BAB III

PELAKSANAAN KERJA MAGANG

3.1. Divisi, Kedudukan dan Koordinasi

3.1.1. Divisi

Penulis ditempatkan didalam departemen IT unit *database*. Unit *database* merupakan unit yang melakukan management pada setiap data dalam sebuah *table* yang akan digunakan dalam aplikasi atau system perusahaan.

Database Administrator ini memastikan bahwa data yang ada dalam *table* memiliki constraint dan *index* yang gunanya untuk membuat agar *table* yang diakses menjadi lebih cepat. Tidak hanya itu seorang *database* administrator juga harus rutin untuk melakukan *backup* agar mencegah terjadinya kehilangan, maupun kerusakan data perusahaan dalam *database*. Fungsi utama dari *database* administrator yaitu memonitori keadaan *database* yang digunakan perusahaan, meningkatkan kinerja dari *database*, dan melakukan maintenance terhadap *database* perusahaan.

Penulis sebagai *database* administrator didalam perusahaan Dipa Healthcare harus dapat mengelola setiap data maupun *query* yang masuk kedalam *database*, *query* tersebut dapat dimaksimalkan dengan cara *re-write query*, memberikan *index* dan melakukan pengurangan *table* yang diakses sesuai dengan kebutuhan perusahaan, dan melakukan pembagian *backup* sesuai *module* yang digunakan agar mempercepat waktu dari *backup* itu sendiri. Hal tersebut

3.1.2. Kedudukan dan Koordinasi

Dalam praktek kerja magang ini, kedudukan yang ditempati oleh penulis yaitu *Database Administrator (DBA)* dalam *project tuning* and *backup* untuk *department* IT Dipa Healthcare. Penulis melakukan perbaikan pada *query* didalam *database* guna mempercepat performance dari *query* tersebut dan melakukan *split backup* pada beberapa modul didalam *database*.

3.2. *Detail* Proyek

Tuning dan *split backup database* ini merupakan sebuah *project* yang diperlukan oleh *department* IT Dipa Healthcare dikarenakan *department* IT belum memiliki seorang DBA (*Database Administrator*) sehingga *database Oracle* yang digunakan sehari-hari untuk menjalankan proses bisnis tidak ter-*manage* dengan baik dan mengakibatkan lambatnya performance dari *database Oracle* itu sendiri. *Tuning Query sql* dimaksudkan untuk memperbaiki peforma dari *query* yang dipakai untuk menampilkan data yang dibutuhkan oleh perusahahaan agar data yang dinginkan dapat muncul lebih cepat dan tidak memberikan beban terlalu berat kepada *server*. *Split Backup* merupakan salah satu cara untuk *backup* dimana seorang DBA tidak perlu melakukan *backup* total terhadap, melainkan *backup* secara per *module* sehingga mempercepat waktu untuk mem-*backup* dibandingkan dengan mem-*backup* secara keseluruhan. Untuk dapat melakukan *Tuning* maupun *Split Backup database*, seorang DBA harus mengetahui karaktersitik *database Oracle* dan menguasai seluruh *query* yang dibutuhkan agar mempersingkat waktu kerja.

3.3. Tugas-Tugas yang Dilakukan

Tugas yang dilakukan penulis saat melakukan praktek kerka magang di *department* IT sebagai *Database Administrator* (DBA) pada *project Tuning* and *split backup* adalah:

1. Mempelajari dasar-dasar *database oracle*
2. Membuat cara alternatif untuk *backup database* pada *module* buatan sebanyak 3 *module*.
3. Melakukan *Tuning Query* pada beberapa *database* server.
4. Mencari solusi atas *Locking Table*.

3.4. Uraian Pelaksanaan Kerja Magang

3.4.1. Mempelajari dasar-dasar *database* oracle

Sebagai *Database Administrator* (DBA) di dalam proyek *Tuning* dan *Split backup*, penulis harus mengetahui banyak hal mengenai *Database Oracle* terutama dalam hal struktur *query sql* dan cara untuk mem-*backup* data *Oracle*. Penulis diberikan sebuah dokumen mengenai panduan menggunakan *database Oracle*, mulai dari hal yang mendasar yaitu menginstal *database* sampai dengan melakukan management *database* termasuk melakukan *backup*. Penulis tidak hanya belajar lewat buku panduan tetapi juga lewat internet, penulis menemukan sebuah *website* forum mengenai *Oracle* secara lengkap, lewat forum tersebut penulis mendapatkan banyak sekali pengetahuan mulai dari teori dasar sampai dengan cara untuk melakukan *tuning query*.

Setelah belajar mengenai dasar dasar *database Oracle*, penulis melakukan diskusi dengan pembimbing lapangan mengenai pembuatan *database dummy*. *Database Dummy* digunakan untuk mempelajari lebih dalam penggunaan *query* didalam *database Oracle*, pembuatan *database dummy* ini menghindari adanya *data corrupt* yang mungkin akan terjadi jika dibandingkan menggunakan *database server* yang sebenarnya. Data yang

terdapat dalam *database dummy* juga merupakan data yang berasal dari tahun 2006. Penulis diberikan akses masuk kedalam *database server* 10.10.11.125 (CRM), *database* CRM tersebut merupakan *database* simulasi untuk melakukan berbagai macam simulasi penggunaan *query* dengan menggunakan data perusahaan yang sesungguhnya.

Dalam memanipulasi data didalam *database Oracle* milik PT Dipa Pharmalab Intersains, penulis menggunakan aplikasi *SQL editor* yaitu Toad versi 9.0.1 yang digunakan juga oleh *developer* Dipa untuk memanipulasi data. Aplikasi Toad ini sangat membantu penulis karena seluruh kegiatan didalam *database Oracle* dapat di monitor dengan *User Interface* yang mempermudah melakukan manipulasi data dalam *database*.

3.4.2. Membuat cara alternatif *backup database* pada *module* buatan sebanyak 3 *module*

Perusahaan Dipa Healthcare menerapkan full *backup* untuk satu *database server*, penulis diberikan tugas oleh pembimbing lapangan untuk dapat membagi data *backup* kedalam 3 *module* besar, sehingga *database* tidak perlu di-*backup* secara keseluruhan. *Split backup* ini dilakukan pada *database dummy* sehingga mencegah terjadinya *error* pada *database* sesungguhnya. *Split backup* merupakan istilah yang digunakan oleh penulis sebagai cara alternatif yang penulis gunakan untuk melakukan pembagian *data* dalam *database* kedalam 3 *module* besar.

3.4.3. Melakukan *tuning query* pada beberapa *database* server

Pembimbing memberikan tugas berupa *tuning query* dan *view* didalam *database*, *query* dan *view* tersebut merupakan bagian terpenting karena digunakan untuk menampilkan data sales selama periode tertentu. *Tuning query* dan *view* diperuntukan untuk mengurangi beban dan waktu yang digunakan oleh database dalam memproses *query* maupun *view* tersebut. Banyak faktor yang menyebabkan sebuah *query* dapat berjalan

lambat diantaranya karena penggunaan *index* yang kurang maksimal, *join table* yang banyak, penggunaan klausa distinct dibandingkan dengan group by dalam menyortir data yang duplikat, penggunaan *table* yang tidak diperlukan sehingga memperlambat proses pemangilan *query*. Terdapat 2 *View* dan 1 *query* yang harus dapat di-*tuning* sehingga menghasilkan performa yang lebih baik dalam *database*.

3.4.4. Mencari solusi atas *locking table*

Locking Table merupakan keadaan dimana sebuah *table* terkunci akibat terdapat *session* yang belum selesai dan di *commit*, sedangkan ada *session* lain yang ingin mengakses *table* tersebut, sehingga menyebabkan *locking table*. Permasalahan *locking table* ini sering terjadi dalam *database* pada saat penutupan *sales* akhir bulan, dan terjadi pada *table* secara acak. Pembimbing lapangan memberikan tugas untuk mencari solusi atas masalah tersebut, agar kejadian *locking table* tidak terjadi lagi pada *table-table* dalam *database*.

3.5. Hasil Kerja

3.5.1. Alternatif *backup plan*

Backup database merupakan sebuah kegiatan untuk menyimpan *data* perusahaan dalam suatu periode tertentu agar pada saat terjadi masalah didalam *database* perusahaan data yang telah disimpan dapat dipergunakan kembali setelah proses *recovery*. Perusahaan Dipa Healthcare biasanya dalam melakukan kegiatan *backup* data pada *database* dilakukan secara menyeluruh dalam kurun waktu tertentu, waktu yang dibutuhkan untuk mem-*backup* keseluruhan *database* memakan waktu yang lama dan resource yang besar, maka dari itu diperlukan beberapa cara alternatif untuk mengurangi waktu dan resource yang dibutuhkan *database* dalam memproses *backup*. Perusahaan Dipa Healthcare sudah memiliki sebuah

prodecure backup plan yang dilakukan setiap hari pada saat malam hari, backup yang dilakukan di malam hari memiliki guna mencegah adanya entry data yang masuk karena traffic data biasanya sudah tidak ada pada malam hari.

