Analisis Performa Sistem pada Pemisahan Database Analitik dengan Transaksional

I Made Ari Madya Santosa[a1], I Gusti Ngurah Anom Cahyadi Putra[a2]

[a]Program Studi Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Udayana
Jalan Raya Kampus Udayana, Bukit Jimbaran, Kuta Selatan, Badung, Bali Indonesia
[1]arimadyasantosa020@student.unud.ac.id
[2]anom.cp@unud.ac.id

Abstract

The speed and efficiency of a system currently has a very important role. Currently there are several solutions to answer these challenges, one of which is to separate the database on the system. In this study, an analysis was carried out on the separation of databases on a transactional and analytic system. Analysis is carried out with a research flow that begins with problem identification followed by sampling data on technology companies that have transactional systems, then implementing transactional and analytical programs on microservices architecture, conducting tests with Load-Testing, followed by analysis of test results, and then drawing conclusions. After conducting research in accordance with the research flow, it was concluded that the separation of databases on transactional and analytic systems is better for producing faster system performance compared to transactional and analytic systems using the same database.

Keywords: *Big Data, Optimization, Transactional, Analytical Systems, Microservices*

1. Pendahuluan

Salah satu tantangan dalam sistem informasi saat ini adalah kinerja sistem yang cepat dan efisien. Untuk menyelesaikan masalah ini, pemrosesan data terdistribusi menjadi salah satu solusi yang banyak digunakan [1]. Terdapat dua tipe pemrosesan data dalam sistem distribusi yaitu Transactional Processing (OLTP) dan Analytical Processing (OLAP) yang merupakan cara pemrosesan yang berbeda namun saling melengkapi. OLTP digunakan untuk memproses operasi transaksi bisnis, sedangkan OLAP digunakan untuk analisis dan pengambilan keputusan dengan menggunakan data historis.

Pada saat ini, terdapat beberapa solusi untuk mengintegrasikan OLTP dan OLAP pada sistem yang sama, salah satunya adalah dengan menggunakan pemisahan servis dan database [2]. Pemisahan database memungkinkan pengelolaan data yang lebih efisien, sementara pemisahan servis memungkinkan pengelolaan servis yang lebih efektif. Oleh karena itu, implementasi pemisahan servis dan database analitik dengan transaksional menjadi salah satu solusi yang paling populer saat ini [3].

Namun, penggunaan database terdistribusi juga memiliki beberapa tantangan. Masalah utama dalam penggunaan database terdistribusi adalah konsistensi data dan skalabilitas [4]. Oleh karena itu, metode konsistensi data yang tepat harus digunakan untuk memastikan bahwa data yang tersedia dalam database terdistribusi konsisten dan akurat. Selain itu, sistem database terdistribusi juga harus dapat mengatasi masalah skalabilitas untuk memastikan bahwa sistem dapat menangani volume data yang besar dan meningkat seiring waktu.

Dalam kaitannya dengan performa sistem, pemisahan database dan servis juga dapat membantu dalam mengoptimalkan kinerja sistem. Dalam sebuah penelitian oleh Li et al. (2019), mereka menunjukkan bahwa pemrosesan transaksional dan analitis pada database terdistribusi dapat meningkatkan kinerja sistem, khususnya dalam hal throughput dan latensi [3]. Kim et al. (2019)

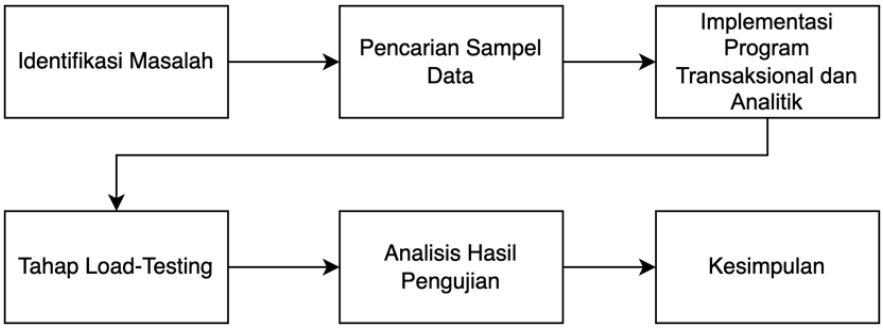
juga melakukan penelitian yang menunjukkan bahwa arsitektur database terdistribusi dapat membantu mengintegrasikan pemrosesan transaksional dan analitis [4].

