiran A.

Contoh 1 : Salary > 28000 .[2]

Contoh 2 : SSN = ‘123456789’

Contoh 3 : DNO = 5

---

[2] Disebut dengan batasan query

Contoh 1 dan contoh 3 diambil dari file EMPLOYEE. Sedangkan contoh 3 diambil dari file DEPARTMENT.

Simple selection dari contoh 1, contoh 2 dan contoh 3 dapat diimplementasikan dengan menggunakan metode-metode :

- ***Linear Search (brute force)***, untuk mendapatkan kembali setiap record dalam file dan menguji apakah nilai-nilai attribute dari record tersebut memenuhi kondisi pilihan.
- ***Binary Search***, apabila kondisi pilihan melibatkan sebuah perbandingan persamaan pada sebuah *key attribute* pada file yang berurutan. Maka binary search yang lebih efisien daripada linear search dapat digunakan. Contohnya adalah SSN =’123456789’, apabila SSN adalah permintaan attribute untuk file EMPLOYEE.
- ***Primary Index atau Hash Key***, apabila kondisi pilihan melibatkan sebuah perbandingan persamaan pada sebuah key attribute dengan primary index (hash key). Sebagai contoh, SSN = ‘123456789’ menggunakan primary index untuk mendapatkan kembali record. Perlu diketahui bahwa kondisi ini kebanyakan mengembalikan sebuah record tunggal.
- ***Menggunakan primary index untuk mendapatkan kembali multiple record***, apabila perbandingan kondisi adalah >, >=, <, atau <= pada sebuah *key field* dengan sebuah primary index. Sebagai contoh, DNUMBER > 5 menggunakan index untuk menemukan record yang memenuhi persamaan kondisi yang cocok (DNUMBER = 5), kemudian mengembalikan semua record-record berikutnya pada urutan file. Untuk kondisi DNUMBER < 5, mengembalikan semua record-record terdahulu.

- ***Menggunakan clustering index untuk mendapatkan kembali multiple record***, apabila kondisi pilihan melibatkan sebuah perbandingan persamaan pada non-key attribute dengan sebuah clustering index. Sebagai contoh, DNO = 5 untuk file EMPLOYEE menggunakan index untuk mendapatkan kembali semua record yang memenuhi kondisi.
- ***Secondary (B^+-tree) index pada sebuah perbandingan persamaan***, untuk mendapatkan kembali sebuah record tunggal jika pengindeks-an field adalah sebuah *key* atau untuk mendapatkan kembali multiple record jika pengindeks-an field bukan sebuah key dan juga bisa digunakan untuk perbandingan-perbandingan yang meliputi >, >=, <, atau <=.

2.6.1.2 Metode untuk Complex Selection

Apabila sebuah kondisi dari operasi SELECT adalah sebuah kondisi konjungtif, yaitu terdiri dari beberapa kondisi-kondisi sederhana yang berhubungan dengan logika AND.

Berikut ini akan diberikan beberapa contoh dari *complex selection* yang data-datanya diambil dari file-file pada skema database COMPANY yang dapat dilihat pada lampiran A.

Contoh 4 : DNO = 5 AND SALARY > 30000 AND SEX = ‘F’

Contoh 5 : ESSN = ‘123456789’ AND PNO = 10

Contoh 4 diambil dari file EMPLOYEE. Sedangkan contoh 5 diambil dari file WORKS_ON.

Complex selection dari contoh 4 dan contoh 5 dapat diimplementasikan dengan menggunakan metode-metode :

