ISSN: 2477-5126
e-ISSN: 2548-9356

Tabel Partisi Pada STARS: Konsep Dan Evaluasi
(Studi Kasus STARS UKSW)

Infraim Oktofianus Boymau[1*)], Penidas Fiodinggo Tanaem[2], Andeka Rocky Tanaamah[3]
[1]Program Studi Sistem Informasi, Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana, Salatiga
[2,3]Program Studi Bisnis Digital, Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana, Salatiga
[1,2,3]Fakultas Teknologi Infromasi, Universitas Kristen Satya Wacana
[1,2,3]Jln. Diponegoro 52-60, Salatiga, Jawa Tengah, 50711, Indonesia
email: [1]682018141@student.uksw.edu, [2]penidas.fiodinggo@uksw.edu, [3]atanaamah@staff.uksw.edu

Abstract – **Database performance is one of the main components in supporting the sustainability of a system, in this case, STARS. In the system context, data will usually be collected into a database. Tied to the data collection process, this really affects the performance of a system as a whole, in this case when executing a query to get a return of the execution results, because the performance of the database itself will be affected by the amount of data available. One way to improve the performance of the database is to use the partition table concept. Thus, in this research a design and evaluation of the partition table will be carried out which will then be applied to the SWCU STARS database. This research focuses more on the use of vertical partitions and list partitions by utilizing PostgreSQL version 14. The stages used in this study. These stages include data collection, partition design, technical partition, testing and implementation. The results of this study indicate that partition tables have better performance than non-partition tables. Judging from some of the sql syntax, namely update, delete and select, while insert has poor performance for partition tables compared to non-partition tables.**
Keywords - **Database, Table Partitioning, Postgresql.**

Abstrak – **Kinerja database merupakan salah satu komponen utama dalam menunjang keberlangsungan dari sebuah sistem dalam hal ini adalah STARS. Pada konteks sistem, data biasanya akan di koleksi kedalam sebuah database. Terikat dengan proses koleksi data, hal ini benar-benar berpengaruh pada kinerja dari sebuah sistem secara menyeluruh, dalam hal ini adalah pada saat mengeksekusi sebuah query sampai mendapatkan pengembalian dari hasil ekseskusi, sebab kinerja dari database sendiri akan dipengaruhi oleh jumlah data yang ada. Salah satu cara untuk meningkatkan kinerja dari database adalah dengan menggunakan konsep tabel partisi. Dengan demikian, pada penelitian ini akan dilakukan perancangan dan evaluasi terhadap tabel partisi yang kemudian diterapkan pada database STARS UKSW. Penelitian ini lebih berfokus pada penggunaan partisi vertical dan list partition dengan memamfaatkan postgresql versi 14. Adapun tahapan-tahapan yang digunakan pada penelitian ini. Tahapan tersebut antara lain, data collection, desain partisi, technical partition, testing dan implementasi. Hasil dari penelitian ini menunjukan bahwa tabel partisi memiliki kinerja yang lebih baik dibandingkan tabel non-partisi. Dilihat dari beberapa sintaks sql yaitu update, delete dan select, sedangkan insert memiliki kinerja yang buruk untuk tabel partsi dibanding tabel non-partisi.**
Kata Kunci – **Database, Tabel Partisi, Postgresql.**

*) **penulis korespondensi**: Infraim Oktofianus Boymau
Email: 682018141@student.uksw.edu

I. PENDAHULUAN

Salah satu aspek yang perlu untuk diperhatikan dalam keberlangsungan sebuah sistem adalah data dan tidak dapat dielak bahwa data akan terus meningkat seiring berjalannya waktu. Dalam konteks sistem, data biasanya akan di koleksi kedalam sebuah database. Terkait proses koleksi data, hal ini sangat perpengaruh pada kinerja dari sebuah sistem secara menyeluruh, dalam hal ini adalah dalam mengeksekusi sebuah query sampai mendapatkan pengembalian dari hasil ekseskusi, karena kinerja dari database sendiri akan dipengaruhi oleh jumlah data yang ada. Terdapat beberapa teknik yang dapat diimplementasikan kedalam sistem untuk menjawab kondisi ini, ada cache, partition dll. Namun salahsatu teknik yang dapat digunakan dalam database yakni table partition.

Table partition sendiri merupakan teknik yang melibatkan pemecahan tabel besar (induk) menjadi satu set tabel anak [1]. Maabreh berpendapat bahwa teknik partisi dapat memberikan peningkatan kinerja database [2]. Jika di hubungkan dengan konteks pengelolaan data atau yang biasa dikenal dengan istilah CRUD (Create, Read, Update dan Delete), proses Read lah yang paling sering dieksekusi yang mana didalam database dikenal dengan istilah Select dibanding proses yang lain. Kondisi tersebut juga ditekankan oleh Luthfina dimana sebuah system sangat bergantung pada fungsi query untuk pencarian data dari berbagai atribut tabel database [3]. Dalam menanggulangi proses tersebut, sistem akan lebih dipermudah mengambil data dari serpihan-serpihan kecil dari table atau tabel slave dari pada mengambil data dari tabel utama atau tabel master.