Pembimbing lapangan meminta penulis untuk dapat membagi *backup* bersebut menjadi beberapa bagian sesuai dengan *module* yang dibutuhkan perusahaan. Fungsi dari hal tersebut adalah agar pada saat melakukan *backup* tidak akan banyak memakan waktu dan lebih praktis dalam melakukan management *table* begitu pula pada saat dilakukan recovery *database* jika terjadi permasalahan dalam *database*. Untuk meminimalisir dampak terjadinya *error* di dalam *database* perusahaan, penulis mengambil inisiatif untuk membuat *database dummy*. *Database dummy* merupakan *database* yang berisi *data test* dan beberapa *random table*, yang diperuntukkan untuk melakukan *backup* untuk mencegah *error* pada database yang sebenarnya.

Penulis memiliki sebutan tersendiri untuk melakukan Penulis Untuk melakukan *Split Backup* dibutuhkan *user* baru yang telah diberikan izin oleh *master user* (SYS) untuk dapat melakukan *backup*. Tujuan dari pembuatan *user* baru yaitu *database Oracle* yang digunakan perusahaan telah berjalan cukup lama dan *table-table* yang ada dalam *database* tidak dapat begitu saja dipindahkan *directory tablespace*-nya, sehingga penulis harus membuat *user* baru dan mengatur ulang peletakan *table* sesuai *module*. Sebelum melakukan *Split Backup* penulis harus melakukan pemindahan *table* kedalam *user* lain. Ada 2 cara untuk melakukan pemindahan *table* kedalam *user* lain yaitu:

1. *CTAS* (*Create Table AS*)

   CTAS merupakan fitur yang disediakan oleh *Oracle database* untuk dapat men-copy keseluruhan isi *table* akan tetapi pada saat *table* ter-copy, *index* dan constraint dalam *table* tidak ikut berpindah. Keuntungan menggunakan cara ini yaitu lebih cepat dibandingkan dengan menggunakan export dan import *table*. Kelemahan terbesar

jika menggunakan *CTAS*, DBA harus membuat ulang *index* dan constraint agar sama seperti *table* aslinya.

User yang digunakan oleh perusahaan Dipa yaitu *Compiere* pada seluruh *database* server secara default. *User* compiere digunakan untuk segala kegiatan didalam *database* termasuk menambahkan, memanipulasi,dan menghapus data dalam *database*. Pada penerapan split *backup* ini, penulis menggunakan *user* test sebagai *user* untuk mensimulasikan penggunaan CTAS dan export *table* dalam split *backup*.

```
93
94   SQL> CREATE TABLE TESTING.QAD_GL TABLESPACE DUDUH AS SELECT * FROM CLONING.QAD_GL;
95
96   Table created.
97
98   SQL> CREATE TABLE TESTING.QAD_DOMAIN TABLESPACE DUDUH AS SELECT * FROM CLONING.QAD_DOMAIN;
99
100  Table created.
101
102  SQL> CREATE TABLE TESTING.QAD_DEBTOR TABLESPACE DUDUH AS SELECT * FROM CLONING.QAD_DEBTOR;
103
104  Table created.
105
```

Gambar 3. 1 Contoh penerapan CTAS

Gambar 3.1 diatas merupakan contoh penerapan CTAS pada saat penulis melakuan pemindahan *table-table* dari *user* compiere menujur *user* test. Pada saat pemindahan sebuah *table*, kita dapat mengatur *table* tersebut akan di simpan kedalam *table*space tertentu agar mempermudah pengelompokan *table*.

2. *Export* dan *Import*

   Export dan *import* merupakan salah satu cara agar dapat memindahkan *table* dengan sangat baik karena *index* dan constraint dalam *table* juga ikut bersama dengan *table* tersebut. Penggunaan *Export* dan *import* dapat menggunakan *EXPDP* dan *IMPDP*. Hal pertama yang perlu dilakukan untuk memindahkan *table* adalah dengan men-*export table* sesuai dengan *module* yang diinginkan, selanjutnya file *export* tersebut di *import* kedalam *user* baru, dan pilih *tablespace* mana *table* akan di pindahkan.

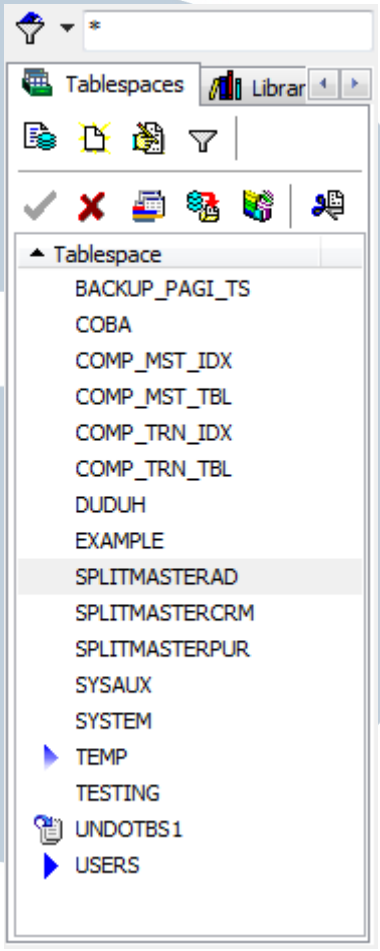
Gambar 3. 2 Contoh penggunaan IMPDP

Gambar 3.2 diatas merupakan contoh dari proses pemindahan data (import) *table* menuju *database* dummy, dengan melakukan import dengan menggunakan IMPDP data yang masuk akan bersamaan dengan seluruh strukturnya seperti data, *index* dan constraint.

Setelah melakukan pemindahan *table* sesuai kedalam *user* baru, hal selanjutnya yaitu melakukan *Split Backup*. Split *backup* merupakan cara dimana sebuah *database* dibagi kedalam beberapa bagian sesuai *module* perusahaan yang disebut dengan *table*space. Terdapat beberapa cara untuk dapat melakukan *Split Backup database* diantaranya:

1. 3 *Tablespace* 1 *Datafile*

   Penulis membuat 3 *tablespace* dengan masing masing 1 *datafile* dalam *database dummy*, tujuan dibuatnya 3 *tablespace* tersebut yaitu untuk membuat *table* tiap *module* menjadi satu agar setiap kali melakukan *backup* pada *database*, seorang *database* administrator hanya perlu mem-*backup* 1 datafile.

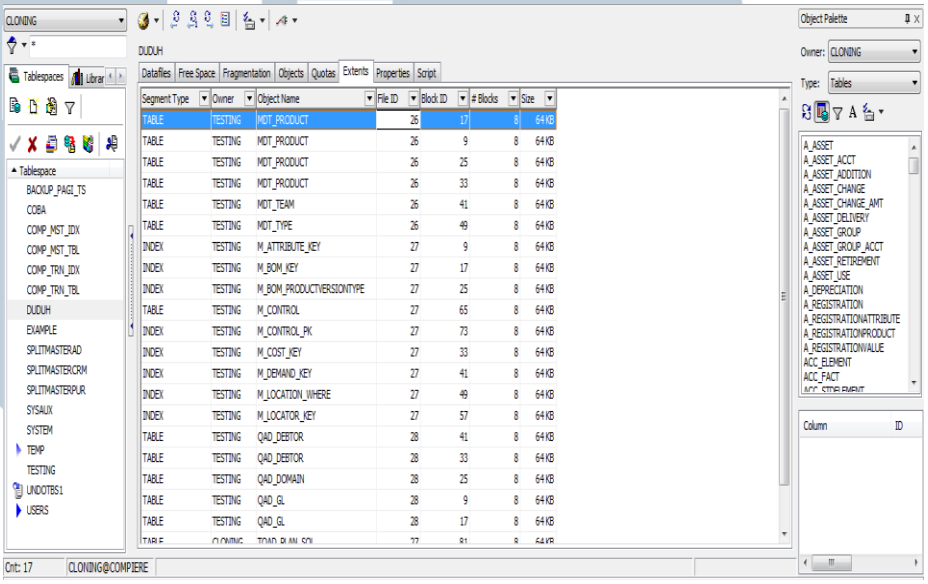


Gambar 3. 3 List Tablespace

Setiap kali melakukan *backup database*, DBA hanya perlu melakukan *backup* sesuai dengan *tablespace* (*module*) yang diinginkan. Hal tersebut sangat mengurangi beban dari *database* itu sendiri dan waktu yang diperlukan untuk mem-*backup tablespace* dibandingkan dengan mem-*backup* secara keseluruhan *database*. Kelebihan yang ditawarkan dengan cara ini yaitu mem-*backup* dan me-*restore tablespace* menjadi lebih mudah karena file yang digunakan sedikit dan archivelog yang dihasilkan juga sedikit. Tetapi kekurangannya yaitu akan membuat *tablespace* menumpuk didalam *database* menumpuk sesuai dengan banyak *module* yang digunakan oleh perusahaan.

