ISSN : 2407 - 3911

SEGMENTASI KOLEKSI DATA UNTUK MENDUKUNG PERFORMANSI SISTEM

Aradea
Teknik Informatika Fakultas Teknik
Universitas Siliwangi Tasikmalaya
Jl. Siliwangi no. 24 Tasikmalaya
aradea@unsil.ac.id

Iping Supriana, Kridanto Surendro
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha no. 10 Bandung
iping@informatika.org, endro@informatika.org

Abstrak

Pengelolaan koleksi data merupakan hal yang sangat kritis, apalagi jika melibatkan volume data yang sangat besar, karena hal ini akan berpengaruh terhadap performansi sistem secara menyeluruh. Suatu rancangan strategi khusus dibutuhkan bagi penyediaan performansi sistem, sehingga sistem mampu mengerjakan tugas-tugasnya sesuai dengan indikator sistem. Salah satu indikator paling umum yang dapat digunakan untuk menilai performansi sistem, sebut saja misalnya kecepatan. Kecepatan ini pada dasarnya berhubungan dengan teknik tuning, dan saat ini terdapat ragam tipe, teori, maupun metodologi yang dapat digunakan dalam teknik tuning ini. Makalah ini akan menguraikan salah satu persoalan yang berhubungan dengan *application tuning*, terutama beberapa hal yang berhubungan dengan kebutuhan penggunaan segmentasi atau partisi untuk mengelompokan koleksi data yang besar. Pembahasan diawali dengan latar belakang, dilanjutkan dengan teori-teori terkait, kemudian mendefinisikan ilustrasi kasus, analisis dan pembahasan kasus, proses implementasi dan pengujian, serta kesimpulan dan pekerjaan kedepan.

Kata kunci :

Segmentasi data, partisi data, *application tuning*, performansi database

Abstract

Management of data collection is one of critical matter, event if involving a large volume of data because its will affect to all system performance. a particular strategic planning is needed to provision system performance, then the system able to do its jobs suitable with system indicator. one of very general indicator that can be used to assess system performance is speed. Basically, speed connected with tuning technique, and at the current time, there are many types, both theory or methodology that can be used in tuning technique. This paper will depict one problem that related to tuning application, especially some item that related with the requirement of using segmentation or partition to grouping large data collection. Depiction started with a background, then continued with related theories, case illustration, analyze and case description, implementation process, testing, conclusion and future works.

Keywords:

Data segmentation, data partition, application tuning, database performance

I. PENDAHULUAN

Pada era informasi saat ini, pertumbuhan data yang tersebar pada berbagai koleksi data terjadi sangat cepat, bahkan muncul suatu istilah yang disebut sebagai banjir data, banjir informasi, *big data*, atau istilah-istilah sejenis lainnya. Hal ini merupakan tuntutan dari para pengguna dan pemilik data saat menjalankan berbagai aktivitasnya. Bisa kita bayangkan beberapa perusahaan yang bekerja sama, memiliki berbagai unit kerja dengan fungsinya masing-masing, dan didukung oleh lebih dari satu proses bisnis, sehingga pada setiap harinya dapat terjadi proses transaksi yang sangat banyak dan terjadi secara bersamaan. Kondisi ini menyebabkan jumlah data yang tumbuh dalam perusahaan tersebut menjadi sangat besar, selain itu kebutuhan data harus aktif dan tersedia selama 24 jam, serta dapat terjamin keamanan dan kecepatan prosesnya. Ketika suatu koleksi data yang bekerja selama 24 jam tersebut melakukan proses transaksi yang sangat banyak secara bersamaan, hal ini dapat menimbukan

persoalan, karena bagaimana suatu sistem dapat menjamin pengelolaan sejumlah data dalam suatu perusahaan yang setiap harinya mengalami pertumbuhan yang sangat besar.

Salah satu teknik yang dapat digunakan untuk meningkatkan performansi sistem ketika melakukan *query* terhadap suatu koleksi data adalah dengan teknik *indexing*. *Indexing* ini merupakan penggunaan *index* yang tepat untuk mengakses data, misalnya ketika suatu koleksi data dengan jumlah jutaan *data record* diakses tanpa menggunakan *indexing* pada tabel-tabelnya, maka performansi sistem akan sangat menurun, karena sumber daya sistem akan banyak digunakan untuk pencarian data dengan metode *table-scan* atau disebut juga dengan istilah *heaps*. Artinya pencarian data dilakukan per *data record* dari awal hingga data yang dicari ditemukan, selain itu perlu diingat juga sistem penyimpanan koleksi data dalam suatu *database management system* (DBMS) dapat menggunakan konsep *pages* yang disimpan pada sebuah *extent* dengan urutan data yang acak. Namun jika tabel-tabel dalam koleksi data tersebut ter-*index* maka performansi sistem akan membaik, karena data pada koleksi data akan terurut secara *logical* pada tingkat *pages* dan *extent*, hal inilah yang membantu mempercepat pengaksesan data.

