Hello World
Jurnal Ilmu Komputer
https://jurnal.ilmubersama.com/index.php/hello_world

Klik disini untuk menuliskan kategori naskah

ANALISIS KOMPARASI THROUGHPUT DAN RESOURCE UTILIZATION (CPU & RAM) ANTARA POSTGRESQL DAN MONGODB PADA DATA LOG AKTIVITAS E-LEARNING MENGGUNAKAN ARSITEKTUR CONCURRENT GOLANG

Muhammad Nazhadidin Hasan [1*], *Mira Maisura* [2]

[1,2] *Program Studi Pendidikan Teknologi Informasi, Universitas Islam Negeri Ar-Raniry*
Jalan Syeikh Abdur Rauf Kopelma Darussalam, Banda Aceh, Indonesia

INFORMASI ARTIKEL

Diterima Redaksi: 00 Januari 00
Revisi Akhir: 00 Februari 00
Diterbitkan *Online*: 00 Maret 00

KATA KUNCI

MongoDB, PostgreSQL, Throughput, Resource Utilization, E-learning.

KORESPONDENSI

Phone: +6282277544381

E-mail: 220212072@student.ar-raniry.ac.id

A B S T R A K

Peningkatan volume data log aktivitas e-learning yang bersifat *write-intensive* memicu tantangan performa pada server. Penelitian ini menggunakan metode experimental kuantitatif. yang membandingkan kinerja *throughput*, error Rate, penggunaan sumber daya (CPU dan RAM) antara *PostgreSQL* versi 18.0 dan *MongoDB* versi 8.0 di bawah tekanan konkurensi. Pengujian ini menggunakan load generator kustom berbasis concurrent *Golang* versi 1.24 (Goroutines) tanpa connection pooling guna mengevaluasi batas ketahanan kedua database. Pengujian dijalankan secara berjenjang pada skenario 500 - 10.000 workers (Goroutines) di dalam Docker dengan pembatasan sumber daya. Hasil pengujian menunjukkan *MongoDB* secara konsisten mengungguli *PostgreSQL* signifikan. Pada beban puncak 10.000 workers, *MongoDB* mencapai *throughput* sebesar 12.532,26 TPS (47,42x lebih cepat dibanding PostgreSQL yang hanya 264,30 TPS) dengan success Rate 100,00% (0% error Rate). Sebaliknya, *PostgreSQL* mengalami breaking point mulai dari skenario 1.000 workers (error Rate 21,13%) hingga hancur pada 10.000 workers dengan error Rate 81,40% akibat connection exhaustion. Dari sisi efisiensi sumber daya, arsitektur multi-threaded *MongoDB* menjaga alokasi RAM stabil pada kisaran ~180 MB dengan CPU produktif 108,78%, sedangkan model process-per-connection *PostgreSQL* memicu pembengkakan RAM hingga 1.008,19 MB dan saturasi CPU hingga 518,00% akibat switching operasi. Penelitian memberikan rekomendasi menerapkan segregasi database pada sistem e-learning kampus dengan mendelegasikan *MongoDB* sebagai wadah log aktivitas mandiri.

PENDAHULUAN

Pendidikan tinggi saat ini sangat bergantung pada teknologi digital. Hampir semua kampus menggunak-an *Learning Management System (LMS)* atau platform e-learning untuk mendukung kegiatan belajar mengajar[1]. Seiring dengan penggunaannya yang terus meni-ngkat setiap hari, platform ini menghasilkan tumpukan data yang sangat besar. Data ini berasal dari catatan aktivitas mahasiswa, mulai dari absen masuk, mengu-nduh materi, mengerjakan kuis, hingga mengunggah tugas[2]. Rekam jejak digital inilah yang disebut dengan data log aktivitas.

Pengelolaan data log dalam jumlah raksasa ini memunculkan masalah teknis yang serius. Karakteristik data log sangat berbeda dengan data akademik biasa (seperti data nama mahasiswa atau nilai akhir). Data log bersifat *write-intensive* (sistem lebih sering 'menulis' atau memasukkan data baru daripada 'membaca' data lama) dan volumenya bertambah sangat cepat[3]. Jika data *log* ini dicampur penyimpanannya dengan data akademik biasa di server yang sama, beban kerja

https://doi.org/10.12345/xxxxx Attribution-ShareAlike 4.0 International Some rights reserved

database akan menjadi sangat berat. Akibatnya, kinerja server kampus bisa melambat, sering *error* (latensi tinggi), atau bahkan lumpuh total (*downtime*) pada saat jam-jam sibuk[4].

Kondisi server yang macet (*bottleneck*) ini paling sering terjadi ketika banyak mahasiswa mengakses *LMS* secara bersamaan (*concurrent*), misalnya saat menit pertama ujian daring dimulai atau saat pembukaan pengisian *KRS*. Untuk mengatasi masalah ini, para ahli arsitektur perangkat lunak menyarankan pemisahan tempat penyimpanan data log. Muncul perdebatan mengenai teknologi database mana yang paling tangguh untuk menampung log yang masif ini. Ada dua jenis database utama yang sering dibandingkan: database relasional (*RDBMS*) seperti *PostgreSQL* dan database NoSQL (berbasis dokumen) seperti *MongoDB*[5].

PostgreSQL sangat andal dalam menjaga keakuratan dan keamanan data karena memiliki sistem relasi tabel yang ketat. Namun, aturan yang ketat ini membutuhkan tenaga komputasi (*CPU*) yang besar. *MongoDB* sangat fleksibel dan cepat karena tidak menggunakan tabel kaku, melainkan menyimpan data dalam bentuk dokumen (seperti file BSON/JSON). *MongoDB* sangat cocok untuk data yang tidak terstruktur, namun cara kerjanya cenderung banyak memakan memori (*RAM*)[6].

Penelitian terdahulu menunjukkan bahwa performa *MongoDB* dan *PostgreSQL* pada operasi penulisan dipengaruhi oleh karakteristik *workload* dan ukuran data. Evaluasi performa tidak hanya perlu mempertimbangkan *throughput* dan waktu respons, tetapi juga penggunaan sumber daya seperti CPU dan RAM untuk memberikan gambaran efisiensi database[7]. Sementa-ra itu, Olszak dan Skublewska-Paszkowska[6] menunjukkan bahwa *PostgreSQL* dan *MongoDB* memiliki karakteristik performa yang berbeda bergantung pada jenis operasi dan beban kerja yang diberikan. Penelitian tersebut membandingkan kedua basis data melalui berbagai operasi dan ukuran dataset untuk mengevaluasi waktu pemrosesan serta variasi performanya. Namun, penelitian tersebut belum secara khusus menguji skenario konkurensi ekstrem menggunakan data semi-terstruktur berbasis JSON yang merepresentasikan aktivitas LMS perguruan tinggi.