- ***Conjunctive selection***. Menggunakan salah satu dari metode-metode akses dari simple selection untuk mendapatkan kembali record-record dari kriteria pertama yang cocok, kemudian menggunakan sekumpulan hasil untuk melakukan pengecekan dari sisa kriteria.
- ***Composite Index***. Apabila dua atau lebih attribute diperlukan dalam kondisi- kondisi persamaan pada kondisi konjungtif dan sebuah index gabungan tersedia pada field-field kombinasi. Sebagai contoh, jika sebuah index sudah dibentuk pada key gabungan (ESSN, PNO) dari file WORKS_ON untuk ESSN ='123456789' AND PNO = 10, maka dapat digunakan index secara langsung.
- ***Intersection Record Pointers***. Apabila indeks-indeks secondary (jalan akses lainnya) tersedia pada lebih dari satu field yang diperlukan dalam kondisi-kondisi sederhana dalam kondisi konjungtif dan apabila indeks-indeks termasuk *pointer record* [3], kemudian tiap-tiap indeks dapat digunakan untuk mendapatkan kembali kumpulan pointer record yang memenuhi kondisi masing-masing. *Intersection* dari kumpulan-kumpulan pointer record ini memberikan pointer-pointer record yang memenuhi kondisi konjungtif, yang kemudian digunakan untuk mendapatkan kembali record-record tersebut secara langsung.

Apabila sebuah kondisi tunggal menentukan pilihan seperti contoh 1, contoh 2 dan contoh 3, maka yang dilakukan hanyalah mengecek apakah sebuah jalan akses tersedia pada attribute yang termasuk dalam kondisi tersebut. Apabila sebuah jalan akses tersedia, maka metode yang cocok untuk jalan akses tersebut digunakan.

---

[3] Sebuah pointer record secara khusus mengidentifikasikan sebuah record dan memberikan alamat record pada disk. Oleh sebab itu pointer record disebut juga dengan record identifier atau record id.

Sebaliknya, pendekatan metode brute force linear search dapat digunakan. Optimisasi query untuk sebuah operasi SELECT kebanyakan diperlukan untuk kondisi-kondisi pilihan konjungtif apabila lebih dari satu attribute terlibat dalam kondisi-kondisi yang mempunyai sebuah jalan akses. Optimizer harus memilih jalan akses yang mendapatkan record-record paling sedikit dengan cara yang paling efisien dengan memperkirakan harga-harga yang berbeda dan memilih metode dengan perkiraan harga yang paling sedikit.

Pada saat optimizer memilih antara kondisi-kondisi multiple yang sederhana dalam sebuah kondisi pilihan konjungtif, maka optimizer secara khusus mempertimbangkan *selectivity* dari masing-masing kondisi. ***Selectivity* (s)** didefinisikan sebagai perbandingan dari nomer record-record (tuple-tuple) yang memenuhi kondisi untuk jumlah nomer dari record-record pada file (relasi). Yaitu sebuah nomer antara 0 (nol) dan 1 (satu), yang berarti selectivity 0 (nol) berarti tidak ada record yang memenuhi kondisi dan 1 (satu) berarti semua record memenuhi kondisi. Meskipun selectivity-selectivity yang tepat dari semua kondisi-kondisi mungkin tidak tersedia, perkiraan dari selectivity-selectivity tersebut seringkali disimpan dalam katalog DBMS dan digunakan oleh optimizer. Sebagai contoh, untuk sebuah persamaan kondisi pada key attribute dari relasi r(R), s= 1/ |r(R)|, di mana |r(R)|adalah nomer dari tuple-tuple dalam relasi r(R). Untuk sebuah kondisi persamaan pada sebuah attribute dengan *i distinct values*, s diperkirakan oleh (|r(R)|/ *i*) /|r(R)| atau 1/*i* dengan anggapan bahwa record-record dibagi diantara distinct value. Menurut anggapan tersebut, record-record |r(R)|/ *i* akan memenuhi sebuah kondisi persamaan pada attribute ini. Umumnya, nomer dari record-record yang memenuhi sebuah kondisi pilihan dengan selectivity s akan diperkirakan menjadi |r(R)| * s. Perkiraan yang lebih kecil ini adalah memerlukan

penggunaan kondisi pertama yang lebih tinggi untuk mendapatkan kembali record-record.