Dalam rangka memaksimalkan performa sistem, diperlukan juga penggunaan database yang tepat. Widiastuti dan Adhita (2019) mengemukakan bahwa penggunaan database dapat mempengaruhi performa sistem informasi [5]. Oleh karena itu, pemilihan database yang tepat menjadi kunci penting dalam mengoptimalkan performa sistem.

2. Metode Penelitian

2.1. Alur Penelitian

Penulis mengikuti alur dan juga tahapan yang ditunjukan pada **Gambar 1** untuk emastikan kelancaran dari penelitian ini



Gambar 1. Alur penelitian

Penjelasan dari alur penelitian di atas sebagai berikut.

a. Identifikasi Masalah

Pada awal penelitian, penulis melakukan identifikasi permasalahan di lapangan dan juga melakukan penelitian dengan menggunakan literatur yang dapat membantu penelitian. Kemudian, masalah yang ditemukan oleh penulis adalah lambatnya pengambilan data pada database dan tingginya penggunaan CPU dan RAM ketika melakukan transaksional dan juga analitik terhadap suatu layanan.

b. Pencarian Sampel Data

Pada tahap ini, sampel data dicari melalui survey terhadap perusahaan teknologi yang memiliki produk berupa sistem transaksional. Dilakukan observasi dan wawancara untuk mengetahui bagaimana bentuk data dari sistem transaksional yang dimiliki oleh perusahaan terkait.

c. Implementasi Program Transaksional dan Analitik Menggunakan Arsitektur Microservices

Pada tahap ini, penulis membuat sebuah sistem untuk melakukan transaksional dan analitik. Terdapat satu servis untuk melakukan transaksional dan satu servis analitik serta penulis juga membuat dua database dengan data yang sama.

d. Tahap Load-Testing

Pengujian dilakukan dengan metode *Load-Testing* menggunakan K6 dan memperhatikan beberapa metris.

e. Analisis Hasil Pengujian

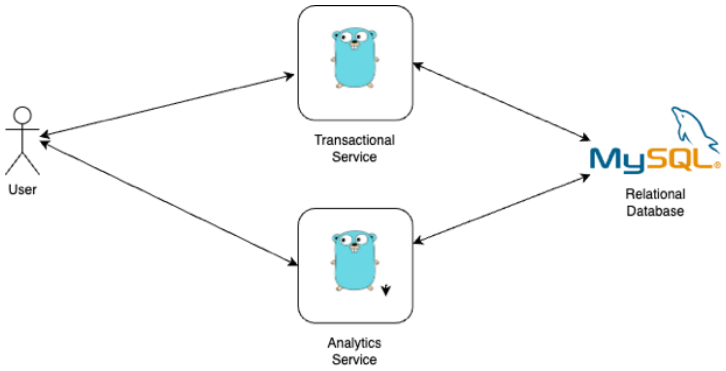
Peneliti melakukan analisis terhadap beberapa metris dan hasil database monitoring untuk menarik sebuah kesimpulan.

f. Kesimpulan

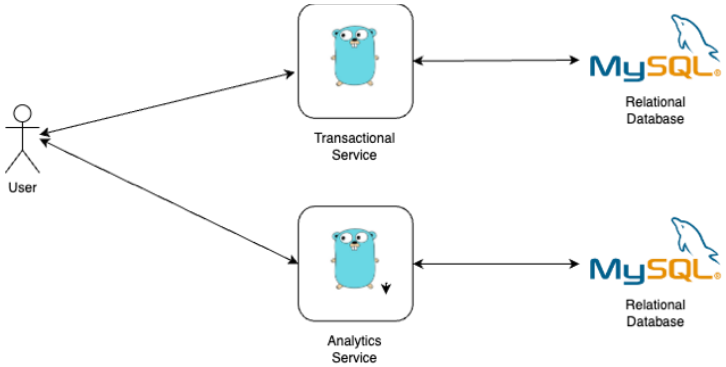
Pada tahap ini, penulis menarik kesimpulan dari beberapa metris dan hasil database monitoring untuk menarik kesimpulan dari hasil penelitian ini.