Penelitian lain juga[4] mengonfirmasi bahwa basis data NoSQL (Mon-goDB) memberikan *throughput* penulisan data log yang lebih tinggi dibandingkan SQL pada aplikasi IoT yang dinamis. Meskipun demikian, penelitian tersebut masih belum memaparkan analisis mendalam terhadap konsumsi memori dan CPU secara realtime, serta tidak menguji hingga titik kehancuran sistem (breaking point). Terakhir, Wisal Khan dkk[8] membuktikan bahwa arsitektur NoSQL memberikan latensi dan *throughput* yang lebih stabil pada volume transaksi tinggi di lingkungan microservices. Namun, kelemahan penelitian ini terletak pada penggunaan load generator eksternal berbasis Java (JMeter) yang justru mengonsumsi RAM sangat tinggi di sisi klien, sehingga berpotensi mengganggu akurasi pengukuran performa murni database .

Berdasarkan celah penelitian (research gap) dari kajian literatur tersebut, diketahui bahwa belum ada penelitian yang secara keseluruhan menguji ketahanan *PostgreSQL* dan *MongoDB* secara *head-to-head* pada skenario data *log e-learning* yang bersifat *write-intensive* dengan beban konkuren ekstrem hingga 10.000 koneksi, sembari mengukur dampaknya terhadap konsumsi *CPU* dan *RAM* secara mendalam. Untuk mengetahui database mana yang benar-benar lebih kuat pada skenario tersebut, perlu dilakukan pengujian beban ekstrem (*Stress testing*) dengan simulasi yang adil dan presisi. Penelitian ini menggunakan bahasa pemrograman Go (*Golang*) untuk mengisi kekosongan tersebut. *Golang* memiliki fitur Goroutines yang mampu menyimulasikan puluhan ribu koneksi pengguna secara bersamaan dengan sangat ringan. *Golang* tidak akan membebani RAM komputer penguji, sehingga hasil uji kecepatan (*throughput*) dan beban *CPU/RAM* yang tercatat murni merupakan performa dari database (*PostgreSQL* dan *MongoDB*) itu sendiri.

Berdasarkan latar belakang tersebut, rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana perbandingan *throughput* (TPS) dan *error rate* antara *PostgreSQL* 18.0 dan *MongoDB* 8.0 pada operasi penulisan data *log* aktivitas *e-learning* di bawah beban konkuren bertingkat (500, 1.000, 5.000, dan 10.000 *workers*) tanpa *connection pooling*?
2. Bagaimana perbandingan efisiensi penggunaan sumber daya (*CPU* dan *RAM*) antara *PostgreSQL* 18.0 dan *MongoDB* 8.0 pada kondisi pengujian yang sama?

Sejalan dengan rumusan masalah tersebut, tujuan penelitian ini adalah:

1. Menganalisis dan membandingkan *throughput* (TPS) dan *error rate PostgreSQL* 18.0 dan *MongoDB* 8.0 pada beban penulisan data *log e-learning* dengan konkurensi 500 hingga 10.000 *workers*, termasuk mengidentifikasi titik kegagalan (*breaking point*) masing-masing *database*.
2. Menganalisis dan membandingkan penggunaan CPU dan RAM kedua *database* selama pengujian beban untuk menentukan efisiensi sumber daya masing-masing.

 https://doi.org/10.12345/xxxxx

Hasil penelitian diharapkan menjadi dasar pertimbangan bagi tim *IT* perguruan tinggi dalam memilih teknologi *database* untuk menyimpan data *log* aktivitas *e-learning* yang bersifat *write-intensive*.

TINJAUAN PUSTAKA

Sistem E-Learning dan Managemen LOG Aktifitas

Learning Management System (LMS) adalah platform teknologi informasi terpadu yang memfasilitasi proses pembelajaran perguruan tinggi secara asinkron maupun sinkron di ranah digital. Setiap aktivitas pengguna di dalamnya terekam secara otomatis menjadi data *log*. Karakteristik teknis data *log* aktivitas *e-learning* meliputi tiga hal utama: (1) *append-only*, yaitu data bersifat riwayat historis yang terus bertambah tanpa pernah diubah; (2) *write-intensive*, di mana hingga 90% operasi basis data didominasi oleh penulisan (*insert*); dan (3) semi-terstruktur, dengan format topologi data berbasis dokumen JSON[1].

Arsitektur Relational Database Management System (RDBMS) – PostgreSQL

PostgreSQL merupakan *RDBMS open-source* yang matang dan berkinerja tinggi. Karakteristik arsitektural utama *PostgreSQL* terletak pada pendeka-tan pengelolaan koneksinya yang mempertah-ankan model *process per connection*, yaitu setiap koneksi klien ditangani oleh satu proses sistem operasi terpisah. Mekanisme ini bekerja melalui komponen Postmaster Process yang melakukan forking untuk membentuk proses baru setiap kali terdapat permintaan koneksi baru[8]. Pendekatan ini membe-rikan isolasi proses yang sangat kuat, namun memiliki konsekuensi berupa konsumsi memori yang besar ketika server harus menangani koneksi konkuren dalam jumlah sangat tinggi. Selain itu, *PostgreSQL* menjaga konsistensi transaksi melalui mekanisme *Write-Ahead Logging* (*WAL*) yang secara sinkron menuliskan perubahan data ke media penyimpanan fisik sebelum melaporkan status sukses, yang mana dapat memicu *bottleneck I/O* pada beban kerja *write-heavy*[8].

Arsitektur Document-oriented Database (NoSQL) – MongoDB

MongoDB merupakan sistem basis data NoSQL modern yang mengadopsi pendekatan *document-oriented*, di mana data disimpan dalam bentuk dokumen fleksibel berbasis BSON (*Binary JSON*). *MongoDB* dirancang menggunakan arsitektur pemrosesan *multi-threaded* dalam satu proses *daemon* utama *mongod*. Setiap koneksi klien yang masuk dikelola melalui mekanisme *thread pool*, sehingga penanganan koneksi jauh lebih ringan dibanding model *process-per-connection PostgreSQL*. Keunggulan lain *MongoDB* terletak pada mesin penyimpanan *WiredTiger* yang mengoptimalkan pemanfaatan memori melalui *cache* internal dan *memory mapping*, di mana sebagian besar RAM kosong dimanfaatkan sebagai *cache* untuk mempercepat operasi *bulk insert* secara asinkron sebelum dituliskan ke *disk* secara berkala melalui *checkpointer*[9].

