BAB II

LANDASAN TEORI

2.1 Penelitian Terdahulu

Penelitian terdahulu menunjukkan bahwa perbandingan DBMS merupakan topik yang kompleks karena hasil performa dipengaruhi oleh workload, struktur data, indeks, konfigurasi engine, ukuran dataset, dan lingkungan eksperimen. Oleh karena itu, penelitian tentang DBMS tidak seharusnya menyimpulkan keunggulan universal tanpa menjelaskan konteks pengujian [1], [5], [6].

Kajian SQL dan NoSQL menekankan bahwa model relasional dan model NoSQL memiliki keunggulan yang berbeda. SQL cenderung kuat pada relasi terstruktur, transaksi, constraint, dan query deklaratif, sedangkan NoSQL menawarkan fleksibilitas skema dan variasi model penyimpanan seperti document, key-value, wide-column, dan graph. Perbedaan ini menjadi penting pada RIS karena data publikasi memiliki relasi eksplisit, tetapi juga memuat metadata dan kata kunci yang dapat berubah [2], [3], [4].

Beberapa studi menunjukkan bahwa DBMS dokumen dapat unggul pada pola akses tertentu, terutama ketika data yang dibaca dapat disimpan dalam satu dokumen atau ketika kebutuhan join dapat dikurangi. Namun, performa tersebut sangat bergantung pada desain dokumen dan indeks. Sebaliknya, DBMS relasional dapat tetap kompetitif pada query terstruktur dan relasi yang jelas ketika indeks, constraint, dan query plan dirancang dengan baik [10], [11], [12].

Tabel 2.1 merangkum penelitian terdahulu dan referensi teknis yang digunakan untuk memosisikan penelitian ini. Tabel ini menunjukkan bahwa sebagian kajian membahas perbandingan SQL dan NoSQL, sebagian membahas benchmark DBMS, dan sebagian lain mendukung konteks sistem informasi penelitian serta synthetic data.

Tabel 2.1 Penelitian terdahulu dan relevansi terhadap penelitian

Ref.	Sumber	Fokus	Relevansi
[1]	Taipalus (2024)	Systematic literature review benchmark DBMS	Menegaskan pentingnya workload, dataset, konfigurasi, dan batas generalisasi benchmark.
[2]	Khan et al. (2023)	Systematic literature review SQL dan NoSQL	Menjadi dasar perbedaan arsitektur dan karakteristik SQL atau NoSQL.
[10]	Carvalho et al. (2023)	Benchmark NoSQL document database	Memberi konteks performa MongoDB dan document database.
[11]	Salunke dan Ouda (2024)	Benchmark PostgreSQL dan MySQL	Mendukung pembahasan kandidat SQL open-source dan workload relasional.
[12]	Makris et al. (2021)	Perbandingan MongoDB dan PostgreSQL	Memperkuat konteks perbandingan relational DBMS dan document database.
[13]	Hayek dan Akkad (2024)	SQL Server dan MongoDB	Mendukung pembahasan SQL Server sebagai DBMS relasional pembanding.
[14]	Lupu et al. (2024)	Relational dan NoSQL pada concurrent access	Menjelaskan bahwa performa DBMS dipengaruhi karakter akses dan algoritma.
[15]	Ellul et al. (2024)	PostgreSQL dan MongoDB pada data kompleks	Mendukung pembahasan model data dan kesesuaian database.
[16]	Choi et al. (2024)	Visual analytics benchmark DBMS	Mendukung interpretasi benchmark multi-metrik.
[17]	Henning dan Hasselbring (2022)	Benchmark scalability cloud-native	Mendukung prinsip konfigurasi benchmark dan replikasi pengujian.
[5]	Papadopoulos et al. (2021)	Reproducible performance evaluation	Menjadi dasar metodologis untuk lingkungan, workload, dan pelaporan.
[6]	Hasselbring (2021)	Benchmarking sebagai standar empiris	Menjadi dasar validitas dan fairness benchmark.

Berdasarkan tabel tersebut, penelitian ini berada pada irisan antara *benchmark DBMS*, sistem informasi web, dan kebutuhan institusional RIS. Gap

yang ditekankan adalah kebutuhan *API-layer benchmark* yang dikaitkan langsung dengan konteks *Website* RIS UMN, bukan *benchmark query* langsung yang berdiri sendiri.

2.2 Teori yang Berkaitan

2.2.1 *Database Management System* (DBMS)

Database Management System (DBMS) adalah perangkat lunak yang mengelola penyimpanan, pengambilan, pembaruan, keamanan, dan konsistensi data. Dalam sistem informasi web, DBMS menjadi Lapisan persisten yang memungkinkan data dikelola secara berkelanjutan, tidak hanya selama aplikasi berjalan. DBMS juga menyediakan mekanisme *query*, indeks, transaksi, konkurensi, dan pemulihan data sesuai karakteristik produk yang digunakan.