Sebuah penelitian yang dilakukan oleh Lutfina, dinyatakan bahwa biaya dan kecepatan akses data pada sebuah kueri sangat bergantung pada jumlah atribut tabel pada sebuah database. Dengan demikian dilakukan studi lebih mendalam untuk membandingkan fragmentasi vertical tabel database dengam memanfaatkan algoritma Bond Energy (BEA) dengan dikombinasikan dengan algoritma Graph-based vertical partitioning (GBVP) untuk dengan tujuan untuk dapat digunakan dalam mengembangkan sebuah database terdistribusi. Adapun hasil yang didapatkan yakni algoritma GBVP menunjukan hasil yang lebih baik namun menghasilkan nilai partisi yang lebih tinggi. Dinama GBVP memiliki waktu kesekusi yang relative lebih cepat yakni 0.003s, 0.002s, 0.002s, 0.002s, 0.003s dan 0.003s. Sedangkan BEA menghasilkan 0.033, 0.058, 0.088, 0.004, 0.079, 0.002

[3]. Penelitian tersebut juga diperkuat oleh Nair M dkk. Dimana pada penelitian Nair M dkk dilakukan sebuah perbandingnan dengan metode pengkodean tabel yang tradisional dan pengkodean tabel partisi. Hasil yang didapatkan adalah tabel partisi terbukti lebih efisien secara komperhensif dalam hal melakukan eksekusi kueri [4].

Lebih lanjut penelitian dari Maabreh yang bertujuan untuk mengevaluasi teknik partisi data dalam meningkatkan kinerja query yang yang dikirim ke database utama. Dimana pada penelitian ini dilakukan sebuah eksperiman yang menunjukan bahwa database yang dipartisi dapat meningkatkan kinerja dari sebuah database yang mana lebih baik 35% dibanding dengan database yang tidak dipartisi [2]. Sedangkan peneilitian Tabassam dan Obermaisser, diterapkan teknik partisi tabel ke node Mutli- Query Graph (DMG) yang memiliki kueri diagnostik dalam format SQL. Tujuannya adalah untuk meminimalkan penggunaan sumber daya yang dicapai dengan memperkenalkan teknik jistori interve dan skip faktor disetiap DMG. Pendekatan ini menunjukkan pengurangan yang signifikan dari rentang produksi dan konsumsi sumber daya untuk berbagai jenis DMG [5].

Sebuah studi yang dilakukan oleh Swathi dalam penelitannya yakni A Study on SQL – RDMBS (Relational Database Management System) Concept and Database Normalization menjelaskan bahwa rata-rata vendor RDMBS seperti MS Sql Server, IBM DB2, Oracle, MySql dan Microsoft Access mengembangkan system Database Management System mereka dengan mengacu pada standar model relational yang diperkenalkan oleh E. F. Codd [6]. Selanjutnya Praba dan Safitri melakukan perbandingan dari dua RDBMS yang ada saat ini, yakni MySql dan Postgresql. Tujuannya untuk mencari tau kemampuan dari masing-masing RDBMS, yang mana ditunjukan bahwa Postgresql memiliki kemapuan yang lebih baik dari MySql dalam hal melakukan eksekusi query select * from, select count(), dan join table. Hal tersebut dilihat dari perbedaan waktu respon dari keduanya yang mana postgresql lebih cepat dalam hal response time disbanding mysql sekita 0.44ms [7].

Adapun penelitian lainnya yang dilakukan oleh Hassan dan Bansal yang mengungkapakan bahwa terdapat tiga masalah utama dalam RDF (Resource Description Framework), yakni efisiensi kueri, solusi dioptimalkan untuk jenis pola kueri tertentu dan yang terakhir adalah berkaitan dengan pengurangan waktu pra-pemrosesan dan pemuatan data. Dengan demikian diusulkan skema partisi relasional yang disebut Subset Property Table (SPT) dengan model Partisi Vertikal (VP) untuk data RDF yang selanjutnya akan diimplementasikan kedalam tabel properti yang ada ke dalam subset tabel untuk meminimalkan input kueri dan operasi gabungan. Hasil yang ditunjukan dari penelitian ini adalah pendekatan gabungan (SPT + VP) mengungguli sistem canggih berdasarkan mesin pemrosesan dalam memori dalam lingkungan terdistribusi [8].