2. 1 *Tablespace* 3 *Datafile*

Penulis membuat 1 *tablespace* dengan 3 *datafile* dalam *database dummy*, tujuan dibuatnya 1 *tablespace* tersebut yaitu mengurangi penggunaan *tablespace* yang tidak diperlukan dengan cara membuat *datafile* sesuai dengan *module* yang akan digunakan untuk mem-*backup*.

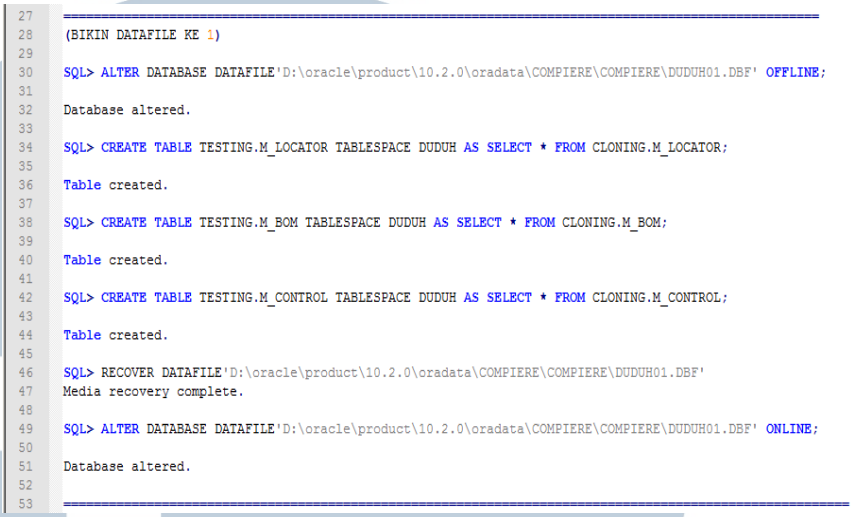


Gambar 3. 4 *Datafile* dalam *tablespace*

Dalam penerapannya untuk dapat memindahkan dan membuat *table* sesuai dengan *datafile* yang dituju diperlukan beberapa cara yaitu:

a. Membuat *datafile* offline

Jika DBA telah membuat *datafile* lebih dari 1 dan tidak ingin *table* yang dibuat masuk kedalam *datafile* yang salah karena pada dasarnya pada saat melakukan pemindahan *table*, maka DBA dapat membuat *datafile* offline sehingga pada saat membuat dan memindahkan *table* kedalam *datafile*, *table* tersebut tidak akan masuk kedalam *datafile* lain.



```
27  ================================================================================================
28  (BIKIN DATAFILE KE 1)
29
30  SQL> ALTER DATABASE DATAFILE'D:\oracle\product\10.2.0\oradata\COMPIERE\COMPIERE\DUDUH01.DBF' OFFLINE;
31
32  Database altered.
33
34  SQL> CREATE TABLE TESTING.M_LOCATOR TABLESPACE DUDUH AS SELECT * FROM CLONING.M_LOCATOR;
35
36  Table created.
37
38  SQL> CREATE TABLE TESTING.M_BOM TABLESPACE DUDUH AS SELECT * FROM CLONING.M_BOM;
39
40  Table created.
41
42  SQL> CREATE TABLE TESTING.M_CONTROL TABLESPACE DUDUH AS SELECT * FROM CLONING.M_CONTROL;
43
44  Table created.
45
46  SQL> RECOVER DATAFILE'D:\oracle\product\10.2.0\oradata\COMPIERE\COMPIERE\DUDUH01.DBF'
47  Media recovery complete.
48
49  SQL> ALTER DATABASE DATAFILE'D:\oracle\product\10.2.0\oradata\COMPIERE\COMPIERE\DUDUH01.DBF' ONLINE;
50
51  Database altered.
52
53  ================================================================================================
```

Gambar 3. 5 Contoh *query* untuk *offline datafile*

Gambar 3.5 diatas merupakan cara untuk memindahkan *table* kedalam salah satu datafile dengan meng-offline-kan datafile. Cara ini merupakan cara yang paling baik pada saat sebuah perusahaan sudah menjalankan *database* cukup lama dan seluruh arsitekturnya sudah terbentuk.

b. Membuat *datafile* secara berurutan

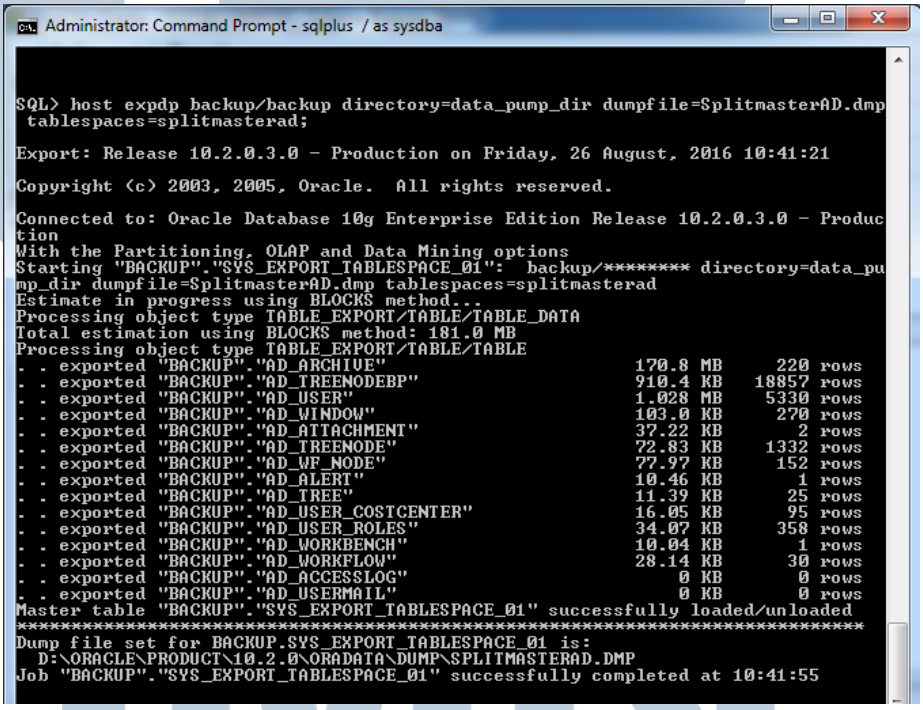
Cara kedua untuk dapat memasukan *table* sesuai dengan *module* yang diingkan kedalam *datafile* yaitu dengan membuat *datafile* secara berurutan. Pada saat *datafile* pertama sudah terisi dengan *module* yang dinginkan, DBA dapat membuat *datafile* baru dan memasukan *module* sebanyak *datafile* yang diperlukan.

Cara ini juga memiliki kelemahan yaitu DBA harus dapat mengingat setiap *module* tersimpan dalam *datafile* mana. Hal tersebut dapat dihindari dengan cara me-rename *datafile* sesuai dengan *module*, sehingga dapat mempermudah DBA dalam mencari *module* dalam *datafile*.

3. *EXPDP*

EXPDP merupakan sebuah fitur yang disediakan oleh *Oracle*

database untuk dapat melakukan *backup* kedalam bentuk file dmp. *EXPDP* dapat dijalankan pada aplikasi Toad, command prompt, dan *sql* plus. *EXPDP* memungkinkan DBA untuk dapat mem*backup table* satu persatu sampai dengan keseluruhan *database*. Fitur ini mempercepat kinerja *backup* dan setiap *table* yang di*backup* memiliki *index* dan constraint, sehingga mempermudah DBA untuk melakukan restore *database* dibandingkan dengan menggunakan 2 cara sebelumnya. Kelemahan dari cara ini yaitu setiap melakukan *backup*, tidak tersimpan dalam archivelog *database*.



Gambar 3. 6 Contoh penggunaan EXPDP

Gambar 3.6 diatas merupakan contoh dari penggunaan EXPDP untuk melakukan export terhadap beberapa *table* yang ingin di-*backup* oleh penulis.

Sebagaimana telah dipraktekan dalam mem-*backup database*, dari ketiga cara diatas cara yang paling mudah untuk digunakan dan efektif yaitu menggunakan *EXPDP* untuk memindahkan *table* kedalam *user* lain dan setelah itu cara paling baik dalam membagi *table* kedalam *table* tertentu yaitu menggunakan 1 *tablespace* 3 *datafile*. Kedua cara tersebut dianjurkan

untuk dilakukan tergantung situasi dan keperluan DBA untuk mem*backup table* dan *module*.