Pada konteks RIS, DBMS perlu menangani data publikasi, peneliti, fakultas, program studi, SDGs, berita, agenda, dan kolaborasi. Setiap kelompok data memiliki tingkat struktur yang berbeda. Karena itu, pemilihan DBMS perlu mempertimbangkan model data, kebutuhan relasi, pola pencarian, pola filter, kemampuan pagination, dan integrasi dengan framework aplikasi [1], [2], [5].

DBMS bukan satu-satunya faktor yang menentukan performa aplikasi. *API* layer, *framework*, driver, connection pool, serialisasi JSON, network stack lokal, dan resource mesin juga berpengaruh. Penelitian ini secara eksplisit menempatkan DBMS sebagai bagian dari arsitektur *API-layer* sehingga hasilnya menggambarkan performa sistem *benchmark*, bukan performa engine *database* murni.

2.2.2 SQL dan NoSQL *Database*

Relational *Database Management System* (RDBMS) menyimpan data dalam tabel yang terdiri atas baris dan kolom. Hubungan antarentitas dikelola melalui primary key, foreign key, constraint, dan tabel relasi. Model ini sesuai untuk data yang memiliki struktur jelas, seperti hubungan peneliti dengan publikasi, publikasi dengan SDGs, serta fakultas dengan program studi.

Keunggulan RDBMS terletak pada konsistensi, integritas referensial, kemampuan join, transaksi, dan bahasa SQL yang sudah matang. PostgreSQL, misalnya, menyediakan constraint dan indeks yang dapat membantu menjaga

integritas data serta mempercepat pencarian. Dalam penelitian ini, Microsoft SQL Server digunakan sebagai salah satu DBMS relasional pembanding untuk melihat performa pada workload API-layer yang sama.

Keterbatasan RDBMS muncul ketika data sering berubah strukturnya atau ketika aplikasi membutuhkan penyimpanan dokumen dengan variasi atribut yang tinggi. Dalam kasus tersebut, skema relasional perlu disesuaikan melalui tabel tambahan, kolom JSON, atau desain hybrid. Pada RIS aktual yang berbasis SQL dengan struktur data relasional yang kompleks, kekuatan RDBMS tetap penting karena kompatibilitas dengan struktur yang sudah ada menjadi faktor implementasi utama.

NoSQL adalah kelompok database yang tidak terbatas pada model tabel relasional. Salah satu model yang relevan untuk penelitian ini adalah document database, yaitu penyimpanan data dalam bentuk dokumen dengan struktur field yang lebih fleksibel. MongoDB dan Firestore menggunakan pendekatan document database, meskipun detail query, indeks, transaksi, dan layanan yang disediakan berbeda. MongoDB dibahas berdasarkan dokumentasi data modeling, sedangkan karakteristik Firestore Emulator dibatasi pada lingkungan pengujian lokal yang digunakan dalam penelitian ini [19], [27].

Database dokumen dapat mengurangi kebutuhan join ketika data yang sering dibaca bersama dapat disimpan dalam satu dokumen. Pada RIS-like dataset, informasi publikasi dapat memuat embedded researchers, SDGs, keyword tokens, atau metadata lain. Pendekatan ini dapat mempercepat pembacaan tertentu, tetapi berpotensi menimbulkan duplikasi data dan kebutuhan sinkronisasi ketika data referensi berubah. NoSQL tidak selalu lebih cepat daripada SQL, dan SQL tidak selalu lebih tepat daripada NoSQL. Kesesuaian bergantung pada pola akses. Jika aplikasi memerlukan transaksi relasional kompleks, RDBMS dapat lebih tepat. Jika aplikasi banyak membaca dokumen yang sudah disusun sesuai kebutuhan tampilan, document database dapat memberi keuntungan. Penelitian ini menilai kedua pendekatan melalui skenario API yang sama [10], [14], [20].

Sistem informasi web biasanya memisahkan antarmuka pengguna, Lapisan aplikasi, dan penyimpanan data. Dalam arsitektur tersebut, *database* tidak langsung diakses oleh pengguna akhir. Pengguna berinteraksi dengan halaman web atau *endpoint API*, lalu *backend* menerjemahkan request menjadi operasi *database*. Oleh karena itu, performa yang dirasakan pengguna merupakan kombinasi performa aplikasi dan *database*.

Pada sistem berbasis SQL, normalisasi data membantu mengurangi duplikasi dan menjaga integritas. *Query* yang melibatkan join dapat merepresentasikan hubungan antarentitas secara eksplisit. Pada sistem berbasis NoSQL, data sering disusun mengikuti pola akses, misalnya menyimpan daftar SDGs atau peneliti langsung dalam dokumen publikasi. Perbedaan ini memengaruhi cara *backend* merancang *adapter* dan *query*.

RIS memiliki kebutuhan yang berada di antara dua pendekatan tersebut. Di satu sisi, struktur fakultas, program studi, peneliti, publikasi, dan SDGs bersifat relasional. Di sisi lain, metadata publikasi, kata kunci, konten berita, dan data kolaborasi dapat lebih fleksibel. Hal ini menjadi alasan penelitian membandingkan SQL Server, PostgreSQL, MariaDB, MongoDB, dan Firestore Emulator secara berdampingan.