Selanjutnya Hassan dan Bansal juga mengajukan skema partisi relasional yang disebut Partisi Tabel Properti (PTP) untuk data RDF, yang selanjutnya mempartisi Tabel Properti yang ada menjadi beberapa tabel berdasarkan properti yang berbeda untuk meminimalkan input data dan operasi gabungan yang dikenal dengan S3QLRDF, yang dibangun diatas Spark dan SPARQL melalui skema PTP. Pada penelitian ini dilakukan evaluasi eksperimental yang berhubungan dengan preprocessing costs and query performance, menggunakan Lehigh University Benchmark (LUBM) dan dataset Waterloo SPARQL Diversity Test Suite (WatDiv) hingga 1,4 miliar tiga kali lipat. Hasil yang didapatkan yakni S3QLRDF mengungguli sistem manajemen RDF terdistribusi yang canggih [9].

Penelitian selanjutnya dari Salgova dan Matiasko, yakni membandinginkan bebagai teknik partisi yang dapat diimplementasikan kedalam sebuah database dengan lebih mempertimbangkan efisiensi akses. Adapun terdapat beberapa teknik yang digunakan yakni Hash, range dan list partition. Adapun eksperimen ini dilakukan dengan menggunakan Oracke 11g. Adapun hasil yang didapatkan yakni secara signifikan menunjukkan bahwa akses ke data partisi dapat secara signifikan mengurangi waktu akses ke database. Penggunaan teknik partisi range memberikan waktu akses yang ditingkatkan secara signifikan ketika kurang dari 30 dari 34 partisi diakses. Dengan akses ke 30 partisi atau lebih, penyimpanan data tanpa partisi terbukti lebih tepat. Saat menggunakan partisi list, ada peningkatan yang lebih signifikan dalam waktu kinerja, dan peningkatan ini bertahan bahkan ketika mengakses beberapa atau semua partisi. Kesimpulannya, dapat dievaluasi dari hasil bahwa partisi dapat meningkatkan waktu akses data secara signifikan [10].

Disisi lain, STARS (Student's Activity Record System) merupakan sebuah sistem yang digunakan untuk melakukan proses manajemen yang berfokus pada bidang kemahasiswaan Universitas Kristen Satya Wacana (UKSW) [11]. Terdapat beberapa subsistem yang terus dikembangkan pada STARS, salah satunya adalah Kredit Keaktivan Mahasiswa (KKM). Sistem KKM pada STARS sendiri dikembangkan untuk melakukan rekapan untuk setiap keaktivan mahasiswa. Dengan banyaknya mahasiswa dilingkungan UKSW yakni berkisar 15000 sampai dengan 16000, akan sangat berpengaruh pada kinerja dari STARS itu sendiri. Dimana masing-masing mahasiswa memiliki aktivitas kemahasiswaan yang berbeda-beda, yang mana nantinya akan berpengaruh pada STARS itu sendiri. Dengan demikian dibutuhkan sebuah konsep yang dapat digunakan atau diimplementasikan kedalam STARS untuk menghasilkan informasi-informasi yang dibutuhkan pengguna dengan efisien.

Mengacu pada uraian yang ada, maka pada penelitian ini akan dilakukan perancangan, evaluasi terhadap tabel partisi yang kemudian diterapkan pada database STARS UKSW. Adapun hasil yang ingin dicapai dari penelitian ini adalah menghasilkan konsep partisi yang digunakan pada sistem STARS dengan melakukan evaluasi-evaluasi terhadap konsep tersebut.

II. LANDASAN TEORI

Tabel Partisi merupakan sebuah teknik yang dapat digunakan untuk membagi data pada tabel kedalam tabel-tabel yang lebih kecil yang dapat kelola secara independent [1]–[4], [8], [12]–[14]. Atau juga dapat diartikan sebagai sebuah teknik yang dapat digunakan untuk memetakan atau mendistribusikan data dari tabel utama kedalam tabel-tabel kecil untuk meningkaktan efiseiensi dari database yang digunakan. Teknik ini bertujuan juga untuk mengurangi jumlah pembacaan fisik pada sebuah database ketika sebuah

query dieksekusi atau di jalankan [1], [4]. Sebagai contoh, dalam database Postgesql terdapat tiga tipe partisi [15], yakni:

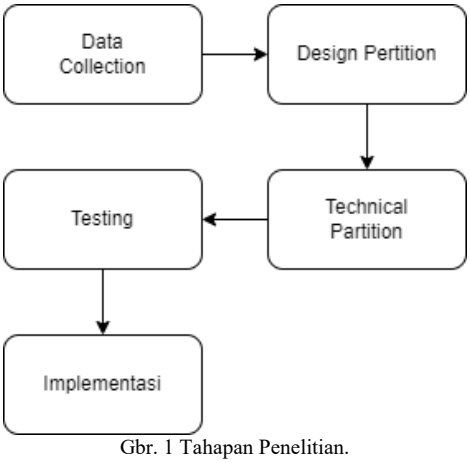
1. Range: Tipe partisi ini digunakan untuk melakukan partisi berdasarkan kelompok data range yang ditentukan. Sebagai contoh partisi berdasarkan tanggal, tahun, umur dll.
2. List: Tipe partisi ini dilakukan secara eksplisit dengan mencantumkan nilai kunci mana yang muncul di setiap partisi.
3. Hash: Tipe partisi ini dilakukan dengan menentukan modulus dan sisa untuk setiap partisi. Setiap partisi akan menampung baris yang nilai hash dari kunci partisi dibagi dengan modulus yang ditentukan akan menghasilkan sisa yang ditentukan.