2.2. Arsitektur Microservices

Penelitian ini menggunakan arsitektur *microservices* untuk melihat dampak dari layanan *single-point-of-failure*, atau layanan yang kegagalannya mengganggu pengoperasian sistem. Ini adalah masalah fatal dalam arsitektur layanan mikro. *Microservices* adalah model arsitektur [6]. Ide dasar di balik jenis arsitektur ini adalah membagi sistem yang lebih besar menjadi sistem yang lebih kecil. Hal ini dapat mengurangi kompleksitas sistem [7], membuat sistem yang lebih kecil menjadi lebih mandiri, dan membuat hubungan antara sistem atau layanan menjadi kurang rumit. Selain itu, sistem yang mandiri membutuhkan proses pengembangan dan implementasi yang lebih mandiri. Keuntungan lain dari microservices adalah jika suatu layanan gagal, kegagalan tersebut hanya diisolasi dan tidak mengganggu layanan lain, kecuali jika layanan tersebut merupakan kegagalan tunggal. **Gambar 2** dan **Gambar 3** menunjukkan arsitektur yang penulis gunakan dalam penelitian ini,



Gambar 2. Arsitektur Microservices 1



***Gambar 3**. Arsitektur Microservices 2*

2.3. Load-Testing

Pengujian yang dilakukan pada penelitian ini menggunakan metode *Load-Testing* dengan bantuan K6. Metode *Load-Testing* akan mensimulasikan pengguna yang mengirim permintaan sesuai dengan parameter yang diberikan ke program [8]. K6 merupakan alat bantu untuk melakukan *Load-Testing* yang bersifat *open-source.* Dengan ini kita bisa menguji performa dan keandalan sistem untuk mencari tahu masalah lebih awal. K6 akan mengembalikan hasil berupa beberapa metris yang berkaitan tentang permintaan yang telah dikirim. Metris ini akan membantu penulis untuk mengukur kecepatan penerimaan data pada penelitian ini. Beberapa metris yang digunakan pada penelitian dapat dilihat pada **Tabel 1**

Tabel 1. Metris Load-Testing

Key	Deskripsi
http_req_duration	Metris yang merepresentasikan durasi dari permintaan hingga menerima data pada HTTP Request (http_req_sending + http_req_waiting + http_req_receiving)

3. Hasil dan Diskusi

Berdasarkan alur penelitian dan juga studi literatur yang telah dijelaskan pada bab sebelumnya. Maka hasil penelitian yang peneliti lakukan dapat dijabarkan sebagai berikut.

3.1. Pencarian Sampel Data

Peneiltian ini melakukan pencarian sampel data ke perushaan teknologi yang memiliki sistem transaksional. Sampel data dicari dengan cara melakukan survey dan observasi terhadap perusahaan terkait. Didapatkan format data pada tabel transaksi yang digunakan oleh perusahaan dalam sistem transaksioal yang dapat dilihat pada **Tabel 2**

Tabel 2. Format Data Tabel Transaksi

Column	Type
transaction_no	Int
date	Varchar
product_no	Int
product_name	Varchar
price	Double
quantity	Int
customer_no	Int
country	varchar

3.2. Implementasi Program Transaksional dan Analitik Menggunakan Arsitektur Microservices

Proses implementasi program transaksional dan analitik dilakukan pada basis kode atau *codebase* berbeda dengan menggunakan bahasa pemrograman Go. Potongan kode inti dari program transaksional dapat dilihat pada **Tabel 3** dan analitik pada **Tabel 4.**