Tabel 2.2 menyajikan perbandingan konseptual antara SQL dan NoSQL dalam konteks sistem informasi web. Tabel ini digunakan untuk memperjelas bahwa pilihan *database* harus dikaitkan dengan model data, pola akses, dan kebutuhan operasional aplikasi.

Tabel 2.2 Perbandingan SQL dan NoSQL

Aspek	SQL/RDBMS	NoSQL Dokumen	Implikasi untuk RIS
Model data	Tabel, baris, kolom, relasi	Dokumen, collection, field fleksibel	RIS memiliki relasi kuat dan metadata fleksibel
Skema	Lebih eksplisit dan terstruktur	Lebih fleksibel antar dokumen	SQL membantu konsistensi, dokumen membantu variasi metadata
Relasi	Join dan foreign key	Embedding atau references	Publikasi-peneliti-SDGs dapat dimodelkan dengan dua cara

Aspek	SQL/RDBMS	NoSQL Dokumen	Implikasi untuk RIS
Konsistensi	Umumnya kuat melalui transaksi dan constraint	Bergantung produk dan desain dokumen	Data akademik membutuhkan konsistensi referensi
Query	SQL deklaratif dan mature	*Query API*/agregasi sesuai produk	Search, filter, dan related data perlu indeks berbeda
Kesesuaian praktis	Lebih dekat dengan Laravel RIS aktual	Membutuhkan perubahan model dan *adapter*	PostgreSQL lebih feasible untuk migrasi SQL

Tabel 2.2 menunjukkan bahwa SQL dan NoSQL tidak berada dalam hubungan substitusi sederhana. SQL lebih kuat dari sisi kompatibilitas relasional RIS aktual, sedangkan NoSQL dokumen dapat memberi keuntungan pada pola baca tertentu apabila struktur dokumen disusun mengikuti kebutuhan *endpoint*.

Dalam penelitian ini, perbandingan SQL dan NoSQL tidak digunakan untuk memilih kategori secara mutlak. Analisis dilakukan pada level DBMS dan skenario karena MongoDB dan Firestore Emulator sama-sama termasuk dokumen, tetapi hasil performanya sangat berbeda. Hal ini menunjukkan bahwa desain *adapter*, indeks, dan implementasi produk lebih penting daripada label SQL atau NoSQL semata.

2.2.3 *Database Benchmark*ing dan *Performance testing*

Performance testing adalah proses mengevaluasi perilaku sistem ketika menjalankan beban tertentu. Pengujian ini tidak hanya melihat apakah fungsi berjalan, tetapi juga bagaimana sistem merespons dari sisi waktu, kapasitas, stabilitas, dan kegagalan. Pada aplikasi web, performance testing membantu mengidentifikasi apakah endpoint dapat melayani request secara konsisten [17].

Metrik *performance testing* harus disesuaikan dengan tujuan sistem. Untuk RIS, pengalaman pengguna banyak dipengaruhi oleh waktu membuka daftar publikasi, detail publikasi, filter, search, dan pagination. Karena itu, metrik *response time*, *P95 latency*, *throughput*, *error rate*, dan *checks rate* dipilih sebagai indikator utama.

Load testing adalah bentuk *performance testing* yang mengevaluasi sistem ketika menerima beban pengguna atau request dalam jumlah tertentu. Dalam penelitian ini, *load testing* dilakukan dengan 10 *virtual users* selama 30 detik untuk setiap skenario dan diulang sebanyak 3 iterasi. Konfigurasi ini memberi dasar perbandingan awal antar-DBMS pada beban yang sama.

Load testing tidak sama dengan stress testing. Stress testing biasanya meningkatkan beban sampai sistem mendekati atau melewati batas kapasitas. Penelitian ini tidak bertujuan menemukan titik kegagalan maksimum, tetapi membandingkan performa relatif pada *workload* terkendali yang sama. Oleh karena itu, rekomendasi penelitian lanjutan mencakup VUs lebih tinggi dan durasi lebih panjang.

Metrik evaluasi diperlukan agar hasil *benchmark* dapat dianalisis secara konsisten. Tabel 2.3 menjelaskan metrik yang digunakan dalam penelitian ini beserta interpretasinya.

Tabel 2.3 Metrik evaluasi benchmark

Metrik	**Sumber**	**Makna**	**Interpretasi**
Average response time	http_req_duration avg	Rata-rata durasi request HTTP	Semakin rendah semakin cepat
P95 latency	http_req_duration p(95)	Batas waktu respons 95 persen request	Semakin rendah semakin stabil
Throughput	http_reqs rate	Jumlah request per detik	Semakin tinggi semakin besar kapasitas
Error rate	http_req_failed rate	Proporsi request gagal	Semakin rendah semakin baik
Checks rate	*checks rate*	Proporsi validasi respons yang berhasil	Semakin tinggi semakin valid
db_time	custom Trend	Waktu *adapter* yang direncanakan	Tidak dianalisis karena kolom CSV kosong

Metrik HTTP K6 menjadi dasar utama karena tersedia lengkap pada CSV. Nilai db_time_avg_ms dan db_time_p95_ms tidak digunakan dalam analisis akhir karena tidak terisi pada data yang dianalisis.