Sedangakan dilihat dari sisi relasinya, terdapat 2 jenis partisi, yakni:

1. Horizontal: yakni menempatkan baris yang berbeda ke dalam tabel yang berbeda [2], [4], [13], [16], [17].
2. Vertical: membuat tabel dengan kolom lebih sedikit dan menggunakan tabel tambahan untuk menyimpan kolom yang tersisa [8], [17] .

III. METODE PENELITIAN

Pada penelitian ini, lebih berfokus pada penggunaan partisi vertikal dan list partition dengan memanfaatkan postgresql versi 14. Terdapat tahapan-tahapan yang digunakan pada penelitian ini. Tahapan tersebut antara lain, data collection, desain partisi, technical partition, testing dan implementasi. Data collection merupakan tahapan yang digunakan untuk mengumpulkan setiap data yang akan digunakan untuk dipartisi. Dari hasil koleksi data yang didapatkan, maka selanjutnya dilakukan desain partisi. Dari hasi partisi, selanjutnya akan diimplementasikan kedalam bentuk partisi yang disepakati dari hasil design. Tahapan berikutnya adalah pengujian untuk mencaritahu efisiensi dari partisi yang dikembangkan. Tahapan terakhir adalah implementasi.



Gbr. 1 Tahapan Penelitian.

IV. HASIL DAN PEMBAHASAN

A. Design

Untuk menunjukan simulasi dari desain partition yang akan digunakan, dalam kasus ini akan digunakan dua data sebagai sampel, yakni data partisipan (s) dan data partisipasi (a) atau diartikan dengan tabel master yakni tabel master partisipasi dan tabel master partisipan. Partisi akan dibuat dalam bentuk tabel dengan tipe horizontal dan tipe list. Asumsi awal yang digunakan adalah, setiap data mahasiswa atau partisipan akan dipartisi (tabel 1), karena masing-masing mahasiwa memiliki data kegiatan yang berbeda-beda atau sama. Jadi, jika terdapat lima mahasiswa, maka partisi partisipan yang akan dibuat adalah sebanyak lima table partisi. Proses pembagian partisi ini (partisi partisipan) dilakukan dengan mengidentifikasi primary key yang telah ditentukan, dalam hal ini adalah nomer induk mahasiswa. Hal ini berlaku sama dengan partisi partisipasi, dimana primary key yang digunakan untuk pembuatan table partition partisipasi adalah id kegiatan. Dengan konsep inilah, maka pertumbuhan partisi akan meningkat secara eksponensial seiring dengan bertambahnya s dan a. Dengan demikian notasi yang akan digunakan adalah sebagai berikut:

$$s = \{s1, s2, s3, s4, s5, \ldots, s_N\} \quad (1)$$

$$= \{s_n\}_n^N = 1 \quad (2)$$

$$a = \{a1, a2, a3, a4, a5, a6, \ldots, a_N\} \quad (3)$$

$$= \{a_n\}_n^N = 1 \quad (4)$$

TABEL I

TABEL MATRIKX PARTISI

s\a	a1	a2	a3	a4	a5	a6	Total partisipasi
s1	1			1		1	3
s2	1		1				2
s3	1	1		1		1	4
s4	1		1				2
s5	1				1	1	3
Total partisipan	5	1	2	2	1	3	14

Berdasarkan tabel 1, maka data yang akan dipartisi dari tabel master partisipan dapat dinotasikan sebagai berikut:

master_partisipan = {a1, a4, a6, a1, a3, a1, a2, a4, a6, a1, a3, a1, a5, a6}, atau terdapat 14 partisipasi mahasiswa di semua kegiatan.

$$p(s1) = \{a1, a4, a6\}; \quad (5)$$

$$p(s2) = \{a1, a3\}; \quad (6)$$

$$p(s3) = \{a1, a2, a4, a6\}; \quad (7)$$

$$p(s4) = \{a1, a3\}, \quad (8)$$

$$p(s5) = \{a1, a5, a6\}. \quad (9)$$

Sedangkan partisi dari tabel master partisipasi dinotasikan sebagai berikut:

master_partisipasi = {a1, a1, a1, a1, a1, a2, a3, a3, a4, a4, a5, a6, a6, a6}, atau terdapat 14 pastisipasi mahasiswa di semua kegiatan.

