PERANGKAT BANTU UNTUK OPTIMASI QUERY PADA ORACLE DENGAN RESTRUKTURISASI SQL

Darlis Heru Murti - Firman Azizi – Esther Hanaya
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember,
Email: darlis@its-sby.edu, firman01@inf.its-sby.edu

ABSTRAK

Query merupakan bagian dari bahasa pemrograman SQL (Structured Query Language) yang berfungsi untuk mengambil data (read) dalam DBMS (Database Management System), termasuk Oracle [3]. Pada Oracle, ada tiga tahap proses yang dilakukan dalam pengeksekusian query, yaitu Parsing, Execute dan Fetch. Sebelum proses execute dijalankan, Oracle terlebih dahulu membuat execution plan yang akan menjadi skenario dalam proses excute.

Dalam proses pengeksekusian query, terdapat faktor-faktor yang mempengaruhi kinerja query, di antaranya access path (cara pengambilan data dari sebuah tabel) dan operasi join (cara menggabungkan data dari dua tabel). Untuk mendapatkan query dengan kinerja optimal, maka diperlukan pertimbangan-pertimbangan dalam menyikapi faktor-faktor tersebut.

Optimasi query merupakan suatu cara untuk mendapatkan query dengan kinerja seoptimal mungkin, terutama dilihat dari sudut pandang waktu. Ada banyak metode untuk mengoptimasi query, tapi pada Penelitian ini, penulis membuat sebuah aplikasi untuk mengoptimasi query dengan metode restrukturisasi SQL statement. Pada metode ini, objek yang dianalisa adalah struktur klausa yang membangun sebuah query.

Aplikasi ini memiliki satu input dan lima jenis output. Input dari aplikasi ini adalah sebuah query sedangkan kelima jenis output aplikasi ini adalah berupa query hasil optimasi, saran perbaikan, saran pembuatan indeks baru, execution plan dan data statistik.

Cara kerja aplikasi ini dibagi menjadi empat tahap yaitu mengurai query menjadi sub query, mengurai query per-klausa, menentukan access path dan operasi join dan restrukturisasi query.Dari serangkaian ujicoba yang dilakukan penulis, aplikasi telah dapat berjalan sesuai dengan tujuan pembuatan Penelitian ini, yaitu mendapatkan query dengan kinerja optimal.

***Kata Kunci** : Query, SQL, DBMS, Oracle, Parsing, Execute, Fetch, Execution Plan, Access Path, Operasi Join, Restrukturisasi SQL statement.*

1. PENDAHULUAN

Salah satu tujuan utama pembuatan software Sistem Informasi adalah untuk mengolah data dengan harapan data tersebut dapat memberikan suatu informasi pada saat diperlukan dengan tepat dan cepat.

Faktor waktu merupakan salah satu aspek penting dalam penilaian baik-buruknya kinerja sebuah *software* Sistem Informasi. Oleh karena itu diperlukan manajemen data yang baik, karena dengan manajemen data yang baik akan mempermudah proses pencarian dan pengambilan data, sehingga akan berdampak pada proses komputasi yang dilakukan oleh komputer. Manajemen data sangat erat kaitannya dengan database yang dirancang, dan untuk berinteraksi dengan database sangat erat kaitannya dengan *query*. Query adalah bagian dari bahasa pemrograman SQL(*Structured Query Language*) untuk berinteraksi dengan database. Oleh karena itu, pengambilan data pada database tidak bisa lepas dari peran query. Kinerja query inilah yang sangat menentukan waktu dari pengambilan data dalam database.

Mengingat pentingnya peran query dalam pengolahan data, penulis mencoba untuk membuat aplikasi yang dapat digunakan sebagai solusi alternatif untuk mendapatkan query alternatif yang diharapkan kinerjanya lebih baik / lebih cepat dalam pengambilan data. Aplikasi ini diharapkan mampu memberikan pilihan (*Opsi*) terhadap pengguna dalam mendapatkan query yang memiliki kinerja optimal dalam pengambilan data, khususnya jika dilihat dari sudut pandang waktu.

2. DASAR TEORI

Setiap database menyimpan data pada sebuah tabel. Ada beberapa jenis tabel yang disediakan oleh Oracle, tetapi secara default tabel yang dibuat adalah bertipe *Heap Organized*.

Pada tabel jenis *heap organized*, dalam mengatur letak data digunakan beberapa parameter, antara lain:

- *Highwater Mark*, berfungsi sebagai batas antara block yang pernah/sedang berisi data dengan block yang belum terisi data.
- *Freelists* adalah tempat pencatatan block-block kosong dibawah *highwater mark*.