Namun ketika jumlah data dalam koleksi data terus tumbuh atau bertambah, misalnya mencapai puluhan juta bahkan mencapai milyaran *data record*, maka teknik *indexing* ini juga akan menurun performansinya, karena setiap tabel koleksi data hanya memiliki satu segmen dengan jumlah *data record* yang sangat banyak. Dalam kasus ini, untuk meningkatkan kembali performansi sistem, solusi segmentasi data dapat diterapkan, yaitu memecah tabel kedalam beberapa segmen/ partisi atau subpartisi pada setiap tabel yang dibutuhkan, sehingga akses data dapat diarahkan hanya untuk segmen-segmen yang diperlukan saja, tentu saja hal ini dapat meningkatkan efektifitas dan kecepatan akses data. Bahasan teori terkait segmentasi data dan performansi sistem akan diurakan pada bagian II, sementara bagian III mendefinisikan kebutuhan ilustrasi kasus, bagian IV membahas hasil dari implementasi sistem, dan bagian V menyimpulkan hasil keseluruhan pembahasan.

II. KAJIAN LITERATUR

II.1 Performansi Sistem

Berdasarkan definisi salah satu literatur, performansi adalah tingkat kemampuan suatu mesin atau sistem (Soanes, 2008). Artinya disini bahwa performansi berhubungan dengan tingkat kemampuan suatu sistem dalam mengerjakan suatu tugas tertentu yang didelegasikan kepada sistem tersebut. Apabila definisi tersebut dipetakan kedalam kebutuhan sistem untuk pengelolaan suatu koleksi data atau *database*, maka akan berkaitan dengan istilah *tuning*. *Tuning* merupakan tindakan mengidentifikasikan hal-hal yang menyebabkan permasalahan pada performansi dan membuat perubahan yang diperlukan untuk mengurangi dampak dari permasalahan tersebut (Chan, 2010). Atau jika merujuk dari definisi lainnya, *tuning* adalah tindakan *reconfigure* atau modifikasi suatu sistem dengan tujuan untuk meningkatkan performansi (Whalen, 2004). Bila ditinjau dari sisi manfaat dan keuntungan, melalui *tuning* ini kita dapat meminimalisasi penambahan *hardware*, memungkinkan untuk menurunkan spesifikasi *hardware* minimum yang dibutuhkan, mengurangi biaya perawatan, sistem dapat menghasilkan *response time* yang cepat sehingga dapat meningkatkan staff morale dan kepuasan pelanggan, menghasilkan *throughput* yang lebih baik dibandingkan sebelumnya, serta mengoptimalkan penggunaan *hardware* (Connolly, 2002). Dengan demikian, jika kita tinjau karakteristik dari *tuning* ini dapat kita simpulkan, yaitu berbagai upaya yang dapat dilakukan untuk meningkatkan performansi sistem. Hal ini selaras dengan definisi tipe *tuning* berdasarkan tujuannya (Whalen,2004)

1. *Tuning for response time*, yaitu *tuning* untuk waktu respon, melibatkan perubahan dari komponen-komponen sistem untuk mengurangi keterlambatan (*latency*) pada transaksi dan mempercepat pemrosesan transaksi. Umumnya *tuning* untuk waktu respon dilakukan dengan *tuning* aplikasinya, menginstal *hardware* yang lebih cepat, atau melakukan *tuning* terhadap *instance* DBMS. Waktu respon juga bisa ditingkatkan dengan melakukan *tuning* skema *database*, misalnya dengan *index tuning*, *partitioning*, dan lain-lain.
2. *Tuning for throughput*, yaitu *tuning* untuk *throughput*, melibatkan perubahan sistem yang

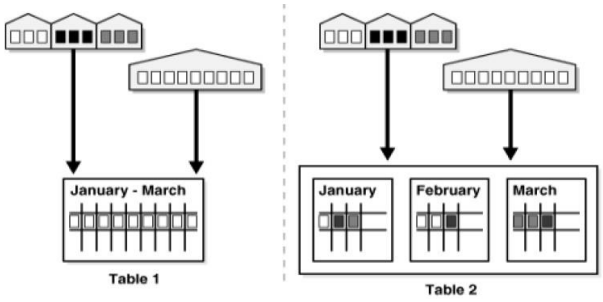
memungkinkan penyelesaian pekerjaaan yang lebih banyak, tanpa perlu membuat proses/ *query* individual berjalan lebih cepat. *Throughput* yang meningkat, memungkinkan beberapa *query* dapat menyelesaikan lebih banyak pekerjaan tanpa membuat salah satu *query* tersebut berjalan lebih cepat. Misalnya, jika sebuah *query* memerlukan waktu 10 detik, dengan melakukan *tuning* terhadap sistemnya kita dapat menjalankan dua buah *query* dalam 10 detik, dengan demikian kita telah meningkatkan *throughput* dari sistem tanpa meningkatkan waktu respon dari tiap *query*.