$$p(a1) = \{s1, s2, s3, s4, s5\}; \quad (10)$$

$$p(a2) = \{s3\}; \quad (11)$$

$$p(a3) = \{s2, s4\}; \quad (12)$$

$$p(a4) = \{s1, s3\}; \quad (13)$$

$$p(a5) = \{s5\}; \quad (14)$$

$$p(a6) = \{s1, s3, s5\}. \quad (15)$$

Dengan adanya konsep partisi seperti ini, maka client akan lebih mudah untuk mengakses data yang sama dari partisi yang berbeda-beda. Sebagai contoh, ketika mahasiswa ingin mengambil data-data partisipasinya pada kegiatan yang ada, maka mahasiswa (s1) tersebut akan diarahkan untuk mengakses partisi p(s1). Sebaliknya, jika admin akan mengakses data-data peserta (a1), maka admin akan diarahkan untuk mengakses partisi p(a1). Tujuan lainnya adalah untuk memudahkan sistem dalam membuat laporan atau report.

B. Technical Partition.

Dari desain yang didapatkan, maka pada tahapan teknikal partisi akan dilakuakan pengembangan yang disesuaikan. Teknikal partisi mencakup bagaimana melakukan generate partisi, hapus partisi dan implementasi fungsi-fungsi CRUD pada STARS, dimana setiap proses tersebut akan dilakukan dilapisan backend dari STARS dan tidak pada lapisan database. Langkah ini diambil dengan tujuan yakni untuk mengurangi beban kerja dari database.

Setiap fungsi seperti generate partisi, hapus partisi, create, update dan delete, dieksekusi dengan menggunakan TRANSACTION COMMIT atau TRANSACTION ROLLBACK selain daripada sintaks SELECT, karena SELECT hanya digunakan untuk mengabil data tanpa ada perubahan didatabase. COMMIT digunakan untuk memastikan apakah setiap sintaks sql yang digunakan berjalan dengan baik, sedangkan ROLLBACK digunakan untuk mendeteksi jika ada keasalah dalam mengeksekusi sintaks yang kemudian database akan kembali ke kondisi sebelumnya.

Dalam mengeksekusi setiap sintaks Create, sistem terlebih dahulu akan mengeksekusi create table partisi participant (“CREATE TABLE IF NOT EXISTS participant_partition_s1 PARTITION OF participant FOR VALUES IN (‘s1’);”) dan tabel partisi participation (“CREATE TABLE IF NOT EXISTS participation_partition_a1 PARTITION OF participation FOR VALUES IN (‘a1’);”). Dimana, jika tabel yang dimaksud tidak ditemukan, maka tabel tersebut akan di create kedalam database yang mana disesuaikan dengan values primary key yang ditentukan, jika tabel yang dimaksud ditemukan maka selanjutnya sistem akan sistem akan mengeksekusi fungsi create. Saat fungsi create dijalankan, sistem akan mengeksekusi dua sintaks insert, yakni insert ke tabel partisi participant_partition_s1 dan insert ke tabel partisi participation_partition_a1. Selanjutnya, jika setiap sintaks yang dieksekusi berhasil dijalankan, maka tindakan selanjutnya adalah COMMIT. Jika tidak maka ROLLBACK akan dijalankan. Proses ini diterapkan pada sintaks Update, Delete dan hapus partisi.

Selanjutnya adalah fungsi read. Read sendiri gigunakan untuk mengambil data dari database. Fungsi ini dijalankan dengan memanfaatkan sintaks database yaitu SELECT. Dimana SELECT akan ditujukan secara spesifik ke tabel partisi dimaksud (“SELECT * FROM participant_partition_s1 ORDER BY id ASC, nim ASC;”). Hal ini dilakukan untuk mengurangi waktu proses di database dibanding langsung mengambil dari tabel master (“SELECT * FROM participant ORDER BY id ASC, nim ASC;”), karena tumpukan data di tabel master akan lebih banyak dibanding tabel partisi.

C. Testing

Proses testing dilakukan pada dengan menggunakan komputer dengan spesifikasi sebagai berikut:

OS: Ubuntu 20.04.5 LTS;
Memory: 15,5 GiB;
Processor: AMD® Ryzen 7 5800x 8-core processor × 16;
Database: Postgresql 14.

Dalam melakukan proses testing dari apa yang telah di buat, digunakan bebarapa kofigurasi atau tunig untuk meningkatkan kinerja pada tabel partisi dan tabel non partisi. Yang pertama adalah indexing dengan menggunakan btree dari postgres, dimana masing-masing tabel (partition dan non-partition) di indexing pada masing-masing 3 kolom yakni id, kegiatan dan nim. Keuntungan dari indexing sendiri adalah untuk meningkatkan kinerja dari I/O yang berbasis pada data. Yang berikut adalah AUTO VACCUM. AUTO VACUUM digunakan untuk menghapus ruang terfragmentasi pada tabel dan indeks dan yang dihasilkan pada saat transaksi berhasil dijalankan. Data harus divakum secara teratur untuk memastikan kesehatan dan kinerja yang lebih baik dari database. Yang terakhir adalah konfigutasi shared_buffers = 512MB. Peranan dari shared_buffers sendiri adalah untuk menentukan jumlah memori yang dapat digunakan PostgreSQL untuk caching. Mekanisme caching ini digunakan untuk menyimpan isi tabel dan indeks dalam memori [15].