Pada saat tabel baru dibuat, posisi *highwater mark* berada pada block pertama. Setiap ada data yang masuk kedalam tabel tersebut, posisi *highwater mark* akan bergeser keatas. Bila terjadi penghapusan data, maka block yang berisi data tersebut dikosongkan. Hal inilah yang menyebabkan adanya block-block kosong pada block yang terletak dibawah *highwater mark*. Block-block ini akan dicatat pada *freelists*. Pada kondisi ini, ketika ada data masuk, maka akan dilakukan pencarian block kosong yang berada dibawah *highwater mark*, tentunya dengan cara memeriksa daftar block kosong pada *freelists*. Jika ditemukan block kosong pada *freelists*, yang tentunya dengan kapasitas yang cukup untuk menampung data baru, maka pada block itulah data diletakkan. Hal inilah yang menyebabkan letak data tidak beraturan.

Selain tabel, objek yang sangat penting dalam proses pembacaan data adalah indeks. Konsep dari indeks adalah mencatat *keyword* dan alamat dari data yang dianggap penting secara terstruktur. Konsep indeks ini juga digunakan oleh DBMS Oracle, indeks menjadi salah satu faktor yang sangat mempengaruhi baik-buruknya performance proses manajemen data. Terlalu banyak indeks yang dibuat pada sebuah tabel dapat menyebabkan semakin lambatnya performance proses *insert*, *update*, *delete* pada table tersebut. Sebaliknya terlalu sedikit indeks akan menyebabkan turunnya performance pengeksekusian sebuah query. Menyediakan indeks dengan tepat menjadi tujuan utama yang ingin dicapai oleh setiap pemrogram agar dapat meningkatkan performance pada aplikasi mereka.

Oracle menyediakan beberapa macam tipe indeks, tetapi secara default indeks yang dibuat adalah bertipe *B*Tree Index*. B*Tree Index adalah struktur indeks yang dimiliki oleh banyak database-database yang lain. Disebut juga sebagai indeks konvensional yang proses pencariannya berdasarkan *binary search tree*.

3. CARA KERJA ORACLE DALAM MENGEKSEKUSI QUERY

Ada tiga tahap proses dalam pengeksekusian sebuah query pada Oracle yaitu:

- *Parse*, dilakukan di *client-process*
  Oracle melakukan cek terhadap sintaks dan semantik untuk memvalidasi perintah tersebut. Cek terhadap sintak untuk megetahui apakah perintah SQL tersebut benar atau salah. Sedangkan cek terhadap semantik untuk mengetahui apakah SQL tersebut berhak mengakses Objek yang diinginkan.
- *Execute*, dilakukan di *server*
  Merupakan pembacaan oleh server dan instance untuk penambahan, pengubahan atau penghapusan data.
- *Fetch*, dilakukan di *client-process*
  Khusus untuk pembacaan data.

Walau tidak dicantumkan di dokumentasi Oracle, sebelum dilakukan *execute*, terlebih dahulu dilakukan pembuatan *execution plan*, di mana hasil *optimasi execution plan* disimpan dalam sebuah tabel PLAN_TABLE.

Optimasi *Execution Plan* dilakukan oleh optimizer. Oracle DBMS memiliki dua jenis optimizer, yaitu:

- Rule-Based Optimizer (RBO)
- Cost-Based Optimizer (CBO)

CBO membuat execution plan hanya berdasarkan *hirarki* atau operasi presedensi [1]. Berikut ini adalah presedensi yang berlaku pada operasi SQL (Rangking terkecil memiliki presedensi tertinggi).

Rangking	Operasi
1	Single-row by RowId
2	Single-row by cluster-join
3	Single-row by hash-cluster key with unique-key or primary-key
4	Single-row by unique-key or primary-key
5	Cluster-join
6	Hash(ed) Cluster Key
7	Index(ed) Cluster Key
8	Composite-Index
9	Single-column Index
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort-merge join
13	MAX or MIN on indexed column
14	ORDER BY on indexed column
15	Full table scan

Single-Row by RowId

Operasi ini merupakan operasi yang paling cepat dijalankan. Server-process pasti akan memperoleh satu *row* saja. Untuk memperoleh satu row ini, server-process pertama kali akan membaca RowId dari indeks, lalu memakai RowId tersebut untuk fetch data dari tabel.

Single-Row by Cluster Join

Operasi ini berlaku saat *retrieve* hasil join tabel-tabel di suatu *cluster-segment*. Syarat:

- Ekspresi boolean di klausa WHERE memakai operasi "=" dan kolom cluster-key di suatu tabel.
- Hasilnya pasti satu row.

Single-Row by Hash Cluster Key

Kasus ini mirip dengan kasus sebelumnya, perbedaan hanya pada jenis cluster. Perbedaan ini tidak tampak pada *execution plan*. Syarat:

- Ekspresi boolean di klausa WHERE memakai operasi "=" dan cluster-key di suatu tabel. Untuk

composite cluster key, ekspresi boolean harus menyertakan operasi AND.