Pada intinya *perfomance tuning* ini merupakan suatu tindakan untuk meningkatkan kinerja dari suatu sistem dengan merubah sistem parameter (*tuning* perangkat lunak) atau memperbaiki konfigurasi sistem (*tuning* perangkat keras). Sementara itu *tuning* yang diterapkan khusus pada aplikasi atau lebih dikenal dengan *application tuning* adalah suatu tindakan yang mayoritas berhubungan dengan perbaikan di sisi sintaks SQL, hal ini melibatkan proses analisis sintaks SQL dan untuk memastikan apakah sintaks *query* yang digunakan telah efisien dan optimal (Connolly, 2002). Terdapat beberapa hal yang perlu diperhatikan dalam melakukan *application tuning* (Oracle), yaitu :

1. *SQL Tuning* (mengoptimalkan *code* yang digunakan)
2. *Indexing* (penggunaan *index* yang tepat untuk mengakses data)
3. *Partitioning/ Segmentation* (penggunaan partisi untuk mengelompokkan data besar)

Dalam pembahasan makalah ini, kita akan fokus pada partisi atau segmentasi, sebagai salah satu hal penting dalam melakukan *application tuning*. Menurut (Cyran, 2003), partisi atau segmentasi adalah suatu metode untuk membantu pembagian tabel yang memiliki data sangat banyak, menjadi bagian yang lebih kecil yang disebut partisi, sehingga memudahkan untuk pengaturan (*manageable*). Setelah di partisi, *DDL statements* dapat mengakses atau memanipulasi data per bagian tanpa perlu mengakses data pada tabel secara keseluruhan. Gambar 1 mengilustrasikan perbandingan koleksi data yang tidak tersegmentasi (tabel 1) dan yang tersegmentasi (tabel 2).

Melalui segmentasi ini pengelolaan historis data yang besar dapat dilakukan dengan mudah, misalnya untuk kebutuhan *data warehouses* yang mengandung satu atau lebih tabel fakta yang besar, tabel fakta tersebut dapat dipartisi berdasarkan tanggal atau tipe data tertentu, sehingga dapat menjadi dokumentasi historis data perusahaan (Fogel, 2015). Selain itu segmentasi juga memungkinkan realisasi untuk kebutuhan sistem terdistribusi, yaitu menggunakan informasi partisi *tablespec* untuk mendistribusikan *rows* dari *query servers* terhadap *load servers*, metode distribusi dapat digunakan untuk menggabungkan antara operasi permintaan dan operasi beban, misalnya ketika jumlah partisi yang dimuat lebih besar atau sama dengan jumlah *load servers*, dan input data akan terjadi secara merata diseluruh partisi (Lorentz, 2014).



Gambar 1. Ilustrasi perbandingan data tidak tersegmen dan tersegmen

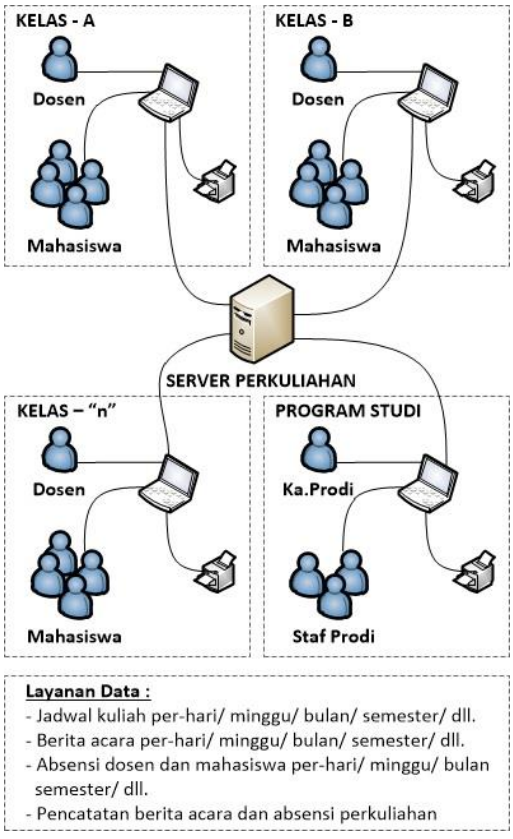
Secara umum manfaat dari penerapan segmentasi ini adalah (a) *faster performance*, yaitu menurunkan waktu query dari menit ke detik, (b) *increases availability*, yaitu 24 jam dalam 7 hari untuk mengakses informasi penting, (c) *improves manageability*, yaitu mengelola '*chunks*' data yang lebih kecil, (d) *enables information lifecycle management*, yaitu penggunaan biaya yang efisien dalam proses penyimpanan. Sementara beberapa ketentuan umum yang dapat dijadikan pedoman saat menentukan kapan tabel harus dipartisi diantaranya: sebuah tabel dapat berisi hingga 64.000 partisi, data bertipe RAW dan LONGRAW tidak dapat dipartisi, setiap partisi yang terbentuk harus memiliki *logical attribute* yang sama, misalnya nama kolom, tipe data dan *constraint*, data yang terkandung dalam tabel tersebut sudah mencapai atau lebih dari 2 GB, dan tabel berisi *historical data*.