Pemograman Konkuren Goroutines pada GO

Bahasa pemrograman *Golang* dikembangkan oleh Google dengan fokus pada efisiensi komputasi konkurensi berskala besar. Mekanisme konkurensi *Golang* menggunakan fitur Goroutine, yaitu unit eksekusi ringan yang dikelola langsung oleh *runtime Golang* melalui Go *Scheduler* dengan pendekatan pemetaan *scheduling*. Ribuan hingga jutaan *Goroutine* dapat berjalan secara efisien di atas sejumlah kecil thread sistem operasi. Berbeda dengan thread tradisional yang membutuhkan memori besar, *Goroutine* dirancang dengan ukuran stack awal yang sangat kecil (~2 KB) dan dapat bertumbuh secara dinamis, sehingga sangat ideal digunakan untuk membuat instrumen pengujian beban (*load generator*) berskala tinggi[10].

Teknik Stress testing

Stress testing merupakan metodologi rekayasa keandalan sistem (*Site Reliability Engineering* / *SRE*) yang bertujuan membebani batas daya tahan sistem perangkat lunak melampaui kondisi operasional normalnya. Fokus utamanya adalah untuk mengobservasi titik kegagalan (*failure point*), saturasi sumber daya (*resource saturation*), serta kemampuan sistem untuk pulih kembali setelah beban ekstrem berakhir[7].

Metrik Pengujian Kinerja Sistem

Parameter evaluasi kinerja pada penelitian komparatif ini difokuskan pada tiga metrik utama: 1. *Throughput* (TPS): Jumlah transaksi atau operasi ba-sis data *INSERT* yang berhasil diproses oleh sistem dalam satu detik (Transactions per Second). 2. CPU Utilization (%): Persentase penggunaan sumber daya prosesor oleh proses database (postgres atau mongod). 3 RAM Utilization (MB): Jumlah penggunaan memori fisik (RAM) oleh sistem basis data selama pengujian berlangsung[11].

METODOLOGI

Jenis Penelitia

Penelitian ini pendekatan Eksperimental Kuantitatif Komparatif. Dalam rekayasa perangkat lunak, penelitian eksperimental melibatkan manipulasi terhadap satu atau lebih variabel bebas untuk mengobservasi efeknya terhadap variabel terikat secara terukur dan dapat direplikasi. Variabel bebas dalam penelitian ini adalah jenis basis data (*PostgreSQL* dan *MongoDB*) serta tingkat beban konkurensi (500, 1.000, 5.000, dan 10.000 workers). Variabel terikat yang diukur meliputi *Throughput* (TPS), CPU *Utilization* (%), dan RAM *Utilization* (MB)[8].

Lokasi dan Lingkungan Penelitian

Penelitian sepenuhnya dilaksanakan pada peladen lokal (*localhost*) untuk mengeliminasi variabel pengganggu eksternal seperti *network jitter* dan ketidakstabilan *bandwidth*. Kedua basis data dijalankan dalam *Docker container* terpisah dengan pembatasan sumber daya yang identik menggunakan parameter *--cpus* dan *--memory* untuk menjamin perbandingan yang adil.

Tabel 1. Spesifikasi Lingkungan Penelitian

Komponen	**Spesifikasi**
Prosesor (CPU)	AMD Ryzen 5 5600H
Memori (RAM)	8 GB DDR4 3200MHz
Penyimpanan	512 GB SSD NVMe
Sistem Operasi	Windows 11 64-bit
Isolasi	*Docker* Desktop
Basis Data 1	*PostgreSQL* versi 18.0
Basis Data 2	*MongoDB* versi 8.0
Bahasa Penguji	Go (*Golang*) versi 1.24

Spesifikasi perangkat keras host dan perangkat lunak yang diisolasi menggunakan *Docker container* dijabarkan dalam Tabel 1. Seluruh metrik dicatat secara terprogram dari *container* ini.

Objek Penelitian dan Skenario Pengujian

Objek penelitian berupa dataset sintetis dalam format *JSON* yang dirancang untuk mensimulasikan pola aktivitas pengguna pada *Learning Management System* (*LMS*) perguruan tinggi[10]. Secara spesifik, struktur data *log* yang gunakan paling mendekati tabel standar *mdl_logstore_standard_log* milik *Moodle LMS*. yang meliputi: *user_id, course_id, action_type* (*log*in, *view*, *submit*), *timestamp*, dan metadata dinamis. Pengujian ini menggunakan durasi tetap selama 180 detik per skenario, setiap *worker* (*Goroutine*) menjalankan operasi *INSERT* secara berulang, yaitu mengirim operasi berikutnya setelah operasi sebelumnya selesai atau gagal, hingga durasi pengujian berakhir. Pengujian kinerja dieksekusi melalui empat skenario beban untuk memetakan kurva performa dan mendeteksi titik jenuh (*bottleneck*) dari masing-masing basis data[12].

Tabel 2. Spesifikasi Lingkungan Penelitian

Skenario	***Goroutines***	**Durasi Pengukuran**	**Kategori Beban**
Skenario 1	500	120 – 180 detik	Beban Rendah - Penggunaan Normal
Skenario 2	1.000	120 – 180 detik	Beban Menengah - Jam Sibuk
Skenario 3	5.000	120 – 180 detik	Beban Tinggi - Lonjakan Aktivitas
Skenario 4	10.000	120 – 180 detik	Beban *Ekstrem - Breaking point*

 https://doi.org/10.12345/xxxxx

Skenario pengujian yang tercantum pada Tabel 2 mencakup penambahan beban secara progresif. Setiap skenario mencakup operasi *write* (*INSERT*)[11].

Variabel Penelitian dan Definisi Operasional

Throughput didefinisikan sebagai jumlah total transaksi sukses (*INSERT*) dibagi durasi waktu eksekusi skrip, dinyatakan dalam Transaksi per Detik (TPS). *Error Rate* didefinisikan sebagai persentase transaksi gagal dari total yang dikirimkan. CPU *Utilization* (%) adalah beban prosesor maksimum proses *database* (*postgres/mongod*) saat berjalan. RAM *Utilization* (MB) adalah total alokasi memori fisik yang dikonsumsi oleh mesin basis data selama pengujian berlangsung[11].