- Hasilnya pasti satu row.

Single-Row by Unique key or Primary Key

Operasi ini tersedia hanya bila klausa WHERE memakai semua kolom pembentuk unique key atau primary key yang melibatkan operasi "=". Server process men-*scan nique key index* atau *primary key* untuk memperoleh satu RowId dan mengakses tabel berdasarkan nilai tersebut.

Cluster Join

Operasi ini tersedia bila tabel-tabel yang di-*join* berada dalam satu cluster-segment dan klausa WHERE berisi ekspresi boolean yang menyertakan operasi "=" untuk kolom-kolom pembentuk cluster key, dan tidak ada ekspresi boolean untuk mendapatkan hanya satu row.

Hash(ed) Cluster Key

Operasi ini tersedia untuk *clustered-table* bila klausa WHERE memakai operasi "=" dengan kolom hashed cluster-key. Server proces memakai operasi hash untuk memperoleh nilai *hash cluster key* yang dispesifikasikan di query. Server-process memakai nilai ini untuk melakukan hash-scan di tabel.

Index(ed) Cluster Key

Operasi ini tersedia bila klausa WHERE dari query terhadap suatu clustered-table memakai operasi "=" untuk semua kolom indexed cluster-key. Server process melakukan unique-scan di cluster-index untuk memperoleh RowId dari suatu row dengan nilai cluster-key tertentu.

Server-process memakai nilai RowId ini untuk mengakses tabel secara cluster scan. Karena semua row dengan nilai cluster-key yang sama disimpan bersama, cluster mensyaratkan hanya satu nilai RowId untuk me-*retrieve* semua row tersebut.

Composite Index

Operasi ini tersedia bila hadir suatu indeks yang menyertakan semua kolom di klausa WHERE dan jumlah kolom lebih besar dari satu. Kolom ini disebut juga *concatenated index*.

Single-Column Index

Operasi ini mirip dengan composite-index, perbedaannya jumlah kolom di indeks sama dengan satu.

Bounded Range Search On Indexed Columns

Operasi ini tersedia bila klausa WHERE menyertakan semua kolom yang di-indeks atau menyertakan kolom awal dari composite indeks. Secara esensial, operasi ini tidak berbeda dengan operai *Single-Column Index*.

Perbedaan hanya ada secara internal, yakni ketika ada penerjemahan ekspresi seperti :

- Column_name=expression
- Column_name>[=] expression AND Column_name<[=] expression
- Column_name BETWEEN expression AND expression
- Column_name LIKE 'c%'

Setiap ekspresi boolean di atas adalah pencarian atas rentang terbatas (bounded range search). Server-process akan melakukan range-scan pada indeks dan mengakses tabel via nilai RowId.

Unbounded Range Search On Indexed Columns

Operasi ini mirip dengan *bounded range search on indexed columns*, bedanya tidak adanya salah satu nilai batas.

Sort-Merge Join

Operasi ini tersedia bila ada dua tabel yang relasi *parent-child* tidak diimplementasikan via *foreign-key primary-key relationship*.

MAX or MIN on Indexed columns

Operasi ini tersedia bila :

- Ada pemanggilan operasi MIN atau MAX dengan operand dari kolom yang di-indeks (single-column index) atau dari kolom-awal composite index. Indeks harus *un-clustered*. Operand bisa berupa ekspresi secara umum yang melibatkan konstanta, operasi "+", atau konkatenasi (|| atau CONCAT).
- Tak ada operasi lain di operasi proyeksi (klausa SELECT)
- Tak ada klausa "GROUP BY" dan klausa WHERE

ORDER BY indexed Columns

Operasi ini tersedia bila :

- Ada klausa "ORDER BY"
- Constraint PRIMARY KEY dan *Not Null* ada dan aktif sehingga minimal satu dari semua indexed-columns yang menjadi operand sort tidak bernilai null.
- Nilai parameter NLS_SORT=BINARY

Full Table Scan

Operasi ini merupakan operasi yang paling mahal dan paling dihindari. Operasi ini tersedia untuk setiap operasi retrieval tanpa klausa WHERE.

Adapun boolean ekspresi yang menyebabkan Full Table Scan adalah :

- Column1>[=]Column2, Column1 dan Column2 pada tabel yang sama
- Column1<[=]Column2, Column1 dan Column2 pada tabel yang sama
- Column_name IS NULL
- Column_name IS NOT NULL
- Column_name NOT IN
- Column_name !=exspression
- Column_name LIKE '%String'

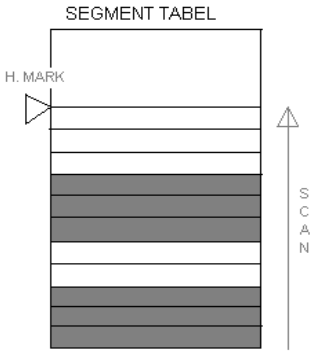
Expression1=Expression2, (Terdapat fungsi buit-in).