III. ILUSTRASI KASUS

Berikut merupakan ilustrasi kasus untuk kebutuhan penerapan segmentasi data di lingkungan perguruan tinggi. Kasus ini merupakan kelanjutan

dari penelitian kami sebelumnya (Aradea dkk., 2014), dimana dari keseluruhan arsitektur informasi yang telah dikembangkan, diambil salah satu sub arsitektur data perkuliahan untuk kebutuhan performansinya.

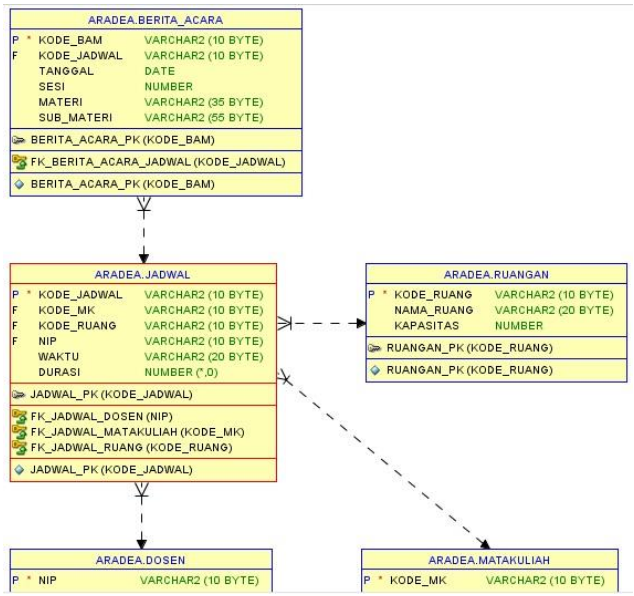
Skenario aplikasi difokuskan untuk memenuhi kebutuhan sistem perkuliahan, dimana penyelenggaraan kuliah di setiap kelas akan didukung oleh fasilitas untuk melakukan pengisian berita acara dan absensi. Selain itu, setiap program studi dapat memonitor penyelenggaraan perkuliahan tersebut, serta mencetak kebutuhan laporan-laporannya, seperti ditunjukan pada Gambar 2. Diasumsikan perguruan tinggi tersebut memiliki jumlah program studi dan mahasiswa yang cukup banyak, sehingga proses *query* data yang terjadi pada saat penyelenggaraan kuliah dapat mencapai ribuan bahkan puluhan ribu *data record* disetiap harinya, belum termasuk jika ditambah kebutuhan dari setiap program studi pada saat melakukan pelaporan per-hari, per-minggu, per-bulan, per-semester, per-tahun akademik dan kebutuhan *query* data lainnya.



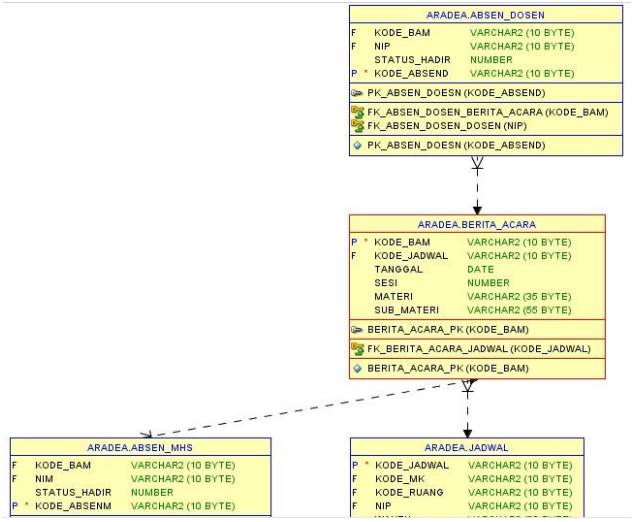
Gambar 2. Ilustrasi layanan akses data

III.1 Pemodelan Data dan Sistem

Skema relasi dari setiap koleksi data untuk memenuhi kebutuhan perkuliahan tersebut dapat dilihat pada Gambar 3 dan 4. Tabel jadwal kuliah yang merujuk dari beberapa tabel data master seperti tabel dosen, mahasiswa, matakuliah, ruangan, peserta kuliah, program studi, dan fakultas melalui *index* nya akan dirujuk oleh tabel berita acara mengajar dan tabel absensi dosen serta tabel absensi mahasiswa.