Instrumen Penelitian dan Kalibrasi

Instrumen penelitian utama meliputi: (1) Skrip *Load Generator* kustom berbasis *Golang* dengan *concurrent Goroutines*; (2) *System Resource Monitor* otomatis menggunakan *Docker stats* dengan *flag --no-stream* yang dieksekusi periodik; dan (3) Prosedur Kalibrasi Pra-Pengujian yang meliputi *restart container*, *flush cache* internal, dan *warm-up run[13]*.

Teknik Pengumpulan dan Analisis Data

Setiap skenario dijalankan sebanyak 3 kali pengulangan (*run*) untuk menjamin reliabilitas. Nilai representatif yang dianalisis adalah nilai rata-rata (*Mean*). Analisis data menggunakan teknik statistik deskriptif dan komparatif deskriptif[6], [14].

1. Rata-rata (Mean):

Persamaan (1) digunakan untuk menentukan nilai rata-rata (*mean*) dari setiap metrik *Throughput*, *CPU Usage*, dan *RAM Usage*. Pada persamaan (1), $\bar{x}$ merupakan nilai rata-rata hitung (*mean*), $\sum(x_i)$ adalah jumlah total seluruh nilai observasi data sampel, sedangkan n merupakan jumlah total sampel pengujian yang dilakukan.

$$\bar{x} = \frac{\sum(x_i)}{n} \qquad (1)$$

Di mana n = 3.

2. Standar *Deviasi* (SD):

Persamaan (2) digunakan untuk menghitung nilai *Standar Deviasi* (SD). Pada persamaan (2), SD merupakan *Standar Deviasi* yang menunjukkan tingkat sebaran data pengujian, x_i adalah nilai data sampel ke-i, $\bar{x}$ adalah nilai rata-rata dari persamaan (1), dan n merupakan jumlah sampel pengujian.

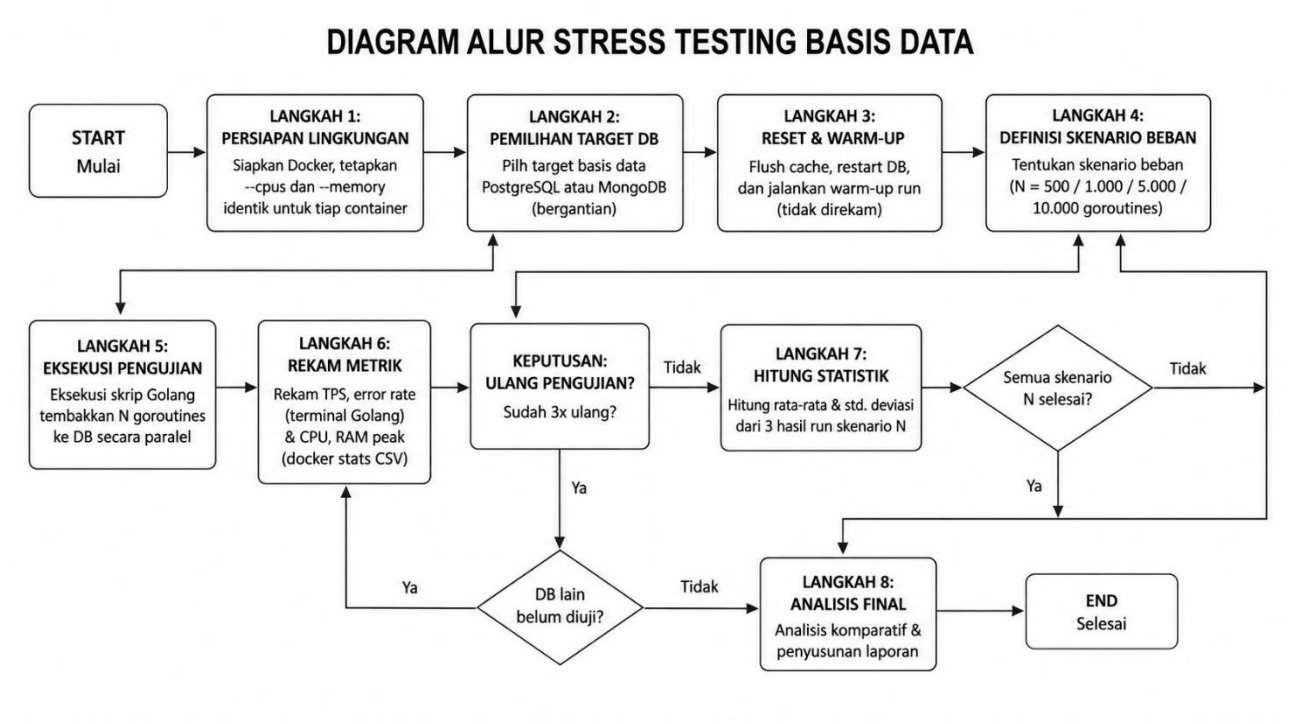
$$SD = \sqrt{\frac{\sum((x_i - \bar{x})^2)}{n-1}} \qquad (2)$$

3. Koefisien Variasi (Coefficient of Variation - CV):

Persamaan (3) digunakan untuk menghitung nilai *Koefisien Variasi* (CV). Pada persamaan (3), CV merupakan *Koefisien Variasi* dalam bentuk persentase, SD adalah nilai *Standar Deviasi* dari persamaan (2), dan $\bar{x}$ adalah nilai rata-rata dari persamaan (1).

$$CV = \left(\frac{SD}{\bar{x}}\right) * 100\% \quad (3)$$

Alur Penelitian



Gambar 1. Diagram alur tahapan penelitian

Penelitian ini menerapkan pendekatan eksperim-ental kuantitatif melalui skenario *Stress testing* untuk mengukur ketahanan *PostgreSQL* dan *MongoDB* terhadap beban *write-intensive*. Variabel bebas yang diuji adalah jenis basis data (*PostgreSQL* vs. *MongoDB*) dan tingkat konkurensi (500, 1.000, 5.000, dan 10.000 *Goroutines*). Variabel terikatnya meliputi *throughput* (TPS), *error Rate*, serta utilisasi CPU dan RAM. Lingkungan pengujian dibangun di atas *Docker* dengan pembatasan sumber daya (*resource cap*) CPU dan memori yang identik untuk menjamin keadilan komparasi.
Instrumen pengujian dikembangkan menggunakan bahasa Go versi 1.24, memanfaatkan *Goroutines* paralel yang dikelola *sync.WaitGroup*, *atomic counter* untuk merekam transaksi sukses/gagal, serta *driver pgx* (*PostgreSQL*) dan mongo-go (*MongoDB*) untuk koneksi. Prosedur penelitian dilaksanakan dalam enam tahapan: (1) studi literatur dan persiapan perangkat keras; (2) konfigurasi *Docker* serta pembuatan skema tabel/koleksi yang setara; (3) pengembangan skrip penguji kustom; (4) eksekusi beban bertingkat terhadap 8 kombinasi (2 DB × 4 skenario) masing-masing diulang 3 kali; (5) konsolidasi metrik *throughput*/*error* dari terminal dan pemanfaatan CPU/RAM dari file *CSV Docker stats* yang diekstrak nilai puncaknya; serta (6) analisis komparatif dan penyusunan laporan berdasarkan nilai rata-rata dan standar *deviasi* dari tiga ulangan. Rangkaian prosedur ini dirancang untuk memastikan replikabilitas serta akurasi pengukuran.