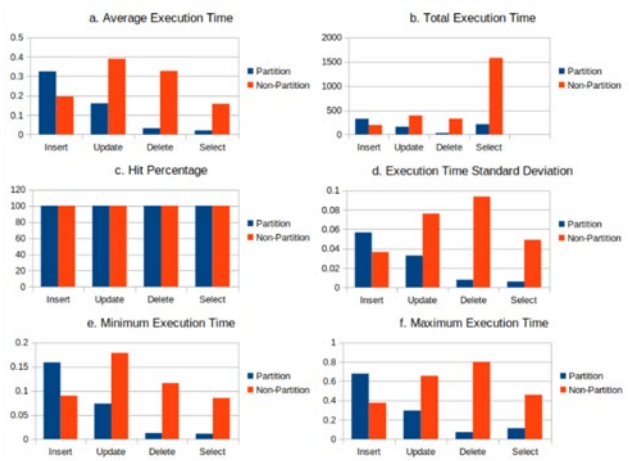
Adapun terdapat beberapa skenario yang digunakan pada saat testing. Skenario pertama adalah di tabel partition dan non-partition terdapat 25 juta data yang digunakan sebagai sample. Testing diterapkan pada dua tabel, yakni tabel x_participant (tabel non partisi) dan tabel participant_partition_$1 (tabel yang dipartisi). Skenario berikut adalah testing akan diawali dengan insert kemudian update, delete dan diakhiri dengan select. Insert dilakukan sebanyak 1k kali dengan batas waktu yang digunakan adalah 100 detik, yang mana disetiap detik akan dieksekusi 10 proses insert ke database. Configurasi ini berlaku sama juga pada saat update dan delete. Sedangkan select sedikit berbeda, dimana select dilakukan dengan dieksekusi sebanyak 10k dalam 100 detik, yang berati setiap detik akan terdapat 100 select yang digunakan. Skenario terakhir adalah proses insert, update, delete dan select hanya menggunakan proses standard yang dikombinasikan dengan klausa where bagi update, delete dan select, selain dari itu tidak digunakan (Tabel 2). Skenario-skenario ini akan dijalankan dengan menggunakan alat bantu yakni apache jmeter [18].

TABEL II.

TABEL SINTAKS TEST.

Partition	Query
N	UPDATE x_participant SET catatan=$1 WHERE nim=$2 and kegiatan = $3
Y	UPDATE participant_partition_$1 SET catatan=$2 WHERE nim=$3 and kegiatan = $4
N	DELETE FROM x_participant WHERE nim=$1 and kegiatan = $2
Y	DELETE FROM participant_partition_$1 WHERE nim=$2 and kegiatan = $3
N	SELECT * FROM x_participant WHERE nim=$1 and kegiatan = $2
Y	SELECT * FROM participant_partition_$1 WHERE nim=$2 and kegiatan = $3
N	INSERT INTO x_participant(nim, kegiatan, catatan, ts)

	VALUES ($1, $2, $3, $4);
Y	INSERT INTO x_participant_partition_$1(nim, kegiatan, catatan, ts) VALUES ($2, $3, $4, $5);



Gbr. 2 Testing Result.

Seluruh percobaan dilakukan dengan memastikan bahwa setiap sintaks yang dieksekusi berhasil dijalankan tanpa ada yang gagal (Gambar 2 c) dengan presentase hit adalah 100%. Adapun hasil yang didapatkan dari pengujian ini adalah sebagai berikut. Yang pertama adalah eksekusi sintak insert, dilihat dari nilai Rata-rata (Gambar 2 a), Total (Gambar 2 b), Standar Deviasi (Gambar 2 d), Minimal (Gambar 2 e) dan Maksimal (Gambar 2 f) waktu eksekusi. Sintaks Insert pada kedua tabel terlihat memiliki perbedaan yang signifikan, terpaut sekitar 39.71%, yang mana tabel partisi memiliki nilai 0.32 ms dan tabel non-partisi bernilai 0.19 ms. Sedangkan untuk mengeksekusi seluruh sintaks insert pada tabel partisi dibutuhkan 325.28 ms dan tabel non-partisi 196.10 ms. Sedangkan populasi dari waktu yang dihabiskan untuk mengeksekusi sintaks adalah 0.05 ms untuk table partition dan 0.03 ms untuk tabel non-partition. Selanjutnya adalah waktu tercepat untuk insert pada tabel partition adalah 0.15 ms dan waktu terlamanya adalah 0.67 ms, sedangkan tabel non-partition adalah tercepat 0.08 ms dan terlama adalah 0.37 ms. Dari kondisi tersebut, insert pada tabel partisi lebih lambat dari tabel non-partisi. Kondisi ini disebabkan atas dasar jika data yang disimpan secara terus menerus, maka postgres akan terus mencari key pada tabel partisi ada. Jadi jika ada 16 ribu tabel partisi, maka akan ada juga tumpukan 16 ribu key partisi, sehingga waktu yang dibutuhkan akan tergantung pada seberapa sering partisi harus dialihkan pada saat sintaks insert dijalankan pada tabel partisi.