Tahap pertama, *parse*, dapat diabaikan jika perintah SQL telah terdapat dalam *shared pool*. Shared pool adalah satu komponen dari SGA yang digunakan sebagai tempat menampung perintah-perintah SQL [2].

Proses pengeksekusian query merupakan proses pembacaan data yang tersimpan dalam tabel. Oleh

karena itu, salah satu faktor yang sangat penting dalam proses pengeksekusian query adalah penentuan *access path* (Cara *engine* Oracle dalam membaca data dari sebuah tabel). Secara garis besar ada dua jenis *access path*, yaitu *table scan* dan *index scan*.

Table scan merupakan cara pengambilan data dengan cara langsung men-*scan* tabel.



Gambar 1 Table Scan

Pembacaan dengan *index scan* dilakukan dengan cara men-*scan* indeks terlebih dahulu dengan tujuan untuk mendapatkan alamat dari data yang dicari. Dengan mengetahui alamat data, pencarian data dapat lebih mudah dilakukan.

Pada query yang mengandung lebih dari satu tabel, proses penggabungan data dari dua tabel atau lebih merupakan proses yang sangat vital. Ada tiga cara yang dapat dilakukan yaitu:

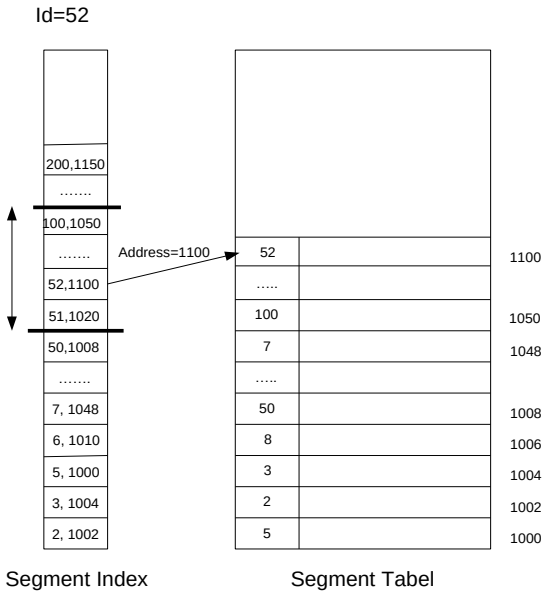
- **Nested Loop**

Merupakan cara yang paling sederhana. Konsep dari *nested loop* adalah konsep *looping* dimana setiap data yang di-*retrieve* salah satu tabel akan melakukan *scan* terhadap tabel lainnya.

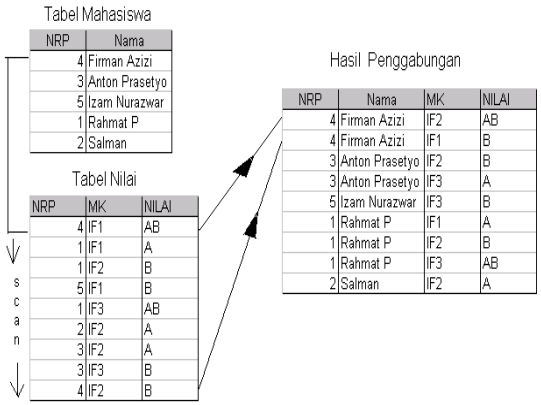
- **Merge Join**

Merge join merupakan operasi penggabungan dua tabel yang menggunakan konsep *Devide-and-Conquer*. Divide-and-Conquer merupakan salah satu algoritma terbaik dalam pemecahan masalah komputasi. Konsep dari teknik ini adalah memecah sebuah permasalahan menjadi permasalahan-permasalahan yang lebih kecil, yang tentunya tanpa mengubah bentuk permasalahan aslinya. Permasalahan-permasalahan yang lebih kecil inilah yang akan dicari solusinya, dimana solusi dari permasalahan-permasalahan yang lebih kecil ini digunakan untuk mendapatkan solusi dari permasalahan aslinya [5].