HASIL DAN PEMBAHASAN

Implementasi dan Lingkungan Pengujian

Pengujian performa komparatif antara PostgreS\QL versi 18.0 dan *MongoDB* versi 8.0 dilakukan dengan merancang program *load* generator kustom berbasis *concurrent Golang* versi 1.24 (*Goroutines*). Arsitektur *load* generator ini menyimulasikan ribuan *workers* (virtual users) yang mengirimkan data *log* aktivitas *e-learning* secara serentak (*concurrent*) ke dalam masing-masing *database instance*. Sebagai perlakuan eksperimental yang ketat, pengujian sengaja menonaktifkan fitur *connection pooling* di sisi klien (menihilkan fungsi *SetMaxOpenConns* dan *SetMaxIdleConns* pada driver) guna mengevaluasi batas ketahanan murni (*raw performance*) dan batas toleransi arsitektural bawaan kedua mesin *database*.

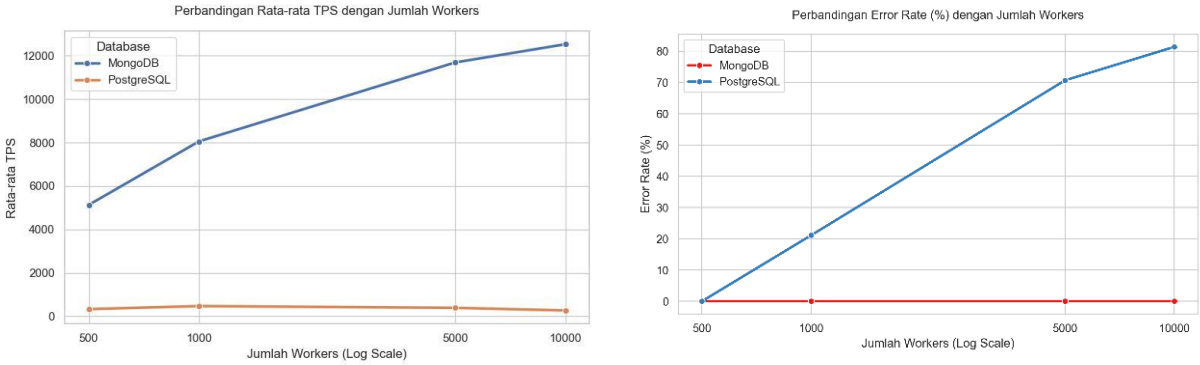
Analisis Hasil Pengujian Throughput dan Success Rate

Metrik *throughput* (TPS) dan *success Rate* merupakan parameter kuantitatif utama untuk mengukur kecepatan pemrosesan data dan tingkat keandalan masing-masing mesin *database* di bawah beban konkuren yang meningkat secara progresif. Hasil tabulasi pengujian dirangkum dalam Tabel 3.

 https://doi.org/10.12345/xxxxx

Tabel 3. Hasil pengujian *throughput* (TPS) dan *error Rate* (%) Terhadap Beban Konkuren

Database	***Workers***	**Rata-rata TPS**	**SD TPS**	**CV TPS (%)**	**Total Sukses**	**Total Gagal**	***Error Rate* (%)**
MongoDB	500	5.127,70	1.738,56	33,91%	1.500	0	0,00%
MongoDB	1.000	8.055,50	1.283,07	15,93%	3.000	0	0,00%
MongoDB	5.000	11.690,76	447,45	3,83%	15.000	0	0,00%
MongoDB	10.000	12.532,26	251,06	2,00%	30.000	0	0,00%
PostgreSQL	500	325,01	153,23	47,15%	1.500	0	0,00%
PostgreSQL	1.000	467,06	71,16	15,24%	2.366	634	21,13%
PostgreSQL	5.000	383,94	43,78	11,40%	4.397	10.603	70,69%
PostgreSQL	10.000	264,30	40,99	15,51%	5.581	24.419	81,40%



Gambar 2. Perbandingan rata-rata TPS dan *error Rate* dengan jumlah *workers*

Berdasarkan Tabel 3 dan Gambar 2, terlihat perbedaan performa *throughput* yang sangat kontras antara kedua *database*. *MongoDB* menunjukkan ketangguhan arsitektural yang sangat superior. Pada skenario beban rendah (500 *workers*), *MongoDB* mencatat *throughput* rata-rata sebesar 5.127,70 TPS. Seiring bertambahnya beban, *throughput MongoDB* melonjak secara progresif: mencapai 8.055,50 TPS pada 1.000 *workers*, meningkat ke 11.690,76 TPS pada 5.000 *workers*, dan akhirnya memuncak pada 12.532,26 TPS pada skenario beban *ekstrem* 10.000 *workers*. Pola pertumbuhan ini mencerminkan skalabilitas vertikal yang sangat linear dan stabil, membuktikan bahwa arsitektur internal *MongoDB* mampu mengeksploitasi seluruh daya komputasi yang tersedia untuk memproses antrean *kueri* masif.

Sebaliknya, *PostgreSQL* mengalami kurva performa yang sangat memprihatinkan dan mengal-ami degradasi yang parah. Pada beban rendah (500 *workers*), *PostgreSQL* hanya mampu memproses *kueri* dengan *throughput* rata-rata sebesar 325,01 TPS. Meskipun sempat mengalami kenaikan tipis pada skenario 1.000 *workers* dengan mencatat *throughput* puncak sebesar 467,06 TPS, performa *PostgreSQL* langsung merosot tajam ketika beban dinaikkan lebih lanjut. Pada skenario 5.000 *workers*, *throughput* menu*run* menjadi 383,94 TPS, dan terus merosot hingga menyentuh titik terendah sebesar 264,30 TPS pada beban *ekstrem* 10.000 *workers*. Degradasi performa yang drastis ini menandakan terjadinya *bottle-neck* sistemik di mana *database* mengalami kelumpuhan dalam mengelola antrean koneksi konkuren.