Dari eksekusi sintaks untuk update, delete dan select, tabel partisi jauh lebih cepat dibandingkan tabel non-partisi dilihat dari rata-rata waktu eksekusi. Untuk update adalah 0.16 ms bagi tabel partisi dan 0.39 ms untuk tabel non-partisi atau tabel partisi lebih cepat 58.86% dibanding tabel non-partisi. Sedangkan delete, 0.03 ms untuk partisi dan 0.39 ms untuk non-partisi atau partisi lebih cepat 90.18% dari non-partisi. Yang terakhir adalah select. Select sendiri sedikit berbeda dari ketiga sintaks sebelumnya, yang mana select memiliki lebih banyak proses eksekusi yang lebih banyak untuk mencari tau tingkat keberhasilan dari setiap eksekusi. Namun demikian proses select pada kedua tabel benar-benar berhasil dijalankan atau dieksekusi tanpa adanya kegagalan. Hal ini dibuktikan dengan presentasi hit (gambar 2 c) select yang ada. Sedangkan untuk rata-rata waktu eksekusi, tabel partisi terlampau lebih jauh dari non-partisi sekitar 86.53% untuk sintak select. Yang mana select pada partisi dapat dieksekusi dengan waktu 0.02 ms dan 0.26 ms untuk non-partisi.

Berdasarkan uraian-uraian pembahasan yang ada, maka hipotesis yang dapat diambil adalah dari tiga sintaks database yang ada yakni update, delete dan select, tabel partisi dapat bekerja dengan maksimal dalam mengeksekusi ketiga sintaks tersebut. Berbeda dengan insert yang mana tabel partisi lebih lama dari non-partisi karena banyaknya tumpukan key yang ada berdasarkan jumlah partisi yang dibangun.

D. Implementation

Adapun hasil dari pengembangan database ini diimplementasikan pada server dengan spesifikasi sebagai berikut:

OS: Ubuntu server 20.04 lts;
Processor: 4 core;
RAM: 4 gb;
Database: Postgresql 14.04.

Adapun dari hasil implementasi selama dua tahun terakhir semenjak partisi diterapkan, tidak ditemukan masalah-masalah yang berpengaruh pada keberlangsungan dari sistem STARS dari sisi database. Dampak dari implementasi ini yakni dapat meningkatkan waktu respons dari sistem ke client.

V. KESIMPULAN

Kesimpulan yang dapat diambil dari penelitian ini adalah penerapan tabel partisi pada database dapat mempengaruhi kinerja dari sebuah sistem, dalam hal ini adalah STARS. Hal ini dapat dilihat dari rata-rata waktu yang dibutuhkan dalam mengeksekusi sebuah sintaks, jumlah waktu yang dibutuhkan dalam keseluruhan eksekusi sintaks, dll. Yang menjadi salah satu catatan pentingnya adalah, dalam menjalankan sintaks insert, terdapat problem yang dihadapi yakni waktu eksekusi sintaks yang lambat dari tabel partisi dibanding tabel non-partisi. Namun hal ini dapat terbayarkan dengan waktu eksekusi sintak update, delete dan select dari tabel partisi yang terlampau lebih cepat dibandingkan tabel non partisi. Yang terakhir adalah, penelitian ini dapat menghasilkan rancangan, konsep tabel partisi yang digunakan pada STARS yang ditunjang dengan evaluasi-evaluasi yang dilakukan pada waktu eksekusi yang didapatkan. Berdasarkan hasil pengujian dan evaluasi yang didapatkan, maka tabel partisi sangat tepat untuk diimplementasikan pada STARS UKSW dan terbukti dapat mengurangi beban kerja pada server database.

Dari hasil yang didapatkan dari penelitian ini, maka perlu untuk dilakukan penelitian lebih lanjut terkait:

1. Menigkatkan waktu ekseskusi insert pada tabel partisi.
2. Table partisi untuk konsumsi data real-time pada STARS

DAFTAR PUSTAKA

[1] A. Viloria, G. C. Acuña, D. J. A. Franco, H. Hernández-Palma, J. P. Fuentes, and E. P. Rambal, “Integration of data mining techniques to postgresQL database manager system,” Procedia Comput Sci, vol. 155, no. 2018, pp. 575–580, 2019, doi: 10.1016/j.procs.2019.08.080.