Database benchmarking adalah pengukuran performa database menggunakan workload, dataset, konfigurasi, dan metrik tertentu. Workload menentukan operasi yang diuji, misalnya insert, read, update, delete, filter, search, atau pagination. Dataset menentukan ukuran dan karakter data, sedangkan konfigurasi menentukan environment dan parameter pengujian [22], [23].

Workload RIS-like pada penelitian ini dirancang melalui 10 skenario *endpoint*. Skenario tersebut tidak meniru semua fitur RIS aktual, tetapi merepresentasikan operasi inti yang relevan untuk sistem informasi penelitian. Hal ini membuat hasil penelitian dapat dibaca sebagai indikasi performa pada fungsi umum, bukan sebagai simulasi lengkap seluruh transaksi produksi.

Synthetic dataset digunakan ketika data produksi tidak tersedia, bersifat rahasia, atau perlu dikontrol agar pengujian dapat direplikasi. Kajian synthetic data menekankan pentingnya memahami kesesuaian data sintetis terhadap tujuan evaluasi. Pada penelitian ini, dataset sintetis digunakan untuk menjaga keamanan dan membuat reset serta seed dapat dilakukan berulang [22], [24], [25].

Dataset sintetis RIS-like mencakup fakultas, program studi, peneliti, SDGs, publikasi, relasi penulis publikasi, relasi publikasi-SDGs, berita, agenda, dan kolaborasi. Struktur tersebut dirancang agar skenario read, filter, search, related data, dan pagination memiliki konteks yang relevan, meskipun tidak menggunakan data produksi RIS.

Tabel 2.4 merangkum research gap yang menjadi dasar kontribusi penelitian. Gap utama berada pada kebutuhan *benchmark DBMS* berbasis API untuk aplikasi RIS-like yang memiliki data relasional dan semi-terstruktur, serta membutuhkan rekomendasi yang dapat ditafsirkan dalam konteks pengembangan *Website* RIS UMN.

Tabel 2.4 Research Gap Penelitian

Area	**Penelitian sebelumnya**	**Kebutuhan yang belum tertutup**	**Kontribusi penelitian**
Benchmark DBMS	Banyak membahas *query* dan *workload* umum	*Workload* RIS-like melalui *API* masih terbatas	Menyusun 10 skenario *endpoint* RIS-like

Area	Penelitian sebelumnya	Kebutuhan yang belum tertutup	Kontribusi penelitian
SQL vs NoSQL	Membahas perbedaan model data	Perlu diuji pada operasi sistem informasi web	Membandingkan SQL Server, PostgreSQL, MariaDB, MongoDB, Firestore Emulator
Load testing tool	K6/JMeter/Locust umum dibahas sebagai alat	Perlu pemilihan tool sesuai *API benchmark*	Menggunakan K6 sebagai load generator berbasis kode
Interpretasi praktis	Sering fokus pada ranking performa	Perlu membedakan performa dan kelayakan migrasi	Membedakan MongoDB terbaik *benchmark* dan PostgreSQL kandidat SQL praktis

Dengan demikian, penelitian ini tidak hanya menghasilkan ranking *DBMS*, tetapi juga memberikan interpretasi aplikatif terhadap kebutuhan RIS. Penelitian ini tidak bertujuan menciptakan teori baru database *benchmark*ing, melainkan menyusun rekomendasi berbasis pengujian untuk mendukung keputusan pemilihan *DBMS* pada pengembangan RIS UMN.

Kerangka pemikiran penelitian dimulai dari kebutuhan RIS sebagai sistem informasi institusional yang mengelola data penelitian, publikasi, SDGs, berita, agenda, dan kolaborasi. Kebutuhan tersebut menimbulkan persoalan pemilihan DBMS karena data memiliki karakter relasional dan semi-terstruktur.

Masalah tersebut dijawab melalui API-layer benchmark menggunakan Grafana K6. K6 mengirim request ke Backend API, Backend API meneruskan operasi ke adapter, dan adapter menjalankan operasi pada DBMS target. Hasil diukur menggunakan average response time, P95 latency, throughput, error rate, dan checks rate. Data hasil kemudian diolah menjadi CSV, processed CSV, tabel, grafik, dan analisis per skenario [26].

Kerangka akhir penelitian membedakan dua Lapisan keputusan. Lapisan pertama adalah hasil performa *benchmark*, yaitu MongoDB sebagai DBMS dengan performa agregat terbaik pada *workload* yang diuji. Lapisan kedua adalah kelayakan implementasi praktis, yaitu PostgreSQL sebagai kandidat SQL paling kuat untuk konteks RIS aktual yang berbasis Laravel dan SQL.