Perbedaan dalam metrik *success Rate* Gambar 2 semakin memperjelas ketangguhan *MongoDB*. *MongoDB* secara luar biasa berhasil mempertahankan *success Rate* mutlak sebesar 100,00% (0,00% *error Rate*) di seluruh skenario beban, termasuk pada beban puncak 10.000 *workers* di mana seluruh 30.000 transaksi berhasil diselesaikan tanpa satu pun kegagalan. Di sisi lain, *PostgreSQL* mengalami fenomena '*Breaking point*' yang mengkhawatirkan. Pada skenario 500 *workers*, *PostgreSQL* masih mencatat 100% *success Rate*. Namun, pada skenario 1.000 *workers*, *PostgreSQL* mulai tumbang dengan mencatat *success Rate* yang anjlok menjadi 78,87% (*error Rate* 21,13%), di mana terdapat 634 transaksi gagal dari total 3.000 transaksi yang dikirimkan. Kegagalan ini berlanjut secara berantai hingga mencapai puncaknya pada skenario *ekstrem* 10.000 *workers*, di mana *success Rate PostgreSQL* hancur lebur hingga hanya menyisakan 18,60% (*error Rate* 81,40%), dengan akumulasi kegagalan transaksi sebanyak 24.419 transaksi .

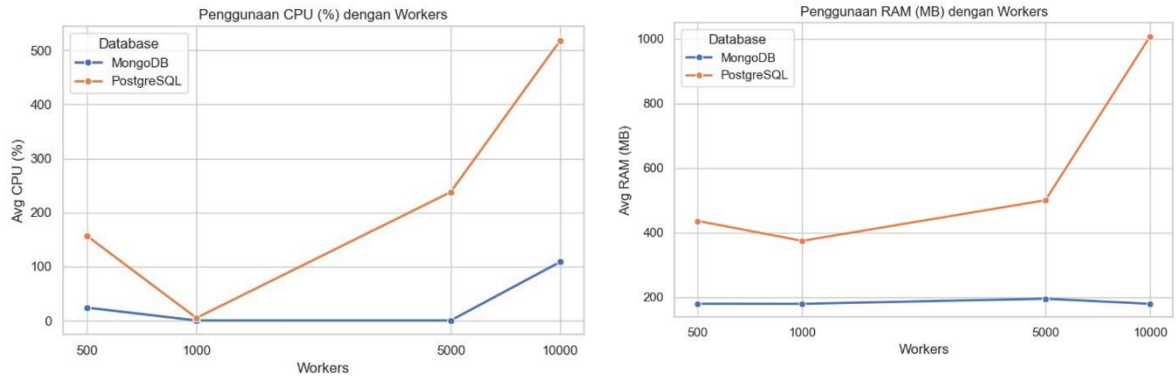
Analisis akar masalah terhadap *run*tuhnya keanda *PostgreSQL* ini pada peniadaan *connection pooling* dan batasan *default* max_*connection*s yang secara standar dikonfigurasi sebesar 100 klien pada *PostgreSQL*. Ketika ribuan *Goroutines* mencoba membuka koneksi TCP mandiri secara simultan ke dalam kontainer *PostgreSQL*, batas max_*connection*s langsung terpenuhi dalam hitungan milidetik. Akib-atnya, *PostgreSQL* menolak secara instan sisa perm-intaan koneksi baru yang masuk, memicu timeout pada aplikasi klien. Ketiadaan *connection pooling* memaksa *PostgreSQL* berulang kali melakukan forking proses sistem operasi baru yang mengonsumsi memori besar hingga mengalami kemacetan total.

Analisis Utilisasi Sumber Daya (Resource Utilization)

Pengukuran konsumsi sumber daya fisik peladen yang meliputi utilitas prosesor (CPU *Utilization*) dan alokasi memori utama (RAM *Utilization*) memberikan pemahaman mendalam mengenai efisiensi komputasi masing-masing *database* . Data hasil pemantauan *realtime* selama eksekusi stress test dirangkum dalam Tabel 4.

Tabel 4. Rangkuman Utilisasi CPU (%) dan RAM (MB) Berdasarkan Skenario Beban

Database	***Workers***	**Rata-rata TPS**	**Avg CPU (%)**	**Avg RAM (MB)**
MongoDB	500	5.127,70	24,29%	180,00
MongoDB	1.000	8.055,50	0,57%	179,87
MongoDB	5.000	11.690,76	0,53%	195,47
MongoDB	10.000	12.532,26	108,78%	179,83
PostgreSQL	500	325,01	156,95%	436,50
PostgreSQL	1.000	467,06	5,20%	374,80
PostgreSQL	5.000	383,94	237,84%	500,50
PostgreSQL	10.000	264,30	518,00%	1.008,19



Gambar 3. Perbandingan penggunaan CPU dan RAM berdasarkan beban workers

Analisis Penggunaan Memori (RAM): Berdasarkan Tabel 4 dan Gambar 3, *MongoDB* menunjukkan efisiensi penggunaan memori yang luar biasa stabil di sepanjang skenario pengujian. Pada 500 *workers*, konsumsi RAM *MongoDB* tercatat sebesar 180,00 MB, kemudian bertahan pada 179,87 MB (1.000 *workers*), 195,47 MB (5.000 *workers*), dan hanya 179,83 MB pada beban puncak 10.000 *workers*. Kestabilan ini disebabkan oleh arsitektur *daemon* mongod yang *multi-threaded*, yang mengelola seluruh koneksi masuk di dalam satu ruang memori tunggal menggunakan *thread pool* yang sangat ringan.