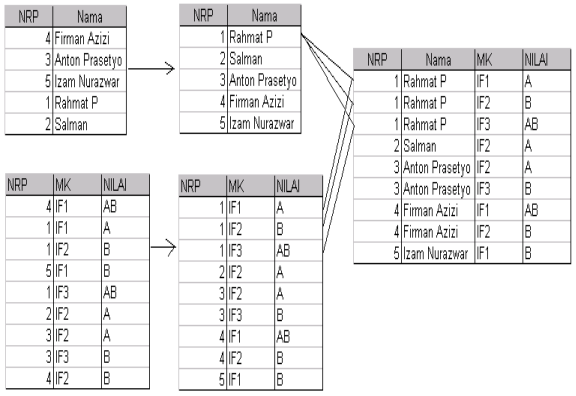
Pada proses penggabungan tabel dengan *Merge Join*, kedua tabel akan dianggap sebagai sub problem dari problem utama. Cara pemecahan permasalahan merga join adalah dengan cara mengurutkan data pada kolom-kolom yang merelasikan kedua tabel yang digabungkan. Tabel-tabel yang sudah urut dianggap sebagai solusi dari masing-masing sub problem. Dari hasil pengurutan kedua tabel tersebut, proses penggabungan data dapat dengan mudah dilakukan.



Gambar 2 Index Scan



Gambar 3 Nested Loop

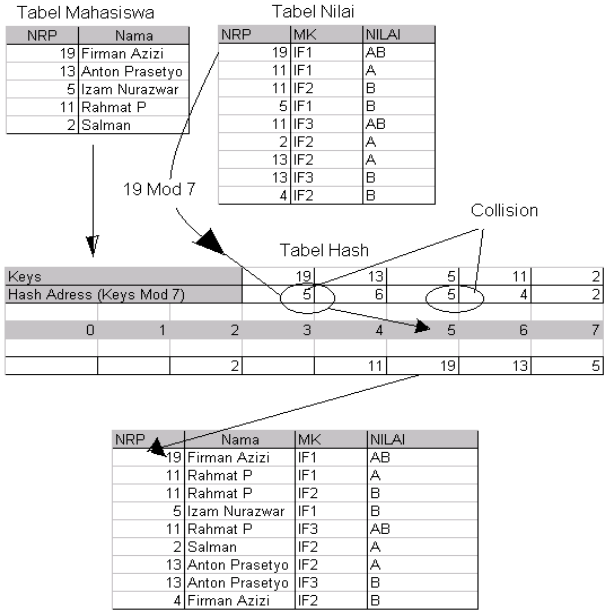


Gambar 4 Merge Join

- **Hash Join**

Hash join merupakan operasi penggabungan dua tabel yang menggunakan konsep *Space and Time Tradeoffs*, yaitu dengan mengoptimalkan ketersediaan space memori yang besar [5].

Pada operasi join ini, *key data* dari sebuah disebar pada *hash tabel* (konsep hash tabel sama dengan konsep array satu dimensi) dengan sebuah fungsi modulo. Dengan fungsi yang sama, table yang lainnya akan mencari data yang bersesuaian dengannya pada hash tabel.



Gambar 5 Hash Join

4. PERANCANGAN APLIKASI

Metode yang digunakan dalam mencari query-query alternatif adalah restrukturisasi SQL, yaitu dengan cara menganalisa struktur dari query bersangkutan. Perlu dipertegas bahwa, sebuah query dapat dianalisa hanya jika query tersebut sudah benar, baik secara sintak maupun secara sematik. Pengecekan benar salahnya sebuah query dilakukan di Oracle, dalam hal ini, aplikasi yang dibuat hanya berperan sebagai perantara saja. Semua pesan kesalahan yang muncul adalah pesan kesalahan dari Oracle.

Secara umum, rangkaian proses pencarian query-query alternatif pada aplikasi ini nantinya terdiri dari empat tahap yaitu Mengurai query menjadi subquery, mengurai query per-klausa, menentukan *access path* dan *operasi join* dan restrukturisasi Query.

▪ MENGURAI QUERY MENJADI SUBQUERY-SUBQUERY

Tahap ini merupakan tahap penyederhanaan permasalahan. Dengan mengurai query menjadi beberapa query sederhana analisa lebih lanjut akan dipermudah karena query sederhana memiliki struktur yang sederhana. Jadi bila query yang di-*input*-kan tidak mengandung sub query maka tahap ini bisa dilewati. Yang menjadi masalah adalah bagaimana jika query yang di-*input*-kan merupakan query kompleks. Query itu harus disederhanakan menjadi beberapa sub query. Karena sub query tidak selalu berbentuk query sederhana maka proses penguraian query dilakukan secara rekursif sampai didapatkan query- query sederhana.

Cara menentukan apakah query yang di-*input*-kan pengguna termasuk dalam kategori query sederhana atau query kompleks adalah dengan cara menghitung klausa SELECT yang ada pada query tersebut.

Setiap sub query akan diawali dengan kurung buka "(" dan diakhiri dengan kurung tutup ")". Jadi dapat disimpulkan bahwa jika terdapat kurung buka yang diikuti kata "SELECT" berarti semua kata yang berada di antara kurung buka dan kurung tutup adalah komponen dari sub query.