2.2.4 Pemeringkatan Indikator Performa

Dalam database benchmarking, setiap indikator performa dapat memiliki satuan dan arah interpretasi yang berbeda. Average response time dan P95 latency diukur dalam milidetik dengan prinsip bahwa nilai yang lebih rendah menunjukkan performa yang lebih baik. Sementara itu, throughput diukur dalam request per second dengan prinsip bahwa nilai yang lebih tinggi menunjukkan kemampuan pemrosesan request yang lebih baik.

Untuk membandingkan posisi setiap DBMS secara terstruktur, hasil pada setiap indikator dapat diubah menjadi peringkat. DBMS dengan average response time dan P95 latency terendah memperoleh peringkat lebih baik, sedangkan DBMS dengan throughput tertinggi memperoleh peringkat lebih baik. Peringkat dari ketiga indikator tersebut kemudian dirata-ratakan untuk memperoleh mean rank.

Error rate dan checks rate tetap dianalisis sebagai indikator kestabilan endpoint. Namun, apabila seluruh DBMS memiliki error rate dan checks rate yang sama, kedua indikator tersebut tidak digunakan sebagai pembeda dalam penentuan ranking. Nilai mean rank yang lebih rendah menunjukkan performa yang lebih baik pada indikator pembeda yang diuji.

2.3 *Framework* atau Metode yang Digunakan

2.3.1 *API-layer benchmark*

API benchmarking mengukur performa endpoint *API* sebagai permukaan akses aplikasi. Pada pendekatan ini, request dikirim ke endpoint HTTP, kemudian backend menjalankan operasi internal. Metrik yang dihasilkan merepresentasikan latensi *API end-to-end*, bukan hanya waktu *query database*. Pendekatan ini sesuai untuk sistem *web* yang memisahkan pengguna dari *database* melalui *backend* [7], [8], [26].

Keunggulan *API benchmarking* adalah konsistensi antarmuka. K6 dapat mengirim request yang sama ke /benchm*ark*/list, /ben*chmark*/detail/:id, /ben*chmark*/filter/year, atau *endpoint* lain, sementara *backend* memilih DBMS

target melalui parameter db. Dengan demikian, perbedaan koneksi native SQL, MongoDB, dan Firestore tidak langsung dibebankan kepada K6.

Keterbatasannya adalah interpretasi performa menjadi lebih luas. Jika response time tinggi, penyebabnya dapat berasal dari database, driver, adapter, serialization, atau proses Node.js. Oleh karena itu, penelitian ini menyatakan secara jelas bahwa hasilnya adalah API-layer DBMS benchmark, bukan pure database engine benchmark [5], [7].

Tabel 2.5 membedakan *API-layer benchmark* dan *direct database benchmark*. Perbedaan ini penting karena penelitian ini mengambil pendekatan *API-layer*.

Tabel 2.5 API-layer benchmark dan direct database benchmark

Aspek	***API-layer benchmark***	***Direct database benchmark***
Permukaan uji	*Endpoint* HTTP/*API*	Koneksi langsung ke DBMS
Komponen yang ikut terukur	*Backend*, routing, driver, *adapter*, *database*	*Query* engine, driver, dan koneksi *database*
Kedekatan dengan aplikasi web	Lebih dekat dengan pengalaman pengguna *API*	Lebih dekat dengan kemampuan mesin *database*
Kelebihan	*Endpoint* seragam untuk SQL dan NoSQL	Isolasi performa *database* lebih kuat
Keterbatasan	Overhead *API* ikut terukur	Sulit menyamakan protokol antar-DBMS
Kesesuaian penelitian	Dipilih karena RIS diakses melalui aplikasi web	Tidak digunakan sebagai metode utama

Berdasarkan tabel tersebut, hasil penelitian harus dibaca sebagai performa *endpoint API* yang menggunakan DBMS tertentu. Kesimpulan tentang engine *database* murni memerlukan *direct database benchmark* tambahan.

2.3.2 *Backend API* dan *Database adapter layer*

Backend API berperan sebagai Lapisan yang menerima request, melakukan routing, memilih adapter, menjalankan operasi database, dan mengembalikan respons JSON. Database adapter layer berfungsi menyamakan operasi konseptual antar-DBMS walaupun query atau metode implementasinya berbeda. Contohnya,

filter SDGs pada SQL menggunakan join, sedangkan MongoDB menggunakan field embedded dan Firestore Emulator menggunakan query array [7], [8]

Perbedaan antara *backend benchmark* dan *backend* RIS aktual perlu dinyatakan secara eksplisit. Hasil *benchmark* ini tidak langsung membuktikan performa Laravel produksi. Namun, *API-layer benchmark* tetap relevan karena pengguna web pada akhirnya mengalami latensi end-to-end melalui Lapisan aplikasi, bukan melalui koneksi *database* langsung.