3.5.2. *Tuning Query Oracle*

Tuning Query Oracle dilakukan didalam 2 *database* yang berbeda yaitu *database server* 10.10.11.125 dan *database server* 10.10.11.153. Untuk melihat seberapa besar perubahan yang dialami pada saat mengubah *query* tersebut, penulis menggunakan Explain Plan untuk men-tracking hasil dan perubahan yang terjadi. Explain Plan merupakan fitur yang terdapat dalam *database Oracle* yang fungsinya melihat *cost* yang dibutuhkan oleh *query* untuk dapat berjalan. Terdapat beberapa *query* yang harus di *tuning* untuk mendapatkan performa yang lebih baik yaitu:

1. STD_V_SALES_DISTRIBUTOR (*SERVER* 10.10.11.125)

   STD_V_SALES_DISTRIBUTOR merupakan *view* yang digunakan untuk menampilkan sales pada periode tahun 2015 keatas, waktu execution time untuk menampilkan datanya yaitu 12-13 detik sebelum dilakukan *tuning*. *View* ini juga menggunakan *join* dengan 3 *view* lainya yaitu, STD_TEAM_PRODUCT, STD_V_PRODCODE, C_PERIOD. Jika *Query* tersebut dijalankan maka akan tampil hasil output *query* yang berisi 33 kolom dan 45364 baris data. Pada saat melakukan tuning, penulis disarankan agar tidak mengubah output baris pada *view* tersebut. Penulis melakukan beberapa *tuning query* yaitu dengan cara :

   a. Mengubah *Left Join* menjadi *Inner Join*

      Left join merupakan salah satu fungsi dalam sql yang digunakan untuk menampilkan isi *table* sisi kiri dari pengabungan beberapa *table* yang tidak berelasi dan *table* yang bernilai null, dengan kata lain penggunaan *left join* digunakan untuk mengambil seluruh data pada setiap *table* yang berbeda dan dapat mengambil nilai null dari sebuh *table*. Sedangkan *inner join* merupakan salah satu fungsi dalam sql yang digunakan untuk menampilkan data dari

pengabungan beberapa *table* yang datanya memiliki kesamaan atau berelasi. Secara teori penggunaan kedua fungsi ini sangat signifikan berbeda dalam memproses data, *left join* lebih cepat memproses data karena *left join* tidak membandingkan isi *table* yang satu dengan yang lainnya seperti *inner join*.

Gambar 3. 7 Perbedaan *inner join* dengan *left join*

Gambar 3.7 diatas merupakan contoh sederhana mengenai perbedaan antara *inner join* dan *left join*. Penulis melihat bahwa *query* yang biasa digunakan oleh perusahaan Dipa untuk mengabungkan *table* yaitu *left join*, sedangkan penggunaan *inner join* sangat jarang karena pengunaannya tergantung pada *table-table* yang digunakan. Penulis menemukan bahwa penggunaan *left join* lebih berat untuk memproses data dan lebih lambat pada saat melakukan explain plan pada *query view*

std_v_sales_distributor.

```
SQL> select * from std_v_sales_distributor;(SEMUA MASIH POLOS=LEFT JOIN SEMUA + DISTINCT)

45364 rows selected.

                                                                                LEFT JOIN
Execution Plan
----------------------------------------------------------
Plan hash value: 756886833

-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |   384 |   492K|       |   168K  (1)| 00:33:46 |
|*  1 |  HASH JOIN OUTER               |                     |   384 |   492K|       |   168K  (1)| 00:33:46 |
|*  2 |   HASH JOIN RIGHT OUTER        |                     |   384 |   432K|       |   168K  (1)| 00:33:44 |
|*  3 |    TABLE ACCESS FULL           | STD_TYPE            |     1 |    19 |       |     3   (0)| 00:00:01 |
|*  4 |    HASH JOIN RIGHT OUTER       |                     |   384 |   424K|       |   168K  (1)| 00:33:44 |
-------------------------------------------------------------------------------------------------

SQL> select * from std_v_sales_distributor;(UDAH DI INNER JOIN+DISTINCT)

45364 rows selected.
                                                                                INNER JOIN
Execution Plan
----------------------------------------------------------
Plan hash value: 3789275351

-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |   384 |   492K|       |   143K  (1)| 00:28:47 |
|*  1 |  HASH JOIN OUTER               |                     |   384 |   492K|       |   143K  (1)| 00:28:47 |
|*  2 |   HASH JOIN RIGHT OUTER        |                     |   384 |   432K|       |   143K  (1)| 00:28:46 |
|*  3 |    TABLE ACCESS FULL           | STD_TYPE            |     1 |    19 |       |     3   (0)| 00:00:01 |
|*  4 |    HASH JOIN RIGHT OUTER       |                     |   384 |   424K|       |   143K  (1)| 00:28:45 |
-------------------------------------------------------------------------------------------------
```

Gambar 3. 8 Explain plan std_v_sales_distributor

Dari gambar 3.8 diatas terlihat bahwa adanya pengurangan cost sebanyak 25 ribu dan pengurangan waktu proses sebanyak 5 detik hanya dengan mengubah *left join* menjadi *inner join* dan jumlah baris yang dihasilkan oleh kedua *query* sama. Penulis merubah beberapa *join left* mejadi *inner join* dalam *view* ini untuk melihat seberapa besar perubahan yang terjadi dalam *query* tersebut lewat explain plan. Penulis telah banyak melakukan percobaan perubahan dari *left join* menjadi *inner join* dan kadang perubahan tersebut mengubah isi baris menjadi lebih banyak maupun menjadi lebih sedikit, dan hal tersebut merubah hasil sesungguhnya struktur *table* dalam *database*. Pada akhirnya penulis mencoba bermacam komposisi penggunaan *inner join* pada *view* ini untuk dapat mendapatkan hasil yang sama dan jumlah baris yang sama dengan yang diharapkan. Akan tetapi penulis tidak mengetahui secara pasti apakah isi dari view tersebut sama seperti menggunakan *left join*, berdasarkan jumlah

baris yang dihasilkan sama jika menggunakan *inner join*.

b. Memberikan *Index*.

Index merupakan sebuah objek system *database* yang dapat mempercepat proses pencarian (*query*) data. Penggunaan *index* pada *database* merupakan salah satu teknik pembuatan atau pengimplementasian yang baik. Penggunaan *Index* sangat baik digunakan untuk proses pencarian data tetapi akan memperlambat proses input data kedalam *database*. Penulis mencoba membuat beberapa *index* didalam beberapa *table* untuk mengetahui efek yang diberikan oleh *index* ke dalam *database*.

```
Index yang ditambahin

(10.10.11.125)

CREATE INDEX COMPIERE.UPPER_FUNCT ON COMPIERE.STD_TYPE
(UPPER("VALUE"))
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
NOPARALLEL;

CREATE INDEX COMPIERE.STD_VERSION_PERIOD ON COMPIERE.STD_VERSION
(C_PERIOD_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
NOPARALLEL;

CREATE INDEX COMPIERE.STD_TYPE_VERSION_ID ON COMPIERE.STD_TYPE
(STD_VERSION_ID, VALUE)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
NOPARALLEL;

CREATE INDEX COMPIERE.STD_TEAM_TYPE ON COMPIERE.STD_TEAM
(STD_TYPE_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
NOPARALLEL;

CREATE INDEX COMPIERE.M_PRODUCT_IDX ON COMPIERE.M_PRODUCT
(M_PRODUCT_ID, ISACTIVE, NAME, VALUE, M_PRODUCT_CATEGORY_ID,
M_LOCATOR_ID, M_ATTRIBUTESET_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
NOPARALLEL;
```

Gambar 3. 9 Contoh penambahan *index*

Gambar 3.9 diatas menunjukan pembuatan beberapa *index* yang diimplementasikan oleh penulis untuk *database* perusahaan. Fungsi dari *index* ini yaitu menekan total cost dan waktu yang dibutuhkan *query* untuk mencari dan menampilkan data. Hasil yang didapat dengan menggunakan *index* terbilang cukup besar karena dengan adanya *index* akan membuat *table* yang diproses seluruh barisnya menjadi diproses sesuai dengan *index* yang sudah dibuat. Penambahan kelima *index* tidak lepas karena kelima

table yang digunakan memproses *table* secara keseluruhan. Dengan menambahkan index kedalam sebuah database maka proses DML (Data manipulation language) terutama *select* akan menjadi lebih cepat, akan tetapi proses *insert data* menjadi lebih lamban. Untuk mendapatkan hasil yang maksimal penulis hanya menambahkan index kedalam *table* yang sering dilakukan proses *select.*