Berikut ini adalah algoritma yang digunakan dalam mengurai query menjadi query- query sederhana.

```
Function GetSubQuery(Query)
N ← length(query)
Key=""
Awal← 0
Subquery ← ""
For I=1 to N
   Key ← Substr(query,I,7)
   if  Key="(select"
      Awal←I
   elseif Substr(query,I,1)= ")"
      SubQ←Substr(Query,awal,I)
      GetSubQuery(SubQ)
   end if
Next
End function
```

▪ MENGURAI QUERY PER-KLAUSA

Pada tahap ini akan dilakukan penguraian komponen-komponen penyusun query dalam kelompok yang lebih kecil yaitu berdasarkan klausa. Ada enam jenis klausa yang menyusun query, yaitu klausa SELECT, klausa FROM, klausa WHERE, klausa GROUP BY, klausa HAVING dan klausa ORDER BY. Keberadaan Klausa SELECT dan klausa FROM bersifat mutlak, sedangkan keempat klausa lainnya persifat opsional.

Setiap kemunculan klausa ini selalu didahului oleh kata kunci. Munculnya klausa SELECT selalu diawali dengan kata kunci SELECT, klausa FROM selalu diawali dengan kata kunci FROM, klausa WHERE selalu diawali dengan kata kunci WHERE, klausa GROUP BY selalu diawali dengan kata kunci GROUP BY, klausa HAVING selalu diawali dengan kata kunci HAVING dan klausa ORDER BY selalu diawali dengan kata kunci ORDER BY.

```
//Q adalah sebuah query
Function GetKlausa(Q)
N             ← length(Q)
Awal          ← 6;
Stat          ← 0
Klausa[]      ← "";
For I=1 to N
  If Substr(Q,I,4) = "FROM"
Kl[stat]←Substr(Q,Awal,I)
Stat←1
Awal←I+4
  ElseIf Substr(Q,I,5)="WHERE"
Kl[stat] ←Substr(Q,Awal,I)
Stat←2
       Awal←I+5
  Elseif Substr(Q,I,8)="GROUP BY"
Kl[stat] ←Substr(Q,Awal,I)
Stat←3
       Awal←I+8
  ElseIf Substr(Q,I,6)="HAVING"
Kl [stat] ←Substr(Q,Awal,I)
Stat←4
       Awal←I+6
  ElseIf Substr(Q,I,8)="ORDER BY"
Kl[stat] ←Substr(Q,Awal,I)
Stat←5
       Awal←I+8
  end if
Next
Kl[stat] ←Substr(Q,Awal,I)
End function
```

- **MENENTUKAN ACCESS PATH DAN OPERASI JOIN**

Tahap ini merupakan tahapan terpenting dari aplikasi ini karena dilakukan proses *logic* karena penentuan *access path* dan *operasi join* merupakan faktor yang sangat penting bagi kinerja sebuah query.

Access path merupakan operasi pembacaan data dari database. Secara garis besar ada dua metode yang disediakan Oracle, yaitu pembacaan data langsung ke tabel (*table scan*) atau dengan membaca indeks terlebih dahulu (*index scan*) sebelum membaca isi tabel.

Untuk menentukan access path ada tiga faktor yang harus dipertimbangkan, yaitu:

- Jumlah record data yang di-*retrieve* dari tabel yang diakses
- Prosentase jumlah record data yang di-*retrieve* dari tabel yang diakses.
- Field data yang ingin dibaca.

```
//Fungsi untuk mengecek apakah ada index
yang mengandung semua field yang ada
pada Klausa SELECT
Function CekIndex(KlSelect) Int
For I=0 to jmlhIndex-1
If klausaSelect=ItemIndex[i]
      Return 1
Exit function
End if
Next
Return 0
End function

Function GetAceessPath(Kl[])
//kl[0] ← klausa SELECT
//kl[1] ← klausa FROM
//kl[2] ← klausa WHERE
Rtriev ←0
JmlhData ← 0
Stat ← CekIndex(Kl[0])
If stat=1
Use Index_Scan
Else
Rtriev←GetRtriev(Kl[1],Kl[2])
JmlhData←GetJmlhData(Kl[1])
If Rtriev/JmlhData<0.15 and Rtriev<10000
      Use Index_Scan
Else
      Use Table_Scan
End if
End if
End function
```

Operasi join merupakan operasi penggabungan dua kumpulan data. Ada tiga metode yang disediakan Oracle, yaitu Nested Loop, Hash Join dan Merge Join. Ketiga metode tersebut dibedakan berdasarkan konsep yang melandasinya.

Pada klausa FROM yang mengandung lebih dari dua tabel, posisi tabel harus diperhatikan karena posisi tabel akan sangat mempengaruhi tabel mana yang harus digabungkan terlebih dahulu dan ini sangat berpengaruh pada proses penggabungan berikutnya. Hal ini tentu saja mempengaruhi pertumbuhan waktu proses yang terjadi pada saat proses penggabungan tabel.