Tabel 3. Potongan Kode Inti Transaksional

Baris	Kode
67	`func createTransaksi(c *gin.Context) {`
68	` var transaksi Transaksi`
69	
70	` if err := c.ShouldBindJSON(&transaksi); err != nil {`
71	`  c.JSON(http.StatusBadRequest, gin.H{"error":`
72	`err.Error()})`
	`  return`
73	` }`
74	
75	` if err := db.Create(&transaksi).Error; err != nil {`
76	`  c.JSON(http.StatusInternalServerError,`
	`gin.H{"error": err.Error()})`
77	`  return`
78	` }`
79	
80	` c.JSON(http.StatusOK, transaksi)`
81	`}`
82	
83	`func getTransaksi(c *gin.Context) {`
84	` id := c.Param("id")`
85	` var transaksi Transaksi`
86	
87	` if err := db.First(&transaksi, id).Error; err != nil {`
88	`  if errors.Is(err, gorm.ErrRecordNotFound) {`
89	`   c.JSON(http.StatusNotFound, gin.H{"error":`
	`"record not found"})`
90	`  } else {`
91	`   c.JSON(http.StatusInternalServerError,`
	`gin.H{"error": err.Error()})`
92	`  }`
93	`  return`
94	` }`
95	
96	` c.JSON(http.StatusOK, transaksi)`
97	`}`
98	
99	`func updateTransaksi(c *gin.Context) {`
100	` id := c.Param("id")`
101	` var transaksi Transaksi`
102	
103	` if err := db.First(&transaksi, id).Error; err != nil {`
104	`  if errors.Is(err, gorm.ErrRecordNotFound) {`
	`   c.JSON(http.StatusNotFound, gin.H{"error":`
105	`"record not found"})`
	`  } else {`
106	`   c.JSON(http.StatusInternalServerError,`
107	`gin.H{"error": err.Error()})`
	`  }`
108	`  return`
109	` }`
110	
111	` if err := c.ShouldBindJSON(&transaksi); err != nil {`
112	`  c.JSON(http.StatusBadRequest, gin.H{"error":`
113	`err.Error()})`
	`  return`

Baris	Kode
114	` }`
115	
116	` if err := db.Save(&transaksi).Error; err != nil {`
117	`  c.JSON(http.StatusInternalServerError,`
118	`gin.H{"error": err.Error()})`
	`  return`
119	` }`
120	
121	` c.JSON(http.StatusOK, transaksi)`
122	`}`
123	
124	`func deleteTransaksi(c *gin.Context) {`
125	`    db := c.MustGet("db").(*gorm.DB)`
126	
127	`    // Get the transaksi record with the specified ID`
128	`    var transaksi Transaksi`
129	`    if err := db.Where("id = ?",`
130	`c.Param("id")).First(&transaksi).Error; err != nil {`
	`        c.JSON(http.StatusNotFound, gin.H{"error": "Record not`
131	`found!"})`
	`        return`
132	`    }`
133	
134	`    // Delete the transaksi record from the database`
135	`    db.Delete(&transaksi)`
136	
137	`    // Return a success message as JSON`
138	`    c.JSON(http.StatusOK, gin.H{`
139	`        "message": "Record deleted successfully!",`
140	`    })`
141	`}`
142	

Tabel 4. Potongan Kode Inti Analitik

Baris	Kode
70	`func getRevenueByProduct(c *gin.Context) {`
71	` var revenueByProduct []RevenueByProduct`
72	
73	` if err :=`
	`db.Table("transaksis").Select("product_no, sum(price *`
	`quantity) as`
74	`revenue").Group("product_no").Find(&revenueByProduct).Err`
	`or; err != nil {`
75	`  c.JSON(http.StatusInternalServerError,`
76	`gin.H{"error": err.Error()})`
77	`  return`
78	` }`
79	
80	` c.JSON(http.StatusOK, revenueByProduct)`
81	`}`
82	
83	
84	`func getRevenueByDate(c *gin.Context) {`
	` var revenueByDate []struct {`