Sebelum

Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
	384	492K		168K (1)	00:33:46
	384	492K		168K (1)	00:33:46
	384	432K		168K (1)	00:33:44
STD_TYPE	1	19		3 (0)	00:00:01
	384	424K		168K (1)	00:33:44
index$_join$_026	25	300		3 (0)	00:00:01
STD_VERSION_PK	25	300		1 (0)	00:00:01
STD_VERSION_PERIOD	25	300		1 (0)	00:00:
	384	420K		168K (1)	00:33:44
	384	128K		168K (1)	00:33:44
	384	128K		168K (1)	00:33:44
	381	116K		168K (1)	00:33:40
	44975	62M		168K (1)	00:33:40
STD_TEAM_PRODUCT	976	38064		25262 (2)	00:05:04
	976	289K		25262 (2)	00:05:04
M_PRODUCT	1488	113K		65 (4)	00:00:01
	4145	914K		25196 (2)	00:05:03
	4145	586K		25196 (2)	00:05:03
	4145	534K		8355 (1)	00:01:41
	4145	453K		36 (9)	00:00:01
	96	9408		17 (12)	00:00:01
	96	5280		10 (10)	00:00:01
	25	775		7 (15)	00:00:01
STD_VERSION	25	300		3 (0)	00:00:01
STD_TYPE	1	19		3 (0)	00:00:01
STD_TEAM	96	2304		3 (0)	00:00:01
C_ELEMENTVALUE	916	39388		6 (0)	00:00:01
STD_PRODUCT	3972	55608		19 (6)	00:00:01

Sesudah

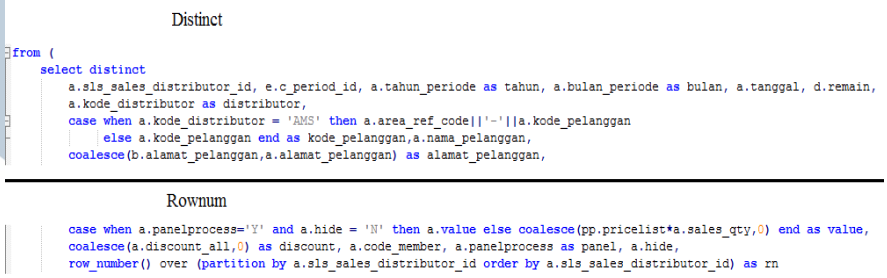
Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
	1	1312		5964 (7)	00:01:12
	1	1312		5964 (7)	00:01:12
	1	1152		5833 (6)	00:01:10
	1	31		4 (0)	00:00:01
index$_join$_023	1	19		3 (0)	00:00:01
STD_TYPE_PK	1	19		1 (0)	00:00:01
STD_TYPE_VERSION_ID	1	19		1 (0)	00:00:01
STD_VERSION	1	12		1 (0)	00:00:01
STD_VERSION_PK	1			0 (0)	00:00:01
	73	81833		5828 (6)	00:01:10
	73	96579		5828 (6)	00:01:10
	73	96579		5827 (6)	00:01:10
	72	93168		5683 (6)	00:01:09
	8467	11M		5681 (6)	00:01:09
	8467	11M		5667 (6)	00:01:08
STD_TEAM_PRODUCT	13	507		386 (18)	00:00:05
	13	2886		386 (18)	00:00:05
	13	2886		385 (18)	00:00:05
	57	8208		327 (20)	00:00:04
	57	7866		39 (13)	00:00:01
	57	5415		32 (13)	00:00:01
	57	4788		30 (14)	00:00:01
	57	3933		27 (12)	00:00:01
	57	3249		24 (9)	00:00:01
STD_TEAM	4	96		1 (0)	00:00:01
	1	43		4 (0)	00:00:01
index$_join$_036	1	19		3 (0)	00:00:01

Gambar 3. 10 Hasil *tuning index*

c. Mengubah Distinct Clause menjadi Rownum Clause.

Distinct clause merupakan salah satu fungsi didalam *database* untuk mengeliminasi data yang memiliki dupilkat dalam suatu *table*. Penggunaan distinct sangat mudah dan praktis dalam

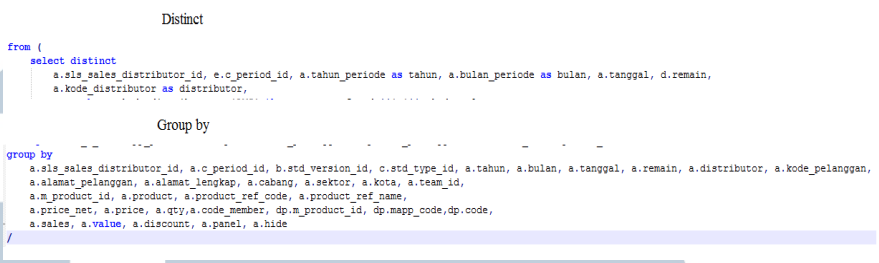
pengimplementasiannya dibandingkan dengan fungsi yang lain seperti *rownum clause*. Perusahaan untuk mengeleminasi duplikat data biasanya menggunakan *distinct clause* didalam *query*. Efek yang diberikan oleh fungsi ini yaitu membebankan kerja dari *database* untuk mengelola setiap *data* yang berada pada *table* yang akhirnya membuat kinerja *database* menjadi lambat. Distinct yang digunakan oleh *developer* perusahaan terbilang cukup boros karena setiap *table* yang diproses akan melakukan eliminasi data duplikat pada *database*, sedangkan dengan menggunakan *query rownum* penulis dapat memilih secara spesifik *table* yang akan dihapus data duplikatnya.



Gambar 3. 11 *Rownum clause*

d. Mengubah Distinct Clause menjadi Groupby Clause.

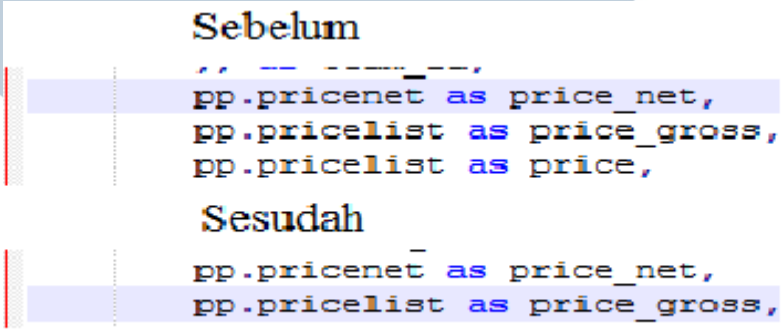
Salah satu alternative dari penggunaan Distinct Clause adalah menggunakan groupby clause, secara fungsi kedua clause tersebut sama, tetapi secara cost dan waktu yang dibutuhkan clause dalam memproses data berbeda. Perusahaan memilih distinct clause dibandingkan dengan groupby clause karena secara penulisan *query*, distinct lebih mudah digunakan karena kode sql yang dibutuhkan sedikit. Dalam penerapannya groupby clause memang lebih sulit untuk digunakan karena isi dari groupby clause harus sama dengan jumlah select yang digunakan dalam sebuah *query*.



Gambar 3. 12 *Groupby clause*

e. Menghilangkan *table* yang tidak diperlukan

Dalam melakukan tuning *query* dalam *database* server ini, penulis menemukan penggunaan *table* yang tidak baik karena *table* tersebut berisi data yang sama dengan salah satu *table* atau dengan kata lain yaitu adanya duplikasi *table*.



Gambar 3. 13 Penghapusan *table price*

Gambar 3.13 diatas merupakan potongan dari *query* yang digunakan dalam *view* std_v_sales_distributor, sesuai dengan gambar diatas *table* price dihilangkan karena isi *table* yang digunakan sama dengan *table* price_gross. Jika salah satu *table* tersebut tidak dihilangkan maka cost yang dibutuhkan oleh *database* akan semakin besar sesuai dengan pertumbuhan *table* tersebut. Setelah dihilangkannya *table* price maka jumlah cost yang dibutuhkan *database* untuk memproses *query view* ini berkurang.

f. Melakukan *tuning* pada *view* STD_TEAM_PRODUCT.

Std_team_product merupakan salah satu turunan yang digunakan oleh *view* std_v_sales_distributor. *View* ini tentunya sangat berpengaruh terhadap *view* sales karena kedua *view* ini di *join* oleh perusahaan untuk mendapatkan outpun yang di inginkan. Tuning yang dilakukan meliputi perubahan *left join* menjadi *inner join*, mengubah distinct clause menjadi groupby clause, dan menambahkan *index* kedalam *table* tertentu. Hasil yang didapatkan dengan adanya perubahan-perubahan diatas lewat explain plan tidak begitu signifikan tetapi tetap baik karena performa yang diberikan menjadi lebih baik. Gambar 3.14 dibawah ini merupakan hasil dari explain plan dari tuning pada *view* std_team_product yang telah mengubah proses select data dari 1000 lebih baris menjadi kurang dari 20 baris dan memakan lebih sedikit *cost per byte*.