Setidaknya ada enam faktor yang harus dipertimbangkan dalam melakukan operasi join, yaitu:

- Jumlah record data yang di-*retrieve* dari masing-masing tabel.
- Kecepatan kinerja operasi hash
- *Join condition* yang merelasikan kedua tabel
- Prosentase jumlah record data yang di-*retrieve* dari sebuah tabel
- Keterurutan data dari tabel-tabel yang akan digabung.
- Posisi Tabel

```
//Fungsi untuk mengecek apakah ada index
yang mengandung semua field yang ada
pada Klausa SELECT
Function GetJoin(Tbl1,Tbl2, KlWh)
//kl[0] ← klausa SELECT
//kl[1] ← klausa FROM
//kl[2] ← klausa WHERE
//kl[5] ← klausa ORDER BY
Rtriev ←0
JmlhData1  ← 0
JmlhData2  ← 0
```

```
Prsn1 ← 0
Prsn2 ← 0
Kl[1]←AturPosTbl(Kl[1],Kl[2])
Stat←CekUrut(Kl[1],Kl[2],Kl[5])
Rtriev← GetRtriev(Tbl1,Kl[2])
JmlhData1 ← GetJmlhData(Tbl1)
Prsn1 ← Rtriev/JmlhData1
Rtriev ← GetRtriev(Tbl2,Kl[2])
JmlhData2 ← GetJmlhData(Tbl2)
Prsn2 ← Rtriev/JmlhData2
If filter="=" then
If stat=1
Use Merge
Exit function
   Else
If   (Prsn1<0.15   or   Prsn2<0.15)   and
JmlhData1<10000 and jmlhData2<10000
             Use Nested
             Exit function
End if
   End if
   Use Hash
Else
If   (Prsn1<0.15   or   Prsn2<0.15)   and
JmlhData1<100000 and jmlhData2<100000
       Use Nested
       Exit function
End if
Use Merge
End if
End function
```

- **RESTRUKTURISASI SQL**

Tahap ini merupakan tahap *finishing* dari proses optimasi. Query yang sudah terpecah-pecah disusun kembali, dan tidak menutup kemungkinan terjadi pengubahan struktur query. Penyusunan kembali dari klausa ke bentuk query sederhana dilakukan dengan meletakkan klausa secara urut sesuai dengan *grammar* yang telah dibuat. Proses ini sangat mudah dilakukan mengingat setiap klausa tersimpan secara urut dalam sebuah array sehingga untuk mengembalikannya kedalam betuk query sederhana tinggal menggabungkan isi dari array secara urut.

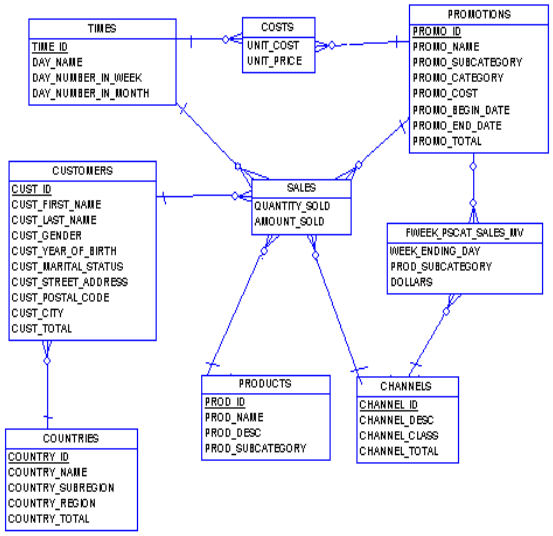
```
Function Restruktur_1(Kl []) Text
SubQ ← ""
For I=0 to 5
SubQ ← SubQ & Kl [I]
Next
End function
```

Sub query-sub query baru yang telah dibentuk akan disusun menjadi query kompleks. Query inilah yang akan ditawarkan kepada pengguna. Proses penyusunannya dilakukan dengan cara menggantikan posisi query sederhana yang lama dengan query sederhana hasil optimasi.

```
//Qlama merupakan  query sederhana yang
masih belum diolah
//Qbaru merupakan  query sederhana hasil
optimasi
//Q merupakan query yang di-input-kan
pengguna
Function     Restruktur_2(QLama,QBaru,Q)
Text
N ← length(Q)
Key=""
M ← length(QLama)
Hasil ← ""
For I=1 to N-M
  Key ← Substr(Query,I,I+M)
  If QLama=Key
 Hasil=Substr(Q,1,I) & QBaru
 Hasil=hasil & Substr(Q,I+M,N)
  End if
Next
End function
```

5. HASIL UJICOBA

Pada uji coba ini akan digunakan skema berikut.