Baris	Kode
85	Date string `json:"date"`
86	Revenue float64 `json:"revenue"`
87	}
88	if err := db.Table("transaksis").Select("date, sum(price * quantity) as revenue").Group("date").Find(&revenueByDate).Error; err
89	!= nil {
90	c.JSON(http.StatusInternalServerError,
91	gin.H{"error": err.Error()})
92	return
93	}
	c.JSON(http.StatusOK, revenueByDate)
	}

Pada **Tabel 3** merupakan kode program untuk melakukan transaksi pada umumnya seperti membuat transaksi, mengubah transaksi, melihat transaksi, dan juga menghapus transaksi. Kemudian pada kode analitik di **Tabel 4**, analitik yang dilakukan adalah menghitung pendapatan berdasarkan tanggal dan berdasarkan produk.

3.3. Tahap Load-Testing

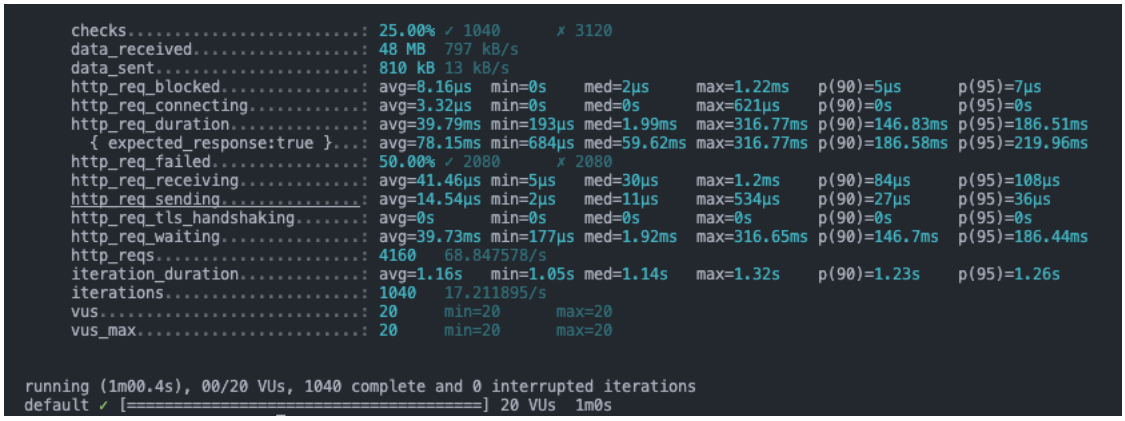
Pengujian dari sistem ini adalah dengan melakukan *Load-Testing* menggunakan K6. Program untuk melakukan *Load-Testing* ditulis dengan menggunakan bahasa pemrograman *JavaScript.* Kode pengujian menggunakan K6 dapat dilihat pada kode di **Tabel 5.**