Sebelum

```
Execution Plan
----------------------------------------------------------
Plan hash value: 795815700

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                     |  1037 |   288K|   115   (8)| 00:00:02 |
|   1 |  VIEW                    |                     |  1037 |   288K|   115   (8)| 00:00:02 |
|   2 |   SORT UNIQUE            |                     |  1037 |   218K|   115   (8)| 00:00:02 |
|*  3 |    HASH JOIN             |                     |  1037 |   218K|   114   (8)| 00:00:02 |
```

Sesudah

```
Execution Plan
----------------------------------------------------------
Plan hash value: 2855782772

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                     |    15 |  3420 |   100   (5)| 00:00:02 |
|   1 |  SORT GROUP BY           |                     |    15 |  3420 |   100   (5)| 00:00:02 |
|   2 |   NESTED LOOPS           |                     |    15 |  3420 |    99   (5)| 00:00:02 |
|*  3 |    HASH JOIN OUTER       |                     |    63 |  9450 |    36  (12)| 00:00:01 |
```

Gambar 3. 14 Hasil explain plan std_team_product

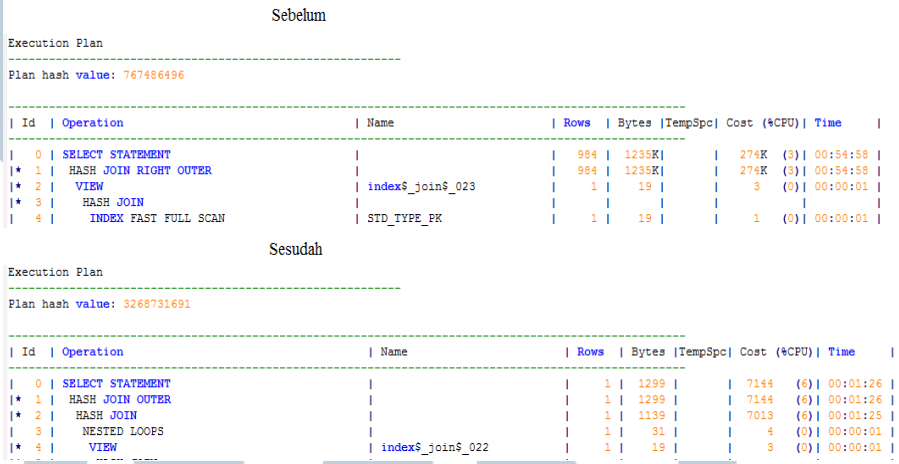
2. STD_V_SALES_PROGRESS (*SERVER* 10.10.11.125).

Sama halnya dengan STD_V_SALES_DISTRIBUTOR, *query* ini hampir sama dengan *View* STD_V_SALES_PROGRESS, hanya berbeda dari periode tahunnya dimana *View* ini memiliki tahun

periode 2014 ketas dengan execution time 29-30 detik sebelum dilakukan *tuning*. Penulis melakukan beberapa *tuning* yaitu dengan cara :

a. Mengubah *Left Join* menjadi *Inner Join*

Std_v_sales_progress tidak jauh berbeda dengan std_v_sales_distributor, hanya ada beberapa *join* dan filter tahun yang berbeda, tetapi secara keseluruhan dan strukturnya sama. *Inner join* membantu *database* untuk mengurangi beban *database* dalam memproses seluruh *table* yang di-*select* dalam *query*. Gambar 3.15 dibawah ini merupakan hasil dari explain plan perubahan yang terjadi dengan mengubah *inner join*.



Gambar 3. 15 Explain plan std_v_sales_progress

b. Memberikan *Index*

Index yang ditambahkan sama dengan std_v_sales_distributor karena masih menggunakan *database* dan *table* yang sama, sehingga efek yang diberikan oleh *index* yang telah dibuat akan sama dengan *view* std_v_sales_distributor. Dengan demikian penulis tidak perlu lagi menambahkan *index* yang tidak diperlukan karena tidak semua *index* dapat mempercepat proses pencarian data, terkadang *index* dapat membebankan *database* pada saat mencari data sehingga membuat performa *database* menjadi lebih buruk.

c. Mengubah Distinct Clause menjadi Rownum Clause

   Pengguna distinct clause di ubah menjadi rownum clause untuk memperbaiki performa *database* dalam memproses data sama seperti view std_v_sales_distributor, hanya saja jumlah data sales_progress lebih banyak yang akhirnya execution timenya bertambah. Walau jumlah datanya bertambah, penggunaan rownum tetap baik di implementasikan kedalam sebuah view karena lebih stabil pada saat diakses oleh *user*.

d. Menghilangkan *table* yang tidak diperlukan

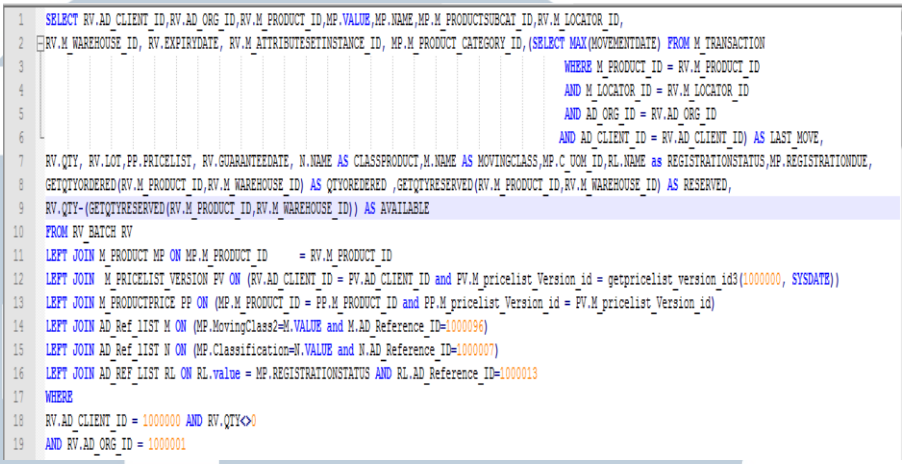
   Sama seperti *view* std_v_sales_distributor, ada 1 *table* yang dihapus yaitu *table price* untuk memperbaiki performa *database*. Dengan demikian *database* tidak perlu melakukan akses kedalam *table* yang sama dan hanya menggunakan 1 table agar meningkatkan efisiensi dalam memproses sebuah table.

e. Melakukan *Tuning* pada *view* STD_TEAM_PRODUCT

   Perubahan yang terjadi STD_TEAM_PRODUCT pada saat melakukan tuning *view* std_v_sales_distributor memberikan dampak yang sama kepada *view* std_v_sales_distributor. Dampak yang diberikan tidaklah signifikan jika dilihat dari explain plan, tetapi sekecil apapun perubahan yang terjadi akan sangat berpengaruh terhadap performa *database* itu sendiri.

Setelah melakukan *tuning* dengan cara-cara yang disebutkan diatas, execution time *view* std_v_sales_progress berkurang menjadi 22-23 detik, hal ini dapat terjadi karena penulis berhasil mencari fungsi alternatif dari distinct clause dimana fungsi ini memakan banyak sekali waktu dan cost didalam *database*.

3. *QUERY* (*SERVER* 10.10.11.153)



```
SELECT RV.AD_CLIENT_ID,RV.AD_ORG_ID,RV.M_PRODUCT_ID,MP.VALUE,MP.NAME,MP.M_PRODUCTSUBCAT_ID,RV.M_LOCATOR_ID,
RV.M_WAREHOUSE_ID, RV.EXPIRYDATE, RV.M_ATTRIBUTESETINSTANCE_ID, MP.M_PRODUCT_CATEGORY_ID,(SELECT MAX(MOVEMENTDATE) FROM M_TRANSACTION
                                                                                            WHERE M_PRODUCT_ID = RV.M_PRODUCT_ID
                                                                                            AND M_LOCATOR_ID = RV.M_LOCATOR_ID
                                                                                            AND AD_ORG_ID = RV.AD_ORG_ID
                                                                                           AND AD_CLIENT_ID = RV.AD_CLIENT_ID) AS LAST_MOVE,
RV.QTY, RV.LOT,PP.PRICELIST, RV.GUARANTEEDATE, N.NAME AS CLASSPRODUCT,M.NAME AS MOVINGCLASS,MP.C_UOM_ID,RL.NAME as REGISTRATIONSTATUS,MP.REGISTRATIONDUE,
GETQTYORDERED(RV.M_PRODUCT_ID,RV.M_WAREHOUSE_ID) AS QTYOREDERED ,GETQTYRESERVED(RV.M_PRODUCT_ID,RV.M_WAREHOUSE_ID) AS RESERVED,
RV.QTY-(GETQTYRESERVED(RV.M_PRODUCT_ID,RV.M_WAREHOUSE_ID)) AS AVAILABLE
FROM RV_BATCH RV
LEFT JOIN M_PRODUCT MP ON MP.M_PRODUCT_ID      = RV.M_PRODUCT_ID
LEFT JOIN  M_PRICELIST_VERSION PV ON (RV.AD_CLIENT_ID = PV.AD_CLIENT_ID and PV.M_pricelist_Version_id = getpricelist_version_id3(1000000, SYSDATE))
LEFT JOIN M_PRODUCTPRICE PP ON (MP.M_PRODUCT_ID = PP.M_PRODUCT_ID and PP.M_pricelist_Version_id = PV.M_pricelist_Version_id)
LEFT JOIN AD_Ref_lIST M ON (MP.MovingClass2=M.VALUE and M.AD_Reference_ID=1000096)
LEFT JOIN AD_Ref_lIST N ON (MP.Classification=N.VALUE and N.AD_Reference_ID=1000007)
LEFT JOIN AD_REF_LIST RL ON RL.value = MP.REGISTRATIONSTATUS AND RL.AD_Reference_ID=1000013
WHERE
RV.AD_CLIENT_ID = 1000000 AND RV.QTY<>0
AND RV.AD_ORG_ID = 1000001
```