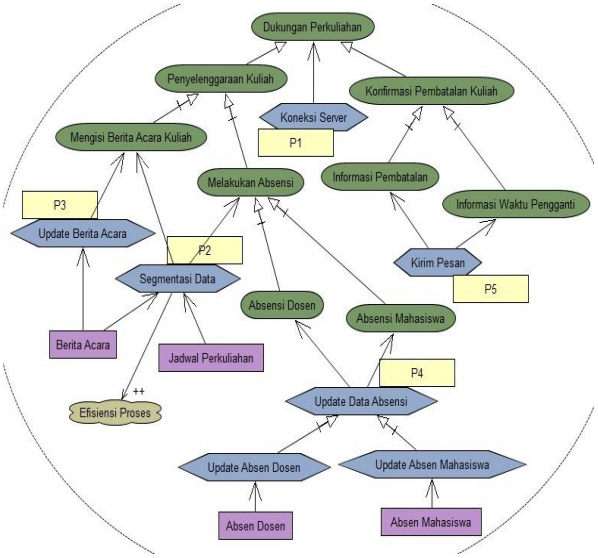
Gambar 3. Skema relasi data jadwal kuliah dan berita acara



Gambar 4. Skema relasi data berita acara dan absensi kuliah

Proses *query* yang dilakukan dari setiap koleksi data antara lain pada saat dosen melakukan pengisian berita acara, pengisian absensi dosen dan mahasiswa, serta kebutuhan pelaporan dari setiap program studi. Sistem akan mengambil daftar jadwal kuliah harian untuk menampilkan berita acara dan absensi pada saat jam perkuliahan berlangsung secara paralel disetiap kelas, dan sistem juga akan mencatat setiap penambahan data untuk setiap berita acara dan absensi perkuliahan kedalam *database*. Selain itu, sistem juga akan melayani *query* dari setiap program studi untuk kebutuhan administratif dan pelaporan.

Fungsionalitas sistem berupa dukungan perkuliahan terdiri dari dua pilihan *goal* (simbol oval), yaitu penyelenggaraan kuliah atau pembatalan kuliah, seperti ditunjukan pada Gambar 5. Pada bahasan ini kita fokus pada *sub goal* penyelenggaraan kuliah, dimana *goal* tersebut dapat tercapai dengan dukungan dari dua *sub goal* nya yaitu mengisi berita acara dan melakukan absensi, serta tiga *plan* (P) yaitu segmentasi data (P2), *update* berita acara (P3), dan *update* data absensi (P4). Sementara non-fungsionalitas sistem yang disimbolkan dengan awan, yaitu berhubungan dengan efisiensi proses yang mendapatkan panah kontibusi positif dari P2, artinya *plan* yang akan memenuhi kebutuhan efisiensi proses tersebut. Dalam kasus ini *plan* tersebut adalah melakukan segmentasi data terhadap sumber daya (simbol segi empat) berita acara dan jadwal perkuliahan.

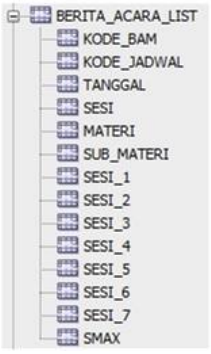


Gambar 5. Fungsionalitas dan non-fungsionalitas sistem

III.2 Segmentasi Data

Database engine yang digunakan untuk kebutuhan segmentasi data pada kasus yang dibahas adalah Oracle Database 11g Enterprise Edition. Efisiensi proses yang diperlukan berdasarkan Gambar 5 diantaranya adalah pada saat *query* berita acara dan absensi yang digunakan berdasarkan sesi pertemuan kuliah dimasing-masing kelas. Kebutuhan tersebut dapat direalisasikan dengan membuat segmen data yang dibagi berdasarkan sesi kuliah, sehingga menjadi beberapa *list* partisi tabel baru, potongan kode dan tabel hasil segmentasi. Seperti dapat dilihat pada Gambar 6 :

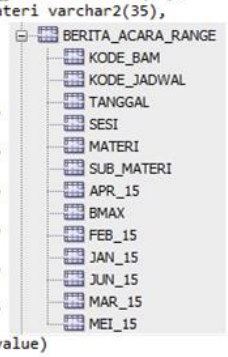
```
... table ...
       (kode_bam varchar2(10), kode_jadwal varchar2(10),
       tanggal date, sesi number, materi varchar2(35),
       sub_materi varchar2(55))
partition by list (sesi)
(
partition sesi_1 values ('1'),
partition sesi_2 values ('2'),
partition sesi_3 values ('3'),
partition sesi_4 values ('4'),
partition sesi_5 values ('5'),
partition sesi_6 values ('6'),
partition sesi_7 values ('7'),
...
partition smax values (default)
);
```