Table 5. Kode Load-Testing

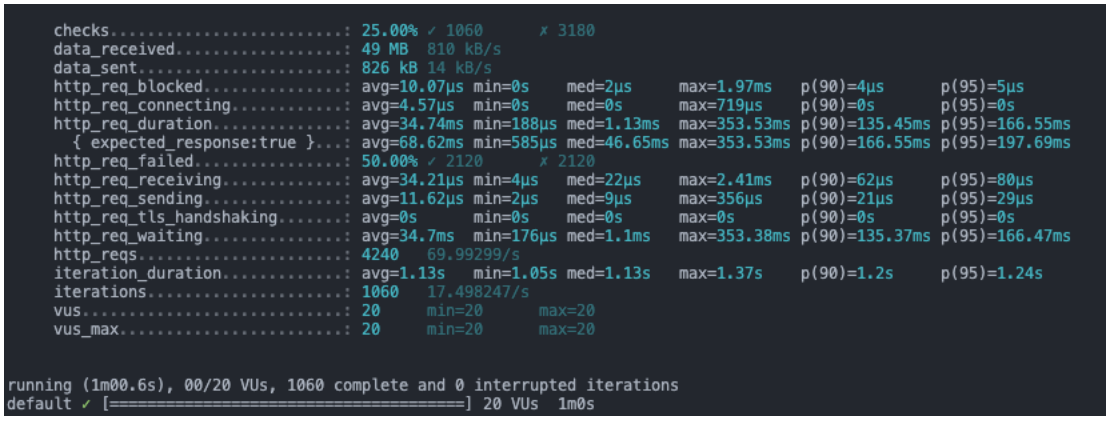
Baris	Kode
1	import http from 'k6/http';
2	import { check, sleep } from 'k6';
3	
4	export let options = {
5	stages: [
6	{ duration: '1m', target: 20 }
7	]
8	};
9	
10	export function testCreateTransaksi() {
11	let payload = {
12	country: 'Indonesia',
13	customer_no: 1,
14	date: '2023-04-24',
15	price: 100.0,
16	product_name: 'Product A',
17	product_no: 1,
18	quantity: 1,
19	transaction_no: 1
20	};
21	
22	let headers = { 'Content-Type': 'application/json' };
23	
24	

Baris	Kode
	`    let res = http.post('http://localhost:8081/transaksi',`
25	`JSON.stringify(payload), { headers: headers });`
26	
27	`    check(res, { 'status is 201': (r) => r.status === 201 });`
28	`}`
29	
30	`export function testGetProductRevenue() {`
	`    let res =`
	`http.get('http://localhost:8080/transaksi/revenue-by-`
31	`product');`
32	
33	`    check(res, { 'status is 200': (r) => r.status === 200 });`
34	`}`
35	
36	`export function testUpdateTransaksi() {`
37	`    let payload = {`
38	`        country: 'Indonesia',`
39	`        customer_no: 1,`
40	`        date: '2023-04-24',`
41	`        price: 200.0,`
42	`        product_name: 'Product A',`
43	`        product_no: 1,`
44	`        quantity: 2,`
45	`        transaction_no: 1`
46	`    };`
47	
48	`    let headers = { 'Content-Type': 'application/json' };`
49	
	`    let res = http.put('http://localhost:8081/transaksi/1',`
50	`JSON.stringify(payload), { headers: headers });`
51	
52	`    check(res, { 'status is 200': (r) => r.status === 200 });`
53	`}`
54	
55	`export function testDeleteTransaksi() {`
56	`    let res = http.del('http://localhost:8081/transaksi/1');`
57	
58	`    check(res, { 'status is 200': (r) => r.status === 200 });`
59	`}`
60	
61	`export default function () {`
62	`    // Create a new transaksi record`
63	`    testCreateTransaksi();`
64	
65	`    // Get the total revenue by product`
66	`    testGetProductRevenue();`
67	
68	`    // Update an existing transaksi record`
69	`    testUpdateTransaksi();`
70	
71	`    // Delete an existing transaksi record`
72	`    testDeleteTransaksi();`
73	
74	`    // Wait for 1 second before repeating the cycle`
75	`    sleep(1);`
	`}`

Implementasi dari *Load-Testing* dapat dilihat pada penggalan kode di atas. Pada kode tersebut terdapat beberapa fungsi yang menjalankan HTTP request ke beberapa endpoint untuk melakukan testing fungsionalitas dari API. Terdapat fungsi untuk membuat transaksi baru, mengambil total revenue dari produk, mengubah transaksi yang sudah ada, dan menghapus transaksi yang sudah ada. Selain itu, terdapat fungsi default yang menjalankan seluruh fungsi di atas secara berurutan dan akan diulang setiap 1 detik hingga durasi yang diatur pada options.stages habis. Pada options.stages, terdapat satu tahap (stage) pengujian yang mengatur bahwa selama 1 menit, akan ada 20 koneksi yang dibuat ke aplikasi. Dalam setiap HTTP request yang dilakukan, juga terdapat pengecekan (check) apakah status response dari API sesuai dengan yang diharapkan, misalnya status 201 untuk create atau 200 untuk read/update/delete. Sehingga didapatkan hasil dari *Load-Testing* seperti pada **Gambar 4** dan juga **Gambar 5** di bawah.