Di sisi lain, *PostgreSQL* menunjukkan RAM yang *ekstrem*. Penggunaan RAM *PostgreSQL* melonjak dari 436,50 MB (500 *workers*), menjadi 500,50 MB (5.000 *workers*), dan membengkak secara masif hingga mencapai 1.008,19 MB pada beban 10.000 *workers*. Pembengkakan memori ini terjadi akibat overhead model *process-per-connection PostgreSQL*, di mana setiap koneksi konkuren yang masuk memaksa sistem operasi melakukan forking proses baru yang masing-masing mengonsumsi memori privat (work_mem dan *stack* proses OS tersendiri).

 https://doi.org/10.12345/xxxxx

Analisis Penggunaan CPU: Pola konsumsi CPU juga menunjukkan efisiensi yang sangat kontras Gambar 3. *MongoDB* mencatat penggunaan CPU yang sangat minim pada skenario menengah, yaitu sebesar 0,57% (1.000 *workers*) dan 0,53% (5.000 *workers*) karena *kueri* penulisan hanya ditulis secara asinkron ke dalam *cache* memori (*WiredTiger* internal *cache*). CPU *MongoDB* baru melonjak menjadi 108,78% pada skenario *ekstrem* 10.000 *workers* sebagai bentuk respons produktif untuk memproses 12.532,26 TPS transaksi sukses. Sebaliknya, *PostgreSQL* mengalami lonjakan CPU yang tidak produktif dan sangat boros. CPU *PostgreSQL* melonjak dari 156,95% (500 *workers*) menjadi rata-rata 518,00% pada skenario puncak 10.000 *workers*. Lonjakan CPU *PostgreSQL* yang masif ini tidak berkorelasi dengan transaksi sukses (*throughput*-nya justru terdegradasi menjadi 264,30 TPS), melainkan habis terbuang secara sia-sia untuk menangani *overhead context switching* pada penjadwal sistem operasi serta pemrosesan kegagalan transaksi (*rollback*) akibat ketiadaan *connection pooling*.

Analisis Stabilitas Performa Berdasarkan Standar Deviasi (SD) dan Koefisien Variasi (CV)

Analisis stabilitas performa menggunakan Standar *Deviasi* (SD) dan Koefisien Variasi (CV) merupakan instrumen penting untuk membedah tingkat konsistensi masing-masing *database* ketika dihadapkan pada tekanan konkuren dari waktu ke waktu. Merujuk pada Tabel 3, terdapat fenomena statistik yang sangat menarik pada *MongoDB* pada skenario beban rendah (500 *workers*). *MongoDB* mencatat nilai SD yang cukup tinggi, yaitu sebesar 1.738,56 TPS, yang menghasilkan nilai CV sebesar 33,91% (jauh melampaui batas *deviasi* ideal 10%). Namun, tingginya nilai CV ini bukan merupakan kelemahan arsitektur, melainkan bukti ilmiah terjadinya proses '*Cache Warming*' (pemanasan *cache*) pada mesin penyimpanan *WiredTiger*.

Pada pengujian awal (*Run* 1) yang bersifat cold start, *MongoDB* mencatat *throughput* rendah sebesar 3.130,32 TPS karena sistem harus mengalokasikan ruang *cache* baru di dalam memori fisik yang di rata-ratakan pada baris pertama. Pada pengulangan berikutnya (*Run* 2 dan *Run* 3) ketika *cache* internal *WiredTiger* telah terisi penuh ('panas'), performa *MongoDB* melonjak drastis masing-masing menjadi 6.300,83 TPS dan 5.951,96 TPS. Bukti kekuatan arsitektur *MongoDB* terlihat jelas pada skenario beban tinggi dan *ekstrem*, di mana nilai CV *MongoDB* menyusut drastis menjadi hanya 3,83% pada 5.000 *workers* (SD 447,45 TPS) dan menyentuh angka istimewa sebesar 2,00% pada beban puncak 10.000 *workers* (SD 251,06 TPS). Hal ini membuktikan bahwa setelah *cache* memori *WiredTiger* aktif secara penuh, *MongoDB* mampu menyajikan kestabilan performa yang sangat tinggi dan dapat diandalkan secara konsisten tanpa fluktuasi yang berarti.

Sebaliknya, *PostgreSQL* mencatat stabilitas yang sangat fluktuatif di awal pengujian dengan nilai CV sebesar 47,15% pada 500 *workers* (SD 153,23 TPS). Meskipun nilai CV *PostgreSQL* tampaknya 'tertahan' secara stagnan di kisaran 11,40% s.d. 15,51% pada skenario beban tinggi (5.000 dan 10.000 *workers*), stabilitas ini merupakan stabilitas semu (pseudo-stability). Kondisi ini terjadi karena *PostgreSQL* telah mengalami saturasi total pada sumber daya sistem operasi akibat proses penolakan transaksi yang seragam secara kontinu. *PostgreSQL* tidak mampu berkembang lagi dan terjebak pada *throughput* rendah yang stagnan karena sistem operasi disibukkan oleh pengelolaan proses baru (*forking*) dan eksekusi transaksi gagal.

Konsistensi dan Efesiensi

Hasil pengujian memberikan jawaban pasti untuk kedua rumusan masalah penelitian ini. Untuk perbandingan *throughput*, analisis kelipatan performa disajikan pada Tabel 5. Untuk perbandingan efisiensi penggunaan CPU dan RAM, komparasi disajikan pada Tabel 6.

Tabel 5. Rasio kelipatan performa *throughput* (TPS) *MongoDB* terhadap *PostgreSQL*

Jumlah *Workers*	***Throughput PostgreSQL* (TPS)**	***Throughput MongoDB* (TPS)**	***MongoDB* Lebih Cepat (x)**
500	325,01	5.127,70	15,78x
1.000	467,06	8.055,50	17,25x
5.000	383,94	11.690,76	30,45x
10.000	264,30	12.532,26	47,42x

Berdasarkan Tabel 5, *MongoDB* secara konsisten mengungguli *PostgreSQL* di seluruh skenario pengujian dengan selisih performa yang melebar secara eksponensial. Pada skenario beban terendah (500 *workers*), *MongoDB* 15,78 kali lipat lebih cepat. Keunggulan ini terus melebar menjadi 17,25 kali lipat pada beban 1.000 *workers*, 30,45 kali lipat pada

5.000 *workers,* dan mencapai titik tertinggi sebesar 47,42 kali lipat (dibulatkan menjadi 47,4x) pada beban *ekstrem* 10.000 *workers* (12.532,26 TPS vs 264,30 TPS). Ini menegaskan keunggulan mutlak *MongoDB* untuk skenario *write-intensive.*

Tabel 6. Komparasi efisiensi penggunaan CPU (%) dan RAM (MB)

Jumlah *Workers*	**CPU *PostgreSQL* (%)**	**RAM *PostgreSQL* (MB)**	**CPU *MongoDB* (%)**	**RAM *MongoDB* (MB)**
500	156,95%	436,50	24,29%	180,00
1.000	5,20%	374,80	0,57%	179,87
5.000	237,84%	500,50	0,53%	195,47
10.000	518,00%	1.008,19	108,78%	179,83

Berdasarkan Tabel 6, *MongoDB* menunjukkan efisiensi alokasi RAM yang sangat tinggi dan stabil, bertahan konsisten pada rata-rata ~180 MB di seluruh skenario. Sebaliknya, *PostgreSQL* bertindak sangat boros memori, melonjak dari 436,50 MB hingga mencapai 1.008,19 MB pada beban 10.000 *workers* akibat overhead OS-level *process-per-connection.* Dari segi CPU, *MongoDB* mencatat penggunaan CPU di bawah 1% pada beban menengah, dan meningkat produktif menjadi 108,78% pada beban 10.000 *workers* untuk memproses 12.532,26 TPS. Sebaliknya, CPU *PostgreSQL* tersaturasi secara boros hingga mencapai 518,00% pada skenario puncak 10.000 *workers*, bukan untuk penulisan sukses melainkan terbuang sia-sia untuk *context* switching dan *rollback* transaksi gagal.