Gambar 6. Segmentasi data berdasarkan pertemuan kuliah

Sementara untuk kebutuhan pelaporan program studi, *query* data yang diperlukan adalah rekap perkuliahan perbulan, dalam kasus ini efensiesi proses dapat dilakukan melalui segmentasi data berdasarkan *range* bulan, seperti dapat dilihat pada Gambar 7. Atau misalnya untuk memenuhi kebutuhan administratif diperlukan *query* data sesi kuliah perbulan, maka gabungan dari partisi *range* dan *list* dapat diintegrasikan seperti ditunjukan pada Gambar 8.

```
... table ...
       (kode_bam varchar2(10), kode_jadwal varchar2(10),
       tanggal date, sesi number, materi varchar2(35),
       sub_materi varchar2(55))
partition by range (tanggal)
(
partition Jan_15 values less than
(to_date('01/02/2015','dd/mm/yyyy')),
partition Feb_15 values less than
(to_date('01/03/2015','dd/mm/yyyy')),
partition Mar_15 values less than
(to_date('01/04/2015','dd/mm/yyyy')),
partition Apr_15 values less than
(to_date('01/05/2015','dd/mm/yyyy')),
partition Mei_15 values less than
(to_date('01/06/2015','dd/mm/yyyy')),
partition Jun_15 values less than
(to_date('01/07/2015','dd/mm/yyyy')),
...
partition bmax values less than (maxvalue)
);
```



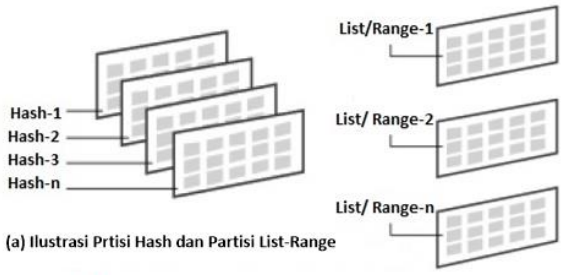
Gambar 7. Segmentasi data berdasarkan bulan

```
... table ...
       (kode_bam varchar2(10), kode_jadwal varchar2(10),
       tanggal date, sesi number, materi varchar2(35),
       sub_materi varchar2(55))
partition by range (tanggal)
subpartition by list (sesi)
(
partition Jan_15 values less than
(to_date('01/02/2015','dd/mm/yyyy'))
(subpartition sesi_A1 values ('1'),
subpartition sesi_A2 values ('2'),
subpartition sesi_A3 values ('3'),
 subpartition sesi_A4 values ('4')
),
partition Feb_15 values less than
(to_date('01/03/2015','dd/mm/yyyy'))
(subpartition sesi_B1 values ('1'),
subpartition sesi_B2 values ('2'),
subpartition sesi_B3 values ('3'),
subpartition sesi_B4 values ('4')
),
partition Mar_15 values less than
(to_date('01/04/2015','dd/mm/yyyy'))
(subpartition sesi_C1 values ('1'),
subpartition sesi_C2 values ('2'),
subpartition sesi_C3 values ('3'),
subpartition sesi_C4 values ('4')
),
...
);
```

Gambar 8. Segmentasi data berdasarkan bulan dan pertemuan kuliah

Selain ketiga contoh tersebut, jika kita menemukan kebutuhan kasus yang tidak sesuai dengan solusi segementasi *list* dan *range*, kita dapat melakukan segmentasi secara *hash*. Dimana koleksi data akan dibagi berdasarkan algoritma fungsi *hash*, yaitu setiap data dibagi secara merata sesuai dengan jumlah partisi yang sudah didefinisikan. Cara ini lebih efektif digunakan untuk mendistribusikan data secara acak tanpa harus memperhatikan nilai pada kolom data tersebut. Misalnya kita akan membuat segmen pada tabel berita acara berdasarkan kode berita acara, sementara partisi nya kita tetapkan sebanyak 5 partisi, maka dengan demikian sistem akan secara acak mendistribusikan data kedalam 5 partisi tersebut. Gambaran perbedaan antara segmentasi *list*, *range* dan *hash*, serta contoh potongan kode seperti terdapat pada Gambar 9. Berdasarkan kebutuhan kasus tertentu, untuk mengoptimalkan kemampuannya segmentasi *hash* ini juga dapat dikombinasikan dengan *range* seperti dikombinasikannya segmentasi *range* dengan *list*.

Beberapa contoh segementasi yang dilakukan pada Gambar 6, 7, 8 dan 9, merupakan upaya efesiensi proses yang berhubungan dengan kecepatan akses data, serta disesuaikan dengan kebutuhan kasusnya masing-masing. Namun jika kita telah lebih jauh solusi-solusi ini masih menyisakan persoalan, misalnya ketika data terus bertambah maka kriteria partisi dalam setiap tabel yang sudah kita definisikan tidak secara otomatis ikut bertambah. Artinya seorang *database administrator* (DBA) harus secara manual membuat partisi baru dalam data terpisah berdasarkan kriteria kebutuhan baru tersebut. Adapun kode partition "*bmax values less than (maxvalue)*" dan "*partition smax values (default)*" pada Gambar 6 dan 7, merupakan partisi untuk menampung data yang tidak sesuai kriteria kedalam satu partisi, bukan berdasarkan kebutuhan kriteria baru. Untuk menyelesaikan persoalan ini, maka penetapan suatu segmentasi harus dibuat secara otomatis dan dapat mengadaptasi kebutuhan kriteria baru.