Gambar 4. Hasil Testing Dengan Satu Database



Gambar 5. Hasil Testing Dengan Database Berbeda

3.4. Analisis Hasil Pengujian

Dalam pengukuran kinerja sistem transaksional dan analitik dengan database yang sama dan berbeda pada penelitian ini, dapat menggunakan perbandingan hasil dari *Load-Testing* pada kedua metode tersebut. Perbandingan dapat dilihat dengan mudah pada **Tabel 6.**

Tabel 6. Hasil metris Load-Testing

Metris	Sistem Dengan Satu Database	Sistem Dengan Database Berbeda	Performa Tertinggi
http_req_duration	186.51 milidetik	166.55 milidetik	Sistem Dengan Database Berbeda

Bedasarkan tabel diatas dapat dilihat metris *http_req_duration* menggunakan p (95) atau persentase *request* pada kecepatan yang jatuh pada persentase ke-95 agar lebih tepat dan menghindari *outlier* pada kedua hasil *Load-Testing*, Sistem transaksional dan analitik dengan menggunakan *database* yang berbeda mendapatkan hasil yang lebih baik dengan jangka waktu dari *request* hingga menerima data lebih cepat dibandingkan dengan satu *database*.

4. Kesimpulan

Berdasarkan metris *http_req_duration* menggunakan persentase *request* pada kecepatan yang jatuh pada persentase ke-95 atau p (95) dari hasil pengujian penelitian yang dilakukan yang dapat dilihat pada **Tabel 6**, ditemukan bahwa pemisahan *database* untuk sistem transaksional dan analitik memiliki kecepatan sistem yang lebih cepat dibanding sistem transaksional dan analitik hanya menggunakan satu *database*. Sehingga dapat direkomendasikan untuk pemisahan *database* pada sistem transaksional dan analitik, namun diperlukan juga sinkronisasi *database* agar data yang digunakan untuk transaksional dan analitik sama.

Daftar Pustaka

[1] Jafarzadeh, A., & Alizadeh, A. (2018). Design and implementation of a high-performance software-defined storage system for big data applications. IEEE Access, 6, 29934-29948.
[2] Chen, Y., Zhang, X., Hu, F., Guo, M., & Yan, X. (2019). An Automatic Data Separation Method for Analytical and Transactional Services. IEEE Access
[3] Li, J., Li, C., Li, L., Xu, X., & Feng, Y. (2019). Performance evaluation of transactional and analytical processing on distributed column-oriented database. IEEE Access, 7, 20934-20944.
[4] Kim, T., Lee, Y., Lee, S., Lee, S., & Song, J. (2019). A distributed database architecture for integrating transactional and analytical data processing. Future Generation Computer Systems, 99, 205-215.
[5] Widiastuti, I., & Adhita, D. (2019). Pengaruh Penggunaan Database Terhadap Performa Sistem Informasi. Journal of Information Technology and Computer Science, 4(1), 1-7.
[6] F. Rademacher, S. Sachweh, and A. Zundorf, “Differences between model-driven development of service-oriented and microservice architecture,” *Proc. - 2017 IEEE Int. Conf. Softw. Archit. Work. ICSAW 2017 Side Track Proc.*, pp. 38–45, 2017, doi: 10.1109/ICSAW.2017.32.
[7] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*, vol. 5, no. March, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.
[8] H. Schulz, T. Angerstein, and A. Van Hoorn, “Towards automating representative load testing in continuous software engineering,” *ICPE 2018 - Companion 2018 ACM/SPEC Int. Conf. Perform. Eng.*, vol. 2018-January, pp. 123–126, 2018, doi: 10.1145/3185768.3186288.