Gambar 3. 16 Query pada server 10.10.11.153

Gambar 3.16 diatas berisikan *query* yang digunakan pada *server* 10.10.11.153. *Query* tersebut memiliki execution time 2-3 detik sebelum dilakukan *tuning*. Penulis melakukan *tuning* dengan beberapa cara yaitu:

a. Filter *where clause*

Where clause merupakan salah satu fungsi filter data dalam *database*, hasil yang di inginkan oleh perusahaan dapat disesuaikan dengan filter yang akan dibuat. Fugsi where clause sebenarnya akan membuat *database* bekerja lebih keras untuk melakukan seleksi data, akan tetapi beberapa fungsi filter akan memperbaiki performa *database* sesuai dengan penggunaan *query* dalam *database*.

Sebelum

```
16  WHERE
17  RV.AD_CLIENT_ID = 1000000
18  RV.QTY<>0
19  AND RV.AD_ORG_ID = 1000001
20  /
```

Sesudah

```
16  WHERE
17  RV.QTY<>0
18  AND RV.AD_ORG_ID = 1000001
19  /
```

Gambar 3. 17 Mengurangi *filter where*

Gambar 3.17 diatas merupakan fungsi filter yang dihilangkan karena filter where tersebut dipindahkan kedalam *view* rv_batch dimana dengan memindahkan where clause tersebut akan mengurangi beban dari *database* dalam memproses data yang di-filter dan menghasilkan output data yang sama dengan performa yang lebih baik.

b. *Tuning view* RV_batch

Didalam *query* server 10.10.11.153 terdapat satu *view* yang merupakan bagian dari *query* utama (*query* server 10.10.11.153). Penulis melakukan beberapa perubahan seperti mengubah fungsi *left join* menjadi *inner join*, menambahkan where clause untuk men-filter data, dan menghapus fungsi orderby clause. Perubahan yang terjadi membuat *view* RV_batch lebih cepat memproses data dari 3-2 detik menjadi 1 detik.

Sebelum

```
SELECT MT.AD_CLIENT_ID,MT.AD_ORG_ID,MA.M_ATTRIBUTESETINSTANCE_ID,MA.LOT,CASE WHEN MA.EXPIRYDATE IS NULL THEN TO_DATE('01/01/3000','DD/MM/YYYY')
      SUM(MT.MOVEMENTQTY) AS QTY,MO.M_LOCATOR_ID,MO.VALUE,MT.M_PRODUCT_ID,M_WAREHOUSE_ID,MA.GUARANTEEDATE
FROM M_TRANSACTION MT
LEFT JOIN M_LOCATOR             MO ON MO.M_LOCATOR_ID = MT.M_LOCATOR_ID
LEFT JOIN M_ATTRIBUTESETINSTANCE MA ON MA.M_ATTRIBUTESETINSTANCE_ID = MT.M_ATTRIBUTESETINSTANCE_ID
LEFT JOIN M_WAREHOUSE            MW ON MW.M_WAREHOUSE_ID = MO.M_WAREHOUSE_ID
GROUP BY MA.M_ATTRIBUTESETINSTANCE_ID,MA.LOT,MA.EXPIRYDATE,MO.M_LOCATOR_ID,MO.VALUE,MT.AD_CLIENT_ID,MT.AD_ORG_ID,MT.M_PRODUCT_ID,M_WAREHOUSE_ID
ORDER BY MA.M_ATTRIBUTESETINSTANCE_ID ASC
```

Sesudah

```
SELECT MT.AD_CLIENT_ID,MT.AD_ORG_ID,MA.M_ATTRIBUTESETINSTANCE_ID,MA.LOT,CASE WHEN MA.EXPIRYDATE IS NULL THEN TO_DATE('01/01/3000'
    SUM(MT.MOVEMENTQTY) AS QTY,MO.M_LOCATOR_ID,MO.VALUE,MT.M_PRODUCT_ID,M_WAREHOUSE_ID,MA.GUARANTEEDATE    FROM M_TRANSACTION MT
INNER JOIN M_LOCATOR             MO ON MO.M_LOCATOR_ID = MT.M_LOCATOR_ID
INNER JOIN M_ATTRIBUTESETINSTANCE MA ON MA.M_ATTRIBUTESETINSTANCE_ID = MT.M_ATTRIBUTESETINSTANCE_ID
INNER JOIN M_WAREHOUSE            MW ON MW.M_WAREHOUSE_ID = MO.M_WAREHOUSE_ID
WHERE MT.AD_CLIENT_ID = 1000000 and MT.AD_ORG_ID=1000001
GROUP BY MA.M_ATTRIBUTESETINSTANCE_ID,MA.LOT,MA.EXPIRYDATE,MO.M_LOCATOR_ID,MO.VALUE,MT.AD_CLIENT_ID,MT.AD_ORG_ID,MT.M_PRODUCT_ID,
```

Gambar 3. 18 *Tuning view rv_batch*

c. Mengubah *Left Join* menjadi *Inner Join*

Penulis seperti biasanya mengubah fungsi *left join* menjadi *inner join* untuk mengurangi beban *database* dalam memproses data seperti yang dilakukan dalam view std_v_sales_distributor maupun view std_v_sales_progress.

d. Memberikan *Index*

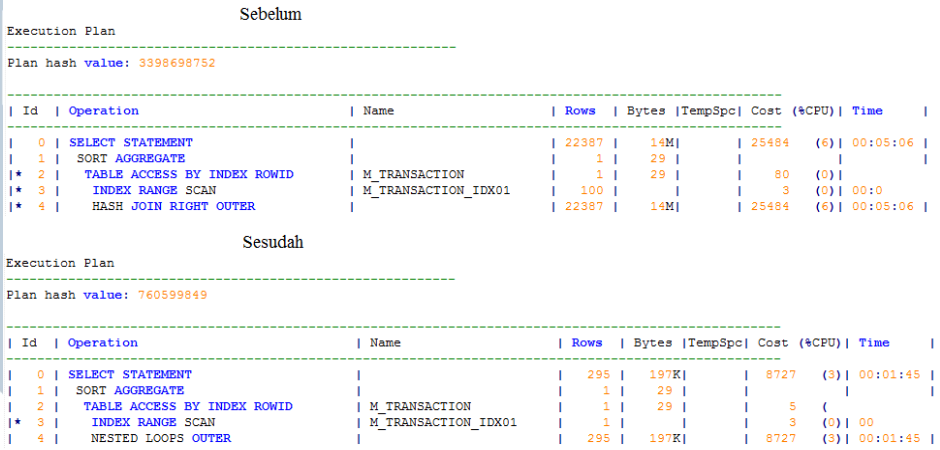
Penulis menambahkan *index* untuk mempercepat proses data dalam sebuah *table*. *Index* ini digunakan untuk mengurangi cost *database* dalam memproses seluruh baris data menjadi memproses menggunakan *index* yang sudah dibuat.

```
(10.10.11.153/5)


CREATE INDEX COMPIERE.M_TRANSACTION_IDX ON COMPIERE.M_TRANSACTION
(M_ATTRIBUTESETINSTANCE_ID, AD_CLIENT_ID, AD_ORG_ID, M_LOCATOR_ID, MOVEMENTQTY,
M_PRODUCT_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
NOPARALLEL;
```

Gambar 3. 19 Penambahan *Index* 10.10.11.153

Gambar 3.19 diatas merupakan penambahan *index* yang dilakukan oleh penulis untuk mempercepat performa dari *database* mengakses setiap data pada table tertentu.

Dalam *server* 10.10.11.153 terjadi *heavy traffic user*, dimana *database* server ini sedang ramai digunakan oleh berbagai macam divisi, sehingga waktu execution time menjadi tidak dapat diketahui secara pasti. Akan tetapi, penulis dapat mengetahui dapat mengetaui perubahan cost yang terjadi dengan bantuan *explain plan*, dan hasil yang didapatkan sesuai dengan gambar 3.20 dibawah ini.