[2] K. S. Maabreh, “Optimizing Database Query Performance Using Table Partitioning Techniques,” ACIT 2018 - 19th International Arab Conference on Information Technology, pp. 1–4, 2019, doi: 10.1109/ACIT.2018.8672584.

[3] E. Lutfina, "A Performance Comparative of Vertical Fragmentation Table using Bond Energy and Graph Based Vertical Partitioning Algorithm.".
[4] M. Abhishek Nair, A. Dewangan, and A. Geetha Mary, "Efficient Retrieval of Data from Cloud Databases using Hash Partitioned Buckets," 2019 Innovations in Power and Advanced Computing Technologies, i-PACT 2019, pp. 1–7, 2019, doi: 10.1109/i-PACT44901.2019.8960047.
[5] N. Tabassam and R. Obermaisser, "Minimizing the Make Span of Diagnostic Multi-Query Graphs Using Query Aware Partitioning in Embedded Real-Time Systems," Proceedings - 2018 International Conference on Promising Electronic Technologies, ICPET 2018, no. 2, pp. 1–7, 2018, doi: 10.1109/ICPET.2018.00007.
[6] P. Swathi, "A STUDY ON SQL-RDBMS CONCEPTS AND DATABASE NORMALIZATION." [Online]. Available: https://ssrn.com/abstract=4282707
[7] A. Dwi Praba and M. Safitri, "STUDI PERBANDINGAN PERFORMANSI ANTARA MYSQL DAN POSTGRESQL," vol. VIII, no. 2, [Online]. Available: https://www.adminer.org/.
[8] M. Hassan and S. K. Bansal, "Data Partitioning Scheme for Efficient Distributed RDF Querying Using Apache Spark," Proceedings - 13th IEEE International Conference on Semantic Computing, ICSC 2019, pp. 24–31, 2019, doi: 10.1109/ICOSC.2019.8665614.
[9] M. Hassan and S. K. Bansal, "S3QLRDF: Property Table Partitioning Scheme for Distributed SPARQL Querying of large-scale RDF data," Proceedings - 2020 IEEE International Conference on Smart Data Services, SMDS 2020, pp. 133–140, 2020, doi: 10.1109/SMDS49396.2020.00023.
[10] V. Salgova and K. Matiasko, "Reducing Data Access Time using Table Partitioning Techniques," ICETA 2020 - 18th IEEE International Conference on Emerging eLearning Technologies and Applications, Proceedings, pp. 564–569, 2020, doi: 10.1109/ICETA51985.2020.9379231.
[11] P. F. Tanaem, A. F. Wijaya, A. D. Manuputty, and G. N. Huwae, "Penerapan RESTFul Web Service Pada Disain Arsitektur Sistem Informasi Pada Perguruan Tinggi (Studi Kasus: STARS UKSW)," JASIEK (Jurnal Aplikasi Sains, Informasi, Elektronika dan Komputer), vol. 2(1), no. 1, pp. 11–20, 2020.
[12] H. Yin, S. Yang, H. Zhao, and Z. Chen, "Partial query optimization techniques for partitioned tables," Proceedings - 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2012, no. Fskd, pp. 792–797, 2012, doi: 10.1109/FSKD.2012.6234270.
[13] L. Qu et al., "Are current benchmarks adequate to evaluate distributed transactional databases?," BenchCouncil Transactions on Benchmarks, Standards and Evaluations, vol. 2, no. 1, p. 100031, 2022, doi: 10.1016/j.tbench.2022.100031.
[14] F. C. Daeng Bani, Suharjito, Diana, and A. S. Girsang, "Implementation of Database Massively Parallel Processing System to Build Scalability on Process Data Warehouse," Procedia Comput Sci, vol. 135, pp. 68–79, 2018, doi: 10.1016/j.procs.2018.08.151.
[15] Postgresql, "Table Partitioning," Postgresql, 2022. https://www.postgresql.org/docs/current/ddl-partitioning.html (accessed Sep. 16, 2022).
[16] D. Vashi, H. B. Bhadka, K. Patel, and S. Garg, "An Efficient Hybrid Approach of Attribute Based Encryption for Privacy Preserving Through Horizontally Partitioned Data," Procedia Comput Sci, vol. 167, no. 2019, pp. 2437–2444, 2020, doi: 10.1016/j.procs.2020.03.296.
[17] A. Dahal and S. R. Joshi, "A Clustering Based Vertical Fragmentation and Allocation of a Distributed Database," in 2019 Artificial Intelligence for Transforming Business and Society (AITB), IEEE, Nov. 2019, pp. 1–5. doi: 10.1109/AITB48515.2019.8947444.
[18] A. JMeter, "Apache JMeter," Apache JMeter, 2022. https://jmeter.apache.org/ (accessed Oct. 02, 2022).