(a) Ilustrasi Prtisi Hash dan Partisi List-Range

```
... table ...
       (kode_bam varchar2(10), kode_jadwal varchar2(10),
       tanggal date, sesi number, materi varchar2(35),
       sub_materi varchar2(55))
partition by hash (kode_bam)
partitions 5;
```

(b) Kode Partisi Hash

Gambar 9. Perbandingan partisi hash, list/range (a), segmentasi hash (b)

Berdasarkan referensi (Lorentz, 2014), (Baer, 2007), teknik *interval partitioning* merupakan salah satu cara yang dapat diterapkan, dimana cara ini memiliki kemampuan membuat partisi baru secara otomatis ketika terdapat data baru yang tidak

memenuhi kriteria partisi yang sudah ada. Partisi baru yang dihasilkan akan sesuai dengan kriteria partisi interval yang sudah didefinisikan sebelumnya dan terbentuk pada saat data tersebut masuk untuk pertama kalinya. Misalnya kita membuat kode segmentasi data untuk tabel berita acara dengan hanya membuat satu partisi yaitu "*partition Jan_15*" seperti ditunjukan pada Gambar 10, artinya semua data dengan kriteria Bulan Januari 2015 dan sebelumnya akan masuk kedalam partisi "*Jan_15*", pada kode kita tambahkan klausa "*interval (numtoyminterval(1,'month'))*", sehingga jika terdapat suatu data baru diluar kriteria itu maka sistem akan secara otomatis membuat partisi baru, misalnya jika dimasukan data untuk bulan Februari 2015. Gambar 11 menunjukan hasil tersebut, dimana tabel no. 208 dengan *partition name* JAN_15 merupakan partisi yang menampung data dengan kriteria tersebut, sementara tabel no. 209 dengan *partition name* SYS_P68 merupakan partisi baru yang dibuat secara otomatis oleh sistem ketika masuk data baru diluar kriteria Januari 2015, dengan *high value* yang sama (*date*) untuk menampung data bulan Februari 2015.

```
... table ...
        (kode_bam varchar2(10), kode_jadwal varchar2(10),
        tanggal date, sesi number, materi varchar2(35),
        sub_materi varchar2(55))
partition by range (tanggal)
interval (numtoyminterval(1,'month'))
(
partition Jan_15 values less than
(to_date('01/02/2015','dd/mm/yyyy'))
);
```

Gambar 10. Segentasi data secara interval

```
select * from dba_tab_partitions;
```

Script Output | Query Result | SQL | All Rows Fetched: 230 in 0,783 seconds

	TABLE_O...	TABLE_NAME	PARTITION_NAME		HIGH_VALUE
208	ARADEA	BERITA_ACARA_INTERVAL	..JAN_15	0	TO_DATE(' 2015-02-01
209	ARADEA	BERITA_ACARA_INTERVAL	..SYS_P68	0	TO_DATE(' 2015-03-01
210	YOGASWARA	BERITA_ACARA_LIST	..SESI_1	0	'1'
211	YOGASWARA	BERITA_ACARA_LIST	..SESI_2	0	'2'

Gambar 11. Hasil pembuatan partisi secara otomatis

IV. PEMBAHASAN DAN EVALUASI

Bahasan ilustrasi kasus memberikan contoh penerapan segementasi koleksi data berdasarkan kebutuhan-kebutuhan kasus. Dimana upaya peningkatan performansi yang berhubungan dengan indikator kecepatan akses, dapat dilakukan dengan berbagai cara partisi tabel. Ketepatan pemilihan cara yang digunakan akan sangat menentukan peningkatan dari kecepatan akses data tersebut. Sebagai evaluasi, kami membandingkan kecepatan akses data antara koleksi data yang tidak tersegmen dengan koleksi data yang tersegmen, dengan menggunakan data uji sejumlah 3.000 *data record*, serta spesifikasi *hardware* berada dikisaran CPU 1,9 Ghz dan Memory 4GB. Percobaan dilakukan masing-masing sebanyak 10 kali *query* data, baik terhadap koleksi data yang tersegmen maupun koleksi data yang tidak tersegmen. Berdasarkan hasil percobaan terbukti bahwa kecepatan akses data untuk koleksi data yang tersegmen lebih cepat atau lebih efisien 52 % seperti ditunjukan pada Tabel 1. Hasil evaluasi tersebut merupakan hasil segmentasi koleksi data dengan menggunakan partisi *hash*, evaluasi-evaluasi lanjutan perlu dilakukan untuk memperoleh informasi yang lebih spesifik, misalnya perbandingan dari berbagai jenis segmentasi lainnya, serta penggunaan beragam tipe data uji.