Sebelum

```
Execution Plan
----------------------------------------------------------
Plan hash value: 3398698752

-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     | 22387 |    14M|       | 25484   (6)| 00:05:06 |
|   1 |  SORT AGGREGATE                |                     |     1 |    29 |       |            |          |
|*  2 |   TABLE ACCESS BY INDEX ROWID  | M_TRANSACTION       |     1 |    29 |       |    80   (0)|          |
|*  3 |    INDEX RANGE SCAN            | M_TRANSACTION_IDX01 |   100 |       |       |     3   (0)| 00:0
|*  4 |    HASH JOIN RIGHT OUTER       |                     | 22387 |    14M|       | 25484   (6)| 00:05:06 |
```

Sesudah

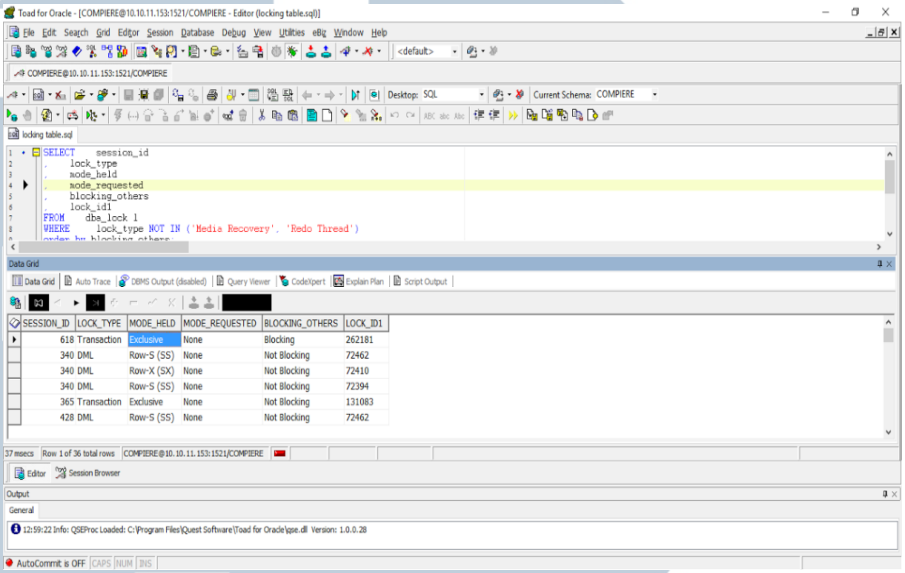
```
Execution Plan
----------------------------------------------------------
Plan hash value: 760599849

-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |   295 |   197K|       |  8727   (3)| 00:01:45 |
|   1 |  SORT AGGREGATE                |                     |     1 |    29 |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID  | M_TRANSACTION       |     1 |    29 |       |     5   (
|*  3 |    INDEX RANGE SCAN            | M_TRANSACTION_IDX01 |     1 |       |       |     3   (0)| 00
|   4 |    NESTED LOOPS OUTER          |                     |   295 |   197K|       |  8727   (3)| 00:01:45 |
```

Gambar 3. 20 *Explain plan* 10.10.11.153

3.5.3 Locking Table

Istilah *lock* pada *database* berarti penguncian, *lock* digunakan pada saat terjadi pengaksesan *database* oleh *user* secara bersamaan. Hal ini ditujukan untuk menjaga data agar tidak *corrupt* ataupun hilang pada saat *user* mengakses *database* secara bersamaan. *Locking table* merupakan permasalahan yang terjadi karena salah satu *table* atau lebih tidak dapat diakses oleh *user* akibat dari jumlah *user* dalam suatu *session* sangat banyak. Permasalahan *locking table* ini sering terjadi pada saat akhir bulan dimana terjadi penutupan penjualan, dimana seluruh divisi akuntan melakukan akses kedalam *database*. Untuk saat ini penulis belum menemukan cara untuk mencegah hal itu terjadi, tetapi jika hal tersebut sudah terlanjur terjadi maka yang harus dilakukan yaitu *kill session*. *Kill session* memiliki fungsi untuk membuat *table* dapat kembali diakses oleh *user*, tetapi *session*

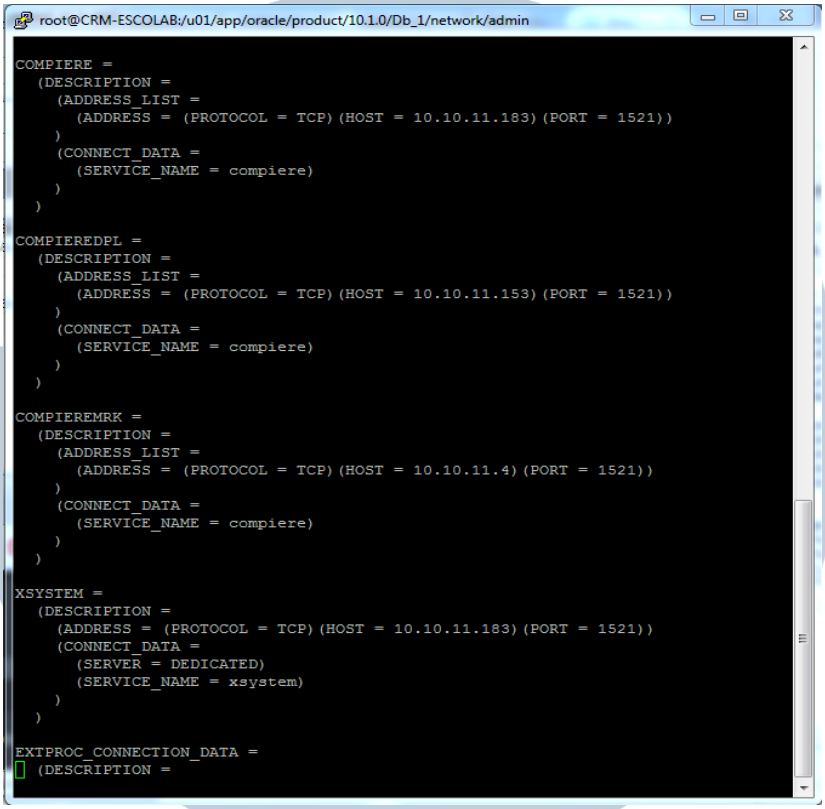
tersebut akan dilakukan *rollback* sehingga data tersebut tidak masuk kedalam *database*.



Gambar 3. 21 Contoh *Locking Table*

3.6. Pekerjaan Tambahan

PT Dipa melakukan *cloning Database server* 10.10.11.185 yang digunakan untuk simulasi *database*, penulis mendapatkan tugas untuk membuat konfigurasi koneksi ke-*database* baru karena pada saat *database* dibentuk belum ada konfigurasi koneksi. Penulis menggunakan aplikasi Putty untuk masuk kedalam *database server* 10.10.11.185 untuk menkonfigurasi agar *user* perusahaan dapat login. Gambar dibawah 3.22 ini merupakan tampilan untuk melakukan konfigurasi listener *database* agar dapat melakukan koneksi ke dalam *database* dengan menggunakan putty.



Gambar 3. 22 *Configure Listener*

3.7. Kendala Yang Dihadapi

Kendala-kendala yang dihadapi penulis pada saat pelaksanaan kerja magang antara lain:

1. Penulis belum belajar mengenai *database Oracle* karena jurusan yang diambil oleh penulis yaitu *Cisco Networking* sehingga penulis harus belajar mengenai *Oracle* sebelum mengerjakan *project* yang diberikan.
2. Materi kuliah mengenai *Oracle Database* yang didapatkan dari universitas belum menyentuh pada level *tuning* sehingga penulis mendapat kesulitan dalam menyelesaikan *project* tersebut.
3. Didalam PT Dipa Pharmalab Intersains sendiri belum memiliki seorang *Database Administrator* (DBA) sehingga penulis harus mempelajari gtgsendiri struktur dan kegunaan *query* dan data

didalam *database Oracle* perusahaan.

4. Minimnya resource mengenai *tuning database Oracle* dan pengaplikasiannya terhadap *query* maupun *database*, sesuai dengan permasalahan yang dihadapi oleh penulis didalam buku maupun internet.

3.8. Solusi Atas Kendala

Solusi atas masalah yang dihadapi oleh penulis dalam melakukan pelaksanaan kerja magang antara lain:

1. Mencari buku panduan mengenai *tuning database Oracle* dari perpustakaan Universitas Multimedia Nusantara untuk membantu penulis mempelajari struktur *database oracle*.
2. Melakukan konsultasi mengenai *database oracle* dengan dosen fakultas sistem informasi Universitas Multimedia Nusantara dan *database administrator* lain melalui forum internasional di internet.
3. Mencari referensi dari internet mengenai *tuning query database* lewat forum oracle yaitu orafaq.com.