2.3.3 Laravel sebagai Konteks Implementasi *Website* RIS Aktual

Laravel adalah *framework* web PHP yang umum digunakan untuk membangun aplikasi berbasis MVC dan *database* SQL. Konteks implementasi RIS aktual yang berbasis Laravel dan SQL dengan struktur data relasional yang kompleks membuat kompatibilitas relasional menjadi faktor penting. Meskipun *benchmark* penelitian ini menggunakan Node.js/TypeScript *Backend API*, interpretasi akhirnya tetap mempertimbangkan konteks Laravel.

2.4 Tools atau Software yang Digunakan

2.4.1 *Database Management System* yang Diuji

Microsoft SQL Server adalah DBMS relasional yang banyak digunakan pada lingkungan enterprise dan ekosistem Microsoft. Dalam penelitian pembanding DBMS, SQL Server dapat diposisikan sebagai representasi DBMS relasional yang dibandingkan dengan MongoDB atau DBMS lain berdasarkan waktu query dan concurrent access [13].

Dalam konteks *benchmark* ini, SQL Server merepresentasikan pilihan DBMS enterprise yang relevan ketika organisasi memiliki integrasi dengan produk Microsoft, kebutuhan administrasi terpusat, dan fitur relasional lanjutan. Namun, hasil *benchmark* lokal tetap dipengaruhi oleh konfigurasi container, driver, resource mesin, connection pool, dan implementasi *query* pada *adapter*.

SQL Server tidak boleh dinilai hanya dari peringkat *benchmark* pada satu konfigurasi. Meskipun hasil penelitian menunjukkan SQL Server berada di bawah MongoDB, PostgreSQL, dan MariaDB pada *workload* ini, SQL Server tetap relevan untuk organisasi yang membutuhkan integrasi enterprise, tooling matang,

dan fitur operasional tertentu. Interpretasi hasil harus dibatasi pada *workload API-layer* yang diuji.

PostgreSQL adalah DBMS relasional open-source yang menekankan extensibility, kepatuhan SQL, constraint, indeks, transaksi, dan mekanisme MVCC. PostgreSQL juga memiliki planner dan optimizer yang menentukan strategi eksekusi query berdasarkan statistik dan rencana query [11], [12], [18]

Dalam penelitian ini, PostgreSQL penting karena menjadi kandidat SQL praktis yang kuat untuk konteks RIS aktual. Jika sistem yang ada berbasis Laravel dan SQL dengan struktur tabel besar, PostgreSQL menawarkan keseimbangan antara performa, kemampuan *query* relasional, dukungan indeks, dan fleksibilitas fitur seperti tipe data JSON untuk kebutuhan semi-terstruktur. Hasil *benchmark* menunjukkan PostgreSQL berada pada peringkat kedua secara keseluruhan dan unggul pada beberapa skenario read, filter, dan pagination. Temuan tersebut perlu dibaca sebagai hasil *API-layer benchmark*, tetapi tetap memperkuat posisi PostgreSQL sebagai kandidat SQL yang layak dipertimbangkan ketika migrasi ke *database* dokumen tidak praktis.

MariaDB adalah DBMS relasional open-source yang berkembang dari ekosistem MySQL dan banyak digunakan pada aplikasi web. Kajian benchmark PostgreSQL dan MySQL serta benchmark variasi nilai data menunjukkan bahwa DBMS relasional perlu dinilai melalui workload, indeks, pola query, dan karakter data yang diuji [11], [22].

MariaDB relevan bagi aplikasi web karena relatif ringan, familier bagi banyak pengembang, dan kompatibel dengan banyak pola pengembangan berbasis SQL. Pada konteks RIS, MariaDB dapat menjadi pilihan apabila organisasi membutuhkan DBMS relasional yang mudah dioperasikan dan dekat dengan ekosistem MySQL. Dalam hasil penelitian ini, MariaDB menempati peringkat ketiga. Nilainya tidak sejauh Firestore Emulator dan masih cukup dekat dengan PostgreSQL pada beberapa skenario. Karena itu, MariaDB tetap dapat dipertimbangkan sebagai opsi SQL ringan, meskipun PostgreSQL lebih kuat

sebagai kandidat SQL praktis berdasarkan kombinasi performa dan kemampuan *query*.

MongoDB adalah document database yang menyimpan data dalam dokumen. Dokumentasi MongoDB menjelaskan bahwa dokumen memungkinkan struktur data yang lebih fleksibel dan dapat merepresentasikan objek aplikasi secara lebih langsung. MongoDB juga menyediakan panduan data modeling dan query optimization untuk memilih embedding, references, serta indeks sesuai pola akses [19].

Pada RIS-like *workload*, MongoDB dapat menyimpan dokumen publikasi bersama embedded researchers, SDGs, kode SDG, search tokens, dan metadata yang sering dibaca bersama. Pendekatan ini dapat mengurangi kebutuhan join pada operasi read related data dan pencarian tertentu. Namun, keuntungan tersebut bergantung pada rancangan dokumen yang digunakan dalam *adapter*. Hasil *benchmark* menunjukkan MongoDB menempati peringkat pertama pada konfigurasi yang diuji. Interpretasi ini harus dibatasi: MongoDB menjadi DBMS dengan performa terkuat pada *API-layer benchmark* ini, bukan *database* terbaik untuk semua kebutuhan. Dalam implementasi RIS aktual berbasis Laravel dan SQL, keuntungan MongoDB perlu dibandingkan dengan biaya migrasi, perubahan model data, dan perubahan pola pengembangan.