Gambar 6 Skema Ujicoba

Ujicoba dibuat dengan dua query untuk menunjukkan bagaimana peran *access path* dan *operasi join* terhadap *response time* yang diperlukan dalam mengeksekusi sebuah query.

- **Skenario 1**

  Query Input: SELECT * FROM sales WHERE prod_id < 570

  Keterangan :
  - Jumlah record data pada tabel sales diatas satu juta.
  - Jumlah record data yang di-*retrieve* 82.677 atau sekitar 8% dari jumlah record data pada tabel sales
  - Access path yang digunakan *index scan*
  - Waktu yang diperlukan 69 detik.

Query Output: SELECT /*+ FULL(SALES) */sales.prod_id, sales.promo_id, sales.quantity_sold, sales.amount_sold FROM sales WHERE prod_id < 570

Keterangan :

- Access path yang digunakan *table scan*
- Waktu yang diperlukan 3 detik.

Analisa :

Pada *query input*, untuk membaca data dari tabel *sales* dilakukan dengan cara *index scan*. Hal ini tentu saja sangat tidak efektif mengingat besarnya data yang di-*retrieve*. Jumlah record data dikatakan besar jika jumlah rekord data lebih besar dari 10.000 [6].

Dengan menggunakan *Hints FULL(SALES)*, menunjukkan Oracle untuk menggunakan *table scan* dalam pembacaan data pada tabel sales.

- **Skenario 2**

Query Input: SELECT c.cust_first_name, s.channel_id FROM customers c,sales s WHERE c.cust_id=s.cust_id

Keterangan :

- Jumlah record data pada tabel sales diatas satu juta.
- Jumlah record data pada tabel customer sekitar 49.500
- Operasi join yang digunakan *nested loop*
- Waktu yang diperlukan 16 detik.

Query Output: SELECT /*+ ORDERED USE_HASH(S) */c.cust_first_name, s.channel_id FROM customers c,sales s WHERE c.cust_id=s.cust_id

Keterangan :

- Operasi join yang digunakan *Hash join*
- Waktu yang diperlukan 9 detik.

Analisa :

Pada *query input*, operasi join yang digunakan dalam menggabungkan data dari tabel sales dan tabel customer adalah *nested loop*, padahal operasi join ini sangat tidak efisien dalam menggabungkan dua kelompok data dalam jumlah besar, apalagi diantara dua tabel tersebut tidak ada tabel yang pantas menjadi *driving table* (salah satu kriteria tabel dijadikan *driving table* adalah sedikitnya data yang di-*retrieve* dari sebuah tabel).

Dengan mengganti operasi join menjadi *hash join* akan meningkatkan kinerja panggabungan data.

6. KESIMPULAN DAN SARAN

Setelah dilakukan uji coba terhadap aplikasi ini, maka dapat diambil kesimpulan sebagai berikut :

1. Penggunaan indeks pada pembacaan data dalam jumlah besar dari sebuah tabel sangat tidak efektif karena akan memperbesar proses pemanggilan I/O *calls* yang menyebabkan *physical reads* menjadi sangat besar.
2. Penggunaan *hash join* dalam menggabungkan dua buah tabel akan mengurangi *consistens gets* karena dengan menggunakan hash join proses penggabungan tiap data dari kedua tabel dapat berjalan lebih cepat.
3. Mendahulukan proses penggabungan dua tabel yang me-*retrieve* data terkecil mengurangi *consistens gets* karena semakin kecil data yang di-*retrieve* akan mengurangi jumlah data yang harus diolah pada tahap berikutnya.

Berikut ini adalah saran untuk kemungkinan pengembangan lebih lanjut yang bisa dilakukan :

1. Aplikasi ini dirancang untuk database yang mengandung tabel dan indeks default Oracle. Aplikasi ini dapat dikembangkan untuk mengoptimasi database yang tidak hanya mengandung tabel dan indeks default Oracle.
2. Pengembangan aplikasi yang memperhitungkan ketersediaan memori yang disediakan oleh administrator database.

7. DAFTAR PUSTAKA

1. Bernaridho I Hutabarat, Msc, OCP., “Oracle 8i/9i Performance Tuning”, Andi, Jakarta, 2004.
2. Evara Syamsiar, S.Kom., “Oracle 9i : Optimasi Database”, Elex Media Komputindo, Surabaya, 2004.
3. Imam Heryanto dan Budi Raharjo, “Memahami Konsep SQL dan PL/SQL di Oracle”, Informatika Bandung, Bandung, 2002.
4. Korth, H. dan Silberschatz, A., “Database System Concepts 2nd”, McGraw-Hill, 1991.
5. Levitin, Anany, “Introduction to The Design And Analysis of Algorithms”, Villanova University, 2002.
6. Oracle Database, “Performance Tuning Guide”, 2003