Tabel 1. Evaluasi Kecepatan Akses Data

Perbandingan Kecepatan Akses		
Evaluasi ke	***Data Tersegmen***	***Data Tidak Tersegmen***
1	0,047 detik	0,563 detik
2	0,031 detik	0,062 detik
3	0,063 detik	0,093 detik
4	0,16 detik	0,031 detik
5	0 detik	0,047 detik
6	0,16 detik	0,047 detik
7	0 detik	0,016 detik
8	0,047 detik	0,078 detik
9	0,031 detik	0,047 detik
10	0 detik	0,047 detik
Rata-rata *query* per detik	0,539 detik	1,031 detik

Selain jenis-jenis segmentasi data yang telah dibahas, sebenarnya masih terdapat beberapa jenis segmentasi lainnya yang dapat digunakan dalam melakukan partisi tabel ini, seperti *system partitioning* untuk kebutuhan penempatan data yang secara khusus dikelola oleh aplikasi, serta untuk dukungan ketika tabel dalam koleksi data perlu dipecah menjadi partisi yang lebih kecil, *virtual column partitioning* yaitu kolom virtual yang

dikalkulasikan atau yang diturunkan dari data lainnya dan dapat digunakan sebagai dasar untuk partisi tabel, hal ini memungkinkan tabel untuk dipartisi berdasarkan informasi bisnis yang tidak perlu disimpan dalam kolom individu, *reference partitioning* merupakan pilihan yang berguna untuk berurusan dengan dua tabel dalam hubungan *one-to-many*, *partition advisor* dapat membantu melakukan *generate* rekomendasi dalam proses partisi dan menampilkan peningkatan efisiensi performansi, serta dapat melakukan *generate script* pembuatan partisi (Baer, 2007) (Kholfi, 2011). Artinya ketepatan pilihan, atau kombinasi, serta pengembangan dari setiap solusi-solusi yang ada merupakan upaya yang tepat untuk meningkatkan performansi sistem secara menyeluruh.

V. KESIMPULAN

Segmentasi koleksi data merupakan upaya meningkatkan performansi sistem, dimana akses data terhadap tabel yang telah tersegmen dapat dilakukan lebih cepat, sehingga proses yang terjadi lebih efisien. Bahasan pada makalah ini merupakan kajian awal, dalam rangka mengenali bahan-bahan dasar yang akan dijadikan pilihan alternatif untuk mengembangkan suatu mekanisme performansi sistem secara menyeluruh, sebagai pekerjaan kami kedepan. Kami berencana mengembangkan suatu model yang lebih generik, sehingga kemampuan segmentasi data ini dapat dilakukan secara lebih fleksibel, tanpa harus melibatkan seorang DBA ketika kebutuhan baru muncul, misalnya hanya cukup dilakukan oleh *end user*, atau jika diperlukan dapat dilakukan secara otomatis, seperti mekanisme partisi interval, namun lebih dapat mengakomodasi kebutuhan pengguna dan mengantisipasi perubahan-perubahan saat *run-time*. Strategi untuk optimasi penerapan *SQL tuning* yang dikombinasikan dengan prinsip pengembangan *self-adaptive software systems* merupakan perhatian kami berikutnya.

REFERENSI

Soanes, C., Stevenson, A. (2008). Concise oxford english dictionary", Oxford University Press.

Chan, I., Ashdown, L. (2010). Oracle database performance tuning guide, 11g release 2 (11.2)", Copyright © 2010, Oracle, All rights reserved.

Whalen, E. (2004). Tuning oracle on windows for maximum performance on poweredge servers", Performance Tuning Corporation, Dell, Oracle and Microsoft.

Connolly, T., Begg, C., Carolyn. (2002). Database systems : a practical approach to design, implementation, and management, 4 rd edition", Addison Wesley, England.

http://www.tusc.com/oracle/training/course-apptune.html.

Cyran, M., Lane, P. (2003). Oracle database concepts, 10g release 1 (10.1)", Copyright © 2003 Oracle Corporation, All rights reserved.

Fogel, S. (2015). Oracle database administrator's guide, 11g release 2 (11.2)", Copyright © 2015, Oracle, All rights reserved.

Lorentz D., Roeser, M., B. (2014). Oracle database sql language reference, 11g release 2 (11.2)", Copyright © 2014, Oracle, All rights reserved.

Aradea, Supriana, I., Surendro, K. (2014). Requirements data mahasiswa antara fakta dan relatias", Jurnal Ilmiah Bidang Ilmu Teknologi (Tekno Insentif), Kopertis Wilayah IV, Volume 8, No.2.

Baer, H. (2007). Partitioning in oracle database 11g", Oracle Corporation, Copyright © 2007, Oracle. All rights reserved, USA.

Kholfi H., Y. (2011). Partitioning Pada Oracle 11g", Copyright © 2003-2011 Ilmu Komputer.com.