Firebase Firestore adalah document database terkelola yang menggunakan collection dan document sebagai model data. Firestore mendukung indeks, query, batas penggunaan, serta praktik perancangan data yang berbeda dari RDBMS. Namun, penelitian ini tidak menguji Firestore Cloud produksi, melainkan Firestore Emulator lokal. Firestore Emulator berguna untuk pengujian lokal, validasi fungsional, dan eksperimen tanpa menggunakan kredensial cloud. Akan tetapi, emulator tidak merepresentasikan latency jaringan, region, replikasi, quota, billing, security rules produksi, dan perilaku managed service. Oleh karena itu, semua hasil Firestore dalam penelitian ini disebut sebagai Firestore Emulator. Dalam benchmark ini, Firestore Emulator berada pada peringkat terakhir. Hasil tersebut tidak boleh disimpulkan sebagai performa Firestore Cloud. Untuk menilai Firestore

Cloud, diperlukan pengujian terpisah pada environment cloud atau staging dengan konfigurasi region, security rules, indeks, dan monitoring yang sesuai [27].

Tabel 2.6 membandingkan lima DBMS yang diuji. Perbandingan ini merangkum karakter umum, relevansi terhadap RIS, dan catatan interpretasi yang harus diperhatikan sebelum membaca hasil *benchmark*.

Tabel 2.6 Perbandingan DBMS yang diuji

DBMS	Model	Kekuatan utama	Catatan interpretasi
SQL Server	Relasional	Enterprise, transaksi, ekosistem Microsoft	Hasil lokal dipengaruhi container dan konfigurasi
PostgreSQL	Relasional	Constraint, indeks, optimizer, MVCC	Kandidat SQL praktis untuk RIS Laravel/SQL
MariaDB	Relasional	Ringan dan umum pada aplikasi web	Viable sebagai opsi SQL dengan konfigurasi sederhana
MongoDB	Dokumen	Fleksibilitas dokumen dan text index	Tercepat pada *benchmark*, tetapi migrasi model perlu dipertimbangkan
Firestore Emulator	Dokumen lokal	Pengujian Firebase lokal tanpa cloud	Tidak mewakili Firestore Cloud produksi

Tabel 2.6 memperjelas bahwa setiap DBMS dipilih untuk mewakili karakteristik yang berbeda. SQL Server mewakili enterprise RDBMS, PostgreSQL dan MariaDB mewakili kandidat SQL open-source, MongoDB mewakili document *database* lokal, dan Firestore Emulator mewakili pendekatan Firebase lokal yang harus ditafsirkan secara hati-hati.

2.4.2 Grafana K6

Grafana K6 adalah alat load testing yang memungkinkan skenario ditulis sebagai kode dan dijalankan melalui command line. Dokumentasi K6 menjelaskan metrik seperti http_req_duration, http_reqs, http_req_failed, dan checks yang digunakan untuk memantau performa request HTTP. K6 juga menyediakan mekanisme output hasil yang dapat digunakan untuk analisis lanjutan [26].

K6 sesuai untuk penelitian ini karena skenario *benchmark* relatif terstruktur, berbasis *endpoint*, dan perlu dieksekusi berulang untuk beberapa DBMS dan

skenario. Dukungan JavaScript membantu skenario S1 sampai S10 disimpan dalam repository, sedangkan environment variable memudahkan perubahan DBMS, jumlah VUs, durasi, iterasi, dan parameter *query*.

K6 tidak digunakan untuk benchmark koneksi database langsung pada penelitian ini. K6 digunakan sebagai HTTP load generator karena tujuan penelitian adalah menguji endpoint API yang merepresentasikan alur aplikasi web. Hal ini sejalan dengan dokumentasi K6 terkait API load testing[7], [8].

Apache JMeter adalah alat performance testing berbasis Java yang mendukung banyak protokol, pembuatan test plan melalui GUI, mode CLI, serta pelaporan hasil. Dalam penelitian ini, JMeter diposisikan sebagai pembanding konseptual terhadap K6, bukan alat yang digunakan pada proses benchmark.

JMeter dapat menjadi pilihan kuat ketika pengujian membutuhkan banyak protokol, test plan visual, atau laporan HTML bawaan. Namun, pada penelitian ini kebutuhan utama adalah skenario *API* yang ringkas, versionable, dan mudah dijalankan berulang dari PowerShell. Oleh karena itu, K6 lebih sesuai dengan alur repository dan script yang digunakan.

Locust adalah alat load testing yang mendefinisikan perilaku pengguna menggunakan Python. Locust diposisikan sebagai alternatif konseptual ketika pengujian membutuhkan perilaku pengguna kompleks, integrasi Python, atau distribusi beban yang fleksibel.