Perbandingan untuk sebuah kondisi pilihan konjungtif adalah sebuah kondisi *disjunctive* (dimana kondisi-kondisi sederhana dihubungkan oleh logika OR lebih daripada AND) yang lebih sulit untuk diproses dan dioptimisasi. Sebagai contoh adalah :

σDNO = 5 OR SALARY > 30000 OR SEX = 'F'

Dengan sebuah kondisi yang demikian, optimisasi yang sederhana dapat dilakukan, karena record-record yang memenuhi kondisi disjungtif adalah *union* (penggabungan) dari record-record yang memenuhi kondisi-kondisi individual. Malahan, apabila terdapat salah satu dari kondisi tersebut tidak mempunyai sebuah jalan akses, maka terpaksa menggunakan pendekatan brute force linear search. Hanya jika sebuah jalan ada pada setiap kondisi pilihan yang dapat dioptimisasi dengan mendapatkan kembali record-record yang memenuhi tiap-tiap kondisi atau record id-nya, dan kemudian menentukan operasi union untuk mengeleminasi kondisi-kondisi yang sama.

Sebuah DBMS menyediakan beberapa metode yang sudah didiskusikan sebelumnya dan khususnya beberapa metode-metode tambahan. Query optimizer harus memilih salah satu metode yang tepat untuk mengeksekusi tiap-tiap operasi SELECT dalam sebuah query. Optimisasi ini menggunakan rumus-rumus untuk memperkirakan harga-harga untuk tiap-tiap metode akses yang tersedia Pada akhirnya, optimizer akan memilih metode akses dengan perkiraan harga terendah.

2.6.2 Algoritma untuk Mengimplementasikan Operasi JOIN

Operasi JOIN merupakan salah satu dari kebanyakan operasi-operasi lainnya yang memakan waktu dalam melakukan pemrosesan query. Algoritma–algoritma yang dipertimbangkan untuk bentuk operasi JOIN adalah $R \bowtie_{A=B} S$ di mana A dan B adalah attribute-attribute domain yang cocok dari R dan S, secara berturut-turut. Sejumlah metode yang tersedia untuk mengimplementasikan operasi JOIN adalah sebagai berikut :

- ***Nested Loop (brute force)***. Untuk setiap record t pada R (outer loop), dapatkan kembali record s dari S (inner loop) dan lakukan tes untuk dua record yang memenuhi kondisi JOIN t[A] = s[B]. [4]
- ***Access Structures (Indexes)*** **atau** ***single-loop join***. Jika sebuah indeks (hash key) tersedia untuk satu atau dua attribute join (B dari S), dapatkan kembali masing-masing record t dalam R, sekali pada satu saat (single loop) dan kemudian dengan menggunakan akses untuk mendapatkan kembali semua record-record s dari S yang memenuhi s[B] = t[A].
- ***Sort-Merge Join***. Jika record-record dari R dan S diurutkan secara fisik oleh nilai dari attribute-attribute join dari A dan B berturut-turut, maka join dapat diimplementasikan dalam banyak cara yang memungkinkan dan efisien. Kedua file tersebut dibaca secara bersamaan dalam perintah join attribute, mencocokkan record-record yang mempunyai nilai-nilai yang sama untuk A dan B. Apabila file-file tersebut belum urut, maka akan diurutkan terlebih dahulu dengan menggunakan *eksternal sorting*.[5]

---

[4] Untuk file-file disk, jelas bahwa pengulangan akan memenuhi blok-blok disk. Jadi teknik ini dapat juga disebut dengan nested-block join.

[5] Merupakan algoritma sorting yang cocok untuk file-file yang besar dari penyimpanan record pada disk.

- ***Hash Join***. Record-record dari file-file R dan S, keduanya digabungkan pada file hash yang sama, menggunakan fungsi-fungsi hash yang sama pada attribute-attribute join A dari R dan B dari S sebagai hash key. Mula-mula, sebuah single pass melalui file yang dengan record yang lebih sedikit (dimisalkan dengan R) menggabungkan record-recordnya ke dalam file hash selama record-record R dibagi ke dalam bucket-bucket hash. Pada tahap yang kedua, sebuah single pass melalui file lainnya (S) kemudian menggabungkan tiap-tiap record-nya untuk memeriksa bucket yang cocok, dan record tersebut dikombinasikan dengan semua record-record yang cocock dari R dalam bucket tersebut. Penyederhanaan ini menggambarkan bahwa ukuran yang lebih kecil dari dua file dimasukkan ke dalam bucket-bucket memory secara tepat setelah melalui tahap pertama.