KESIMPULAN DAN SARAN

Hasil penelitian eksperimental komparatif ini membuktikan secara mutlak bahwa basis data berorientasi dokumen (*MongoDB* versi 8.0) memiliki keunggulan performa, keandalan, dan efisiensi sumber daya yang jauh lebih baik dibandingkan dengan basis data relasional (*PostgreSQL* versi 18.0) dalam menangani data *log* aktivitas *e-learning* yang bersifat *write-intensive* dan konkuren tinggi. Pada skenario beban puncak *ekstrem* (10.000 *workers*), arsitektur *multi-threaded daemon MongoDB* mampu memproses transaksi penulisan sebesar 12.532,26 TPS (47,42 kali lipat lebih cepat dibandingkan *PostgreSQL* yang terpuruk pada 264,30 TPS) dengan *success Rate* sempurna 100,00%, sementara *PostgreSQL* mengalami kelumpuhan sistemik (*breaking point*) mulai dari skenario 1.000 *workers* (*error Rate* 21,13%) hingga hancur pada skenario puncak dengan *error Rate* 81,40% akibat *connection exhaustion.* Selain itu, *MongoDB* sangat efisien dalam alokasi memori dengan konsumsi RAM yang stabil pada rata-rata ~180 MB, berbeda jauh dari model *process-per-connection PostgreSQL* yang boros memori hingga mencapai 1.008,19 MB dan memicu saturasi CPU non-produktif sebesar 518,00% untuk penanganan *context switching OS* dan *rollback* kegagalan transaksi. Sebagai implikasi taktis, tim IT universitas sangat disarankan untuk menerapkan arsitektur segregasi *database* pada sistem *e-learning* mereka, memisahkan data *log* yang *write-intensive* ke dalam *MongoDB*, guna mengamankan ketersediaan dan keandalan *server* akademik transaksional utama. Untuk penelitian di masa mendatang, disarankan untuk melakukan pengujian dengan *CRUD* (*Create, Read, Update, Delete*) menggunakan data *log* produksi aktual di perguruan tinggi, serta menguji performa basis data pada lingkungan infrastruktur cloud terdistribusi untuk melihat pengaruh nyata dari latensi jaringan.

DAFTAR PUSTAKA

[1] R. Rizalul Akhsan, D. Lamilga Sudiono Putra, D. Indra Sensuse, D. Sumirat Hidayat, and E. Hakiki Purwaningsih, “Development of a Conceptual Framework for Implementing Learning Management System (LMS) in Higher Education: An Empirical Validation Study,” 2025.

[2] N. D. Anwar, N. Y. Setiawan, and W. Purnomo, “Analisis Interaksi Aktivitas Pembelajaran Daring Berdasarkan Data Log Aktivitas pada Learning Managemefnt System (LMS) Menggunakan Educational Process Mining,” *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 11, no. 5, pp. 1025–1032, Oct. 2024, doi: 10.25126/jtiik.2024117987.

[3] H. He *et al.*, “When Database Meets New Storage Devices: Understanding and Exposing Performance Mismatches via Configurations,” in *Proceedings of the VLDB Endowment*, VLDB Endowment, 2023, pp. 1712–1725. doi: 10.14778/3587136.3587145.

[4] S. R. S. Al Maamari and M. Nasar, "A Comparative Analysis of NoSQL and SQL Databases: Performance, Consistency, and Suitability for Modern Applications with a Focus on IoT," *East Journal of Computer Science*, vol. 1, no. 2, pp. 10–15, Apr. 2025, doi: 10.63496/ejcs.vol1.iss2.76.

[5] D. Rosmala and I. Rasyidin, "Data Storage Database PostgreSQL and JSON File Format in Golang Using Gin-Gonic and Gin-Gonic with GORM," *Industrial Sciencetech Jurnal*, vol. 1, no. 1, doi: 10.26760/jrh.

[6] J. Olszak and M. Skublewska-Paszkowska, "The Impact of Relational and Non-Relational Databases on Application Performance Wpływ relacyjnych i nierelacyjnych baz danych na wydajność aplikacji," 2025.

[7] S. Ferreira, J. Mendonça, B. Nogueira, W. Tiengo, and E. Andrade, "Impacts of data consistency levels in cloud-based NoSQL for data-intensive applications," *Journal of Cloud Computing*, vol. 13, no. 1, Dec. 2024, doi: 10.1186/s13677-024-00716-7.

[8] W. Khan, T. Kumar, Z. Cheng, K. Raj, A. M. Roy, and B. Luo, "SQL and NoSQL Databases Software architectures performance analysis and assessments-A Systematic Literature review."

[9] H. Ardiansyah and A. Fatwanto, "Comparison of Memory usage between REST API in Javascript and Golang," *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, vol. 22, no. 1, pp. 229–240, Nov. 2022, doi: 10.30812/matrik.v22i1.1325.

[10] S. Agal, "A privacy preserving synthetic learner dataset for learning analytics in technology enhanced higher education," *Sci. Rep.*, vol. 16, no. 1, Dec. 2026, doi: 10.1038/s41598-026-44990-8.

[11] N. Pantelic *et al.*, "Benchmarking SQL and NoSQL Persistence in Microservices Under Variable Workloads," *Future Internet*, vol. 18, no. 1, p. 53, Jan. 2026, doi: 10.3390/fi18010053.

[12] MoodleDocs, "Logging," https://docs.moodle.org/405/en/Logging.

[13] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, Monitoring, and Testing of Microservices Systems: The Practitioners' Perspective," Aug. 2021, doi: 10.1016/j.jss.2021.111061.

[14] J. Redžepagić, A. Kapulica, N. Malešević, and V. Dakić, "Empirical Performance and Operational Analysis of Monolithic and Distributed Database Architectures in Kubernetes Environments," *Computers*, vol. 15, no. 5, May 2026, doi: 10.3390/computers15050282.