Locust relevan ketika pengujian membutuhkan perilaku pengguna kompleks, integrasi Python, atau distribusi beban yang fleksibel. Dalam penelitian ini, skenario lebih banyak berupa *endpoint API* tunggal yang dapat ditulis ringkas dalam JavaScript K6. Karena itu, Locust dibahas sebagai pembanding metodologis, bukan alat implementasi utama.

Tabel 2.7 membandingkan tiga alat *load testing* yang umum digunakan. Perbandingan ini digunakan untuk menjelaskan kecocokan alat terhadap kebutuhan penelitian, bukan untuk menyatakan bahwa satu alat selalu lebih unggul dalam semua konteks.

Tabel 2.7 Perbandingan Grafana K6, Apache JMeter, dan Locust

Aspek	Grafana K6	Apache JMeter	Locust
Pendekatan skenario	JavaScript sebagai kode	GUI test plan dan CLI	Python sebagai kode
Kekuatan utama	*API load testing* ringan dan mudah diotomasi	Dukungan protokol luas dan laporan HTML	User behavior fleksibel dan distributed load
Kesesuaian penelitian	Sangat sesuai untuk *endpoint* S1-S10	Dapat digunakan tetapi lebih berat untuk workflow ini	Dapat digunakan tetapi memerlukan skenario Python
Output	Summary, JSON, dan integrasi observability	JTL, HTML report, listener	Web UI, CSV, statistik runtime
Sumber teknis	Dokumentasi K6 dan metrik threshold [26]	Tidak digunakan sebagai sumber utama	Tidak digunakan sebagai sumber utama

Berdasarkan tabel tersebut, K6 dipilih karena paling sesuai dengan kebutuhan *API benchmark* yang ringkas, dapat direplikasi, dan mudah dijalankan melalui script. JMeter dan Locust tetap relevan sebagai alternatif, terutama untuk pengujian multi-protocol atau user behavior yang lebih kompleks.

2.4.3 Supporting Tools:

2.4.3.1 Node.js, TypeScript, Docker, Python, dan PowerShell:

Backend API pada penelitian ini dikembangkan menggunakan Node.js, TypeScript, dan Express. Komponen utama pada *backend* terdiri dari beberapa berkas yang mengatur konfigurasi aplikasi, server, rute *benchmark*, serta pengendali proses *benchmark*. Berkas tersebut mencakup *API*/src/app.ts, *API*/src/server.ts, *API*/src/routes/*benchmark*.routes.ts, dan *API*/src/controllers/*benchmark*.controller.ts. Melalui komponen tersebut, *backend* dapat menerima permintaan dari Grafana K6, menentukan DBMS yang diuji, lalu meneruskan operasi ke *database adapter* yang sesuai. Operasi yang disediakan pada *endpoint benchmark* meliputi insert, list, detail, update, delete, filter year, filter SDG, search, related, dan pagination.

Lingkungan pengujian dijalankan pada proyek lokal dengan bantuan Docker untuk menjalankan Microsoft SQL Server, PostgreSQL, MariaDB, dan

MongoDB. Sementara itu, Firebase Firestore diuji menggunakan Firebase CLI melalui Firestore Emulator. Backend API berjalan pada alamat localhost:3000, sedangkan Firestore Emulator dikonfigurasi menggunakan variabel FIRESTORE_EMULATOR_HOST=localhost:8080. Konfigurasi ini memungkinkan seluruh DBMS diuji dalam lingkungan lokal yang lebih mudah dikontrol dan direplikasi. Dengan demikian, setiap skenario pengujian dapat dijalankan menggunakan alur yang sama melalui endpoint API [27], [28].

Kebutuhan utama penelitian ini adalah menyediakan skenario *API benchmark* yang ringkas, mudah dipelihara, dan dapat dijalankan berulang secara konsisten. Oleh karena itu, proses pengujian dibantu menggunakan script scripts/run-all.ps1. Script tersebut menjalankan proses *health check* untuk setiap DBMS, melakukan *reset* dan *seed* dataset, kemudian mengeksekusi skenario S1 sampai S10 sebanyak tiga iterasi. Alur ini membantu memastikan bahwa setiap DBMS diuji dengan prosedur yang sama sebelum hasilnya dibandingkan. Dengan pendekatan tersebut, proses *benchmark* menjadi lebih terstruktur dan mengurangi risiko perbedaan perlakuan antar-DBMS.

Setelah proses pengujian selesai, hasil *benchmark* diolah menggunakan script analisis berbasis Python. Script scripts/analyze-results.py digunakan untuk membaca summary JSON dari K6 dan mengubah metrik utama menjadi CSV utama. Selanjutnya, script scripts/analyze_and_visualize_*benchmark*.py digunakan untuk menghasilkan *processed CSV* dan grafik analisis. Berkas hasil pengolahan tersebut mencakup *database_overall_summary.csv, scenario_summary.csv, database_ranking.csv, dan sql_vs_nosql_summary.csv.*