2.6.3 Algoritma untuk Mengimplementasikan Operasi PROJECT dan Sekumpulan Operasi

Sebuah operasi PROJECT $\pi_{<\text{attribute list}>}(R)$ akan dilaksanakan apabila <attribute list> termasuk sebuah key dari relasi R, karena dalam kasus ini hasil dari operasi PROJECT akan mempunyai nomer dari tuple-tuple yang sama sebagai R, tetapi hanya dengan nilai-nilai untuk attribute-attribute dalam <attribute list> pada tiap tuple. Apabila <attribute list> tidak termasuk sebuah key dari R, maka tuple-tuple yang sama harus dieliminasi. Pengeliminasian ini biasanya dilakukan dengan melakukan sorting pada hasil operasi dan kemudian mengeliminasi tuple-tuple yang sama, yang tampak secara berurutan setelah sorting. Hashing juga dapat digunakan untuk mengeliminasi record-record yang sama dengan cara masing-masing record dibagi dan diselipkan ke dalam sebuah bucket pada hash file dalam memory. Dan

kemudian dilakukan pengecekan kembali untuk memastikan apakah record-record tersebut sudah berada di dalam bucket. Apabila record yang dimasukkan tersebut merupakan record yang sama, maka record tersebut tidak diselipkan ke dalam bucket.

Sekumpulan operasi seperti UNION, INTERSECTION, SET DIFFERENCE dan CARTESIAN PRODUCT kadang-kadang sulit untuk dilakukan.Umumnya, operasi cartesian product R X S lebih sulit untuk dilakukan karena hasil dari operasi ini adalah sebuah record untuk tiap-tiap kombinasi dari record-record R dan S. Apabila R mempunyai n record dan j attribute dan S mempunyai m record dan k attribute, maka hasil dari relasi kedua record tersebut adalah n * m record dan j + k attribute. Oleh sebab itu, operasi CARTESIAN PRODUCT harus dihindari dan digantikan dengan operasi-operasi lainnya yang sama selama optimisasi query yang akan dibahas pada sub bab 3.3.1.

Tiga kumpulan operasi-operasi lainnya seperti UNION, INTERSECTION, dan SET DIFFERENCE hanya dipakai untuk relasi-relasi union yang cocok, yang mempunyai nomer attribute yang sama dan domain-domain attribute yang sama. Cara yang biasanya digunakan untuk melaksanakan operasi-operasi ini adalah dengan menggunakan variasi-variasi dari teknik *sort-merge*, yaitu dua relasi diurutkan pada attribute yang sama dan setelah diurutkan dilakukan sekali pembacaan pada tiap-tiap relasi yang memenuhi untuk menghasilkan hasilnya. Sebagai contoh, operasi UNION dapat dilaksanakan (R ∪ S) , dengan *scanning* (pembacaan) dan *merging* (penggabungan) kedua file-file yang sudah urut secara bersamaan dan apabila tuple yang sama tersedia dalam kedua relasi, maka dipilih hanya satu tuple untuk disimpan dalam hasil penggabungan. Untuk operasi

INTERSECTION, $R \cap S$, maka akan disimpan pada hasil gabungan hanya untuk record-record yang tampak pada kedua relasi.

Hasing juga dapat digunakan untuk melaksanakan UNION, INTERSECTION, dan SET DIFFERENCE. Satu tabel dibagi dan yang lainnya digunakan untuk memeriksa partisi yang tepat. Sebagai contoh, untuk melaksanakan $R \cup S$, maka pertama-tama partisi record-record dari R untuk hash file. Kemudian bagi tiap-tiap record S, dilakukan pemeriksaan untuk mengecek apabila sebuah record yang sama dari R ditemukan dalam bucket.