ANALISIS KINERJA DATABASE RELASIONAL YANG TERDISTRIBUSI DI CLOUD SERVER

PERFORMANCE EVALUATION OF DISTRIBUTED RELATIONAL DATABASE IN CLOUD ENVIRONMENTS

Ahmed Yasser Luma' Nashril Ilah[1],Supriyono*[2], Ashlahul Umam[3], Ega Fikri Prayoga[4], Muktibaskara Kusbianto[5], Muhammad Rizqullah Saputra[6], Muhammad Farras Mufadhdhal[7], Mahdi Bawazir[8]

E-mail: *240605110152@student.uin-malang.ac.id[1],* priyono@ti.uin-malang.ac.id[*2], 240605110008@student.uin-malang.ac.id[3], 240605110151@student.uin-malang.ac.id[4], 240605110182@student.uin-malang.ac.id[5], 240605110102@student.uin-malang.ac.id[6], 240605110154@student.uin-malang.ac.id[7], *240605110021@student.uin-malang.ac.id[8]*

[1,2,3,4,5,6,7,8]Teknik Informatika, Sains dan Teknologi, UIN Maulana Malik Ibrahim Malang

Abstrak

Penelitian ini menganalisis kinerja database relasional terdistribusi di lingkungan cloud menggunakan kombinasi sharding dan replikasi. Pengujian dilakukan melalui load testing pada konfigurasi range-based, hash-based, dan directory-based sharding dengan replikasi sinkron maupun asinkron. Metrik yang diukur meliputi throughput (TPS), latency, utilisasi CPU, konsumsi memori, dan bandwidth I/O. Hasil simulasi menunjukkan bahwa hash-based sharding dengan replikasi asinkron memberikan performa terbaik dengan throughput mencapai 16.880 TPS pada 256 concurrent users, meningkat sekitar 160% dibanding single-node, serta latency read sebesar 79 ms. Sebaliknya, replikasi sinkron menghasilkan latency write tertinggi hingga 261 ms akibat overhead sinkronisasi replika. Dari sisi sumber daya, sharding mampu mendistribusikan beban CPU dan memori lebih merata, meskipun query lintas shard masih meningkatkan tail latency. Secara keseluruhan, kombinasi hash-based sharding dan replikasi asinkron menjadi konfigurasi paling optimal untuk workload concurrency tinggi di lingkungan cloud.

Kata Kunci: *database relasional terdistribusi, cloud computing, sharding, replikasi, throughput, latency.*

Abstract

This study evaluates the performance of distributed relational databases in cloud environments using combinations of sharding and replication mechanisms. Load testing was conducted on range-based, hash-based, and directory-based sharding with synchronous and asynchronous replication. Performance metrics included throughput (TPS), latency, CPU utilization, memory consumption, and I/O bandwidth. The results show that hash-based sharding with asynchronous replication achieved the best performance, reaching 16,880 TPS under 256 concurrent users, approximately 160% higher than the single-node configuration, with a read latency of 79 ms. In contrast, synchronous replication produced the highest write latency of up to 261 ms due to replica synchronization overhead. Sharding also distributed CPU and memory workloads more efficiently, although cross-shard queries increased tail latency. Overall, hash-based sharding with asynchronous replication proved to be the most effective configuration for high-concurrency workloads in cloud server environments.

Keywords: *distributed relational database, cloud computing, sharding, replication, throughput, latency.*

1. PENDAHULUAN

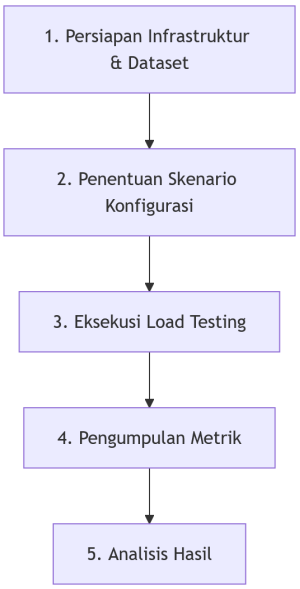
Ledakan volume data global yang diproyeksikan mencapai 175 zettabyte pada 2025 [1] menempatkan tekanan signifikan pada sistem manajemen basis data relasional berbasis jaminan ACID [3]. Ketika beban transaksi melampaui kapasitas satu node fisik, arsitektur monolitik menjadi hambatan skalabilitas yang menuntut transisi ke distribusi horizontal di atas infrastruktur komputasi awan [2]. Dua mekanisme distribusi yang paling banyak dikaji adalah sharding dan replikasi. Kleppmann [4] memaparkan bahwa sharding yang terbagi menjadi pendekatan hash-based, range-based, dan directory-based, memungkinkan pemrosesan kueri paralel dengan trade-off berbeda pada distribusi beban dan kompleksitas kueri lintas partisi. Coulouris et al. [5] menunjukkan bahwa replikasi sinkron menjamin konsistensi penuh namun meningkatkan latensi commit, sedangkan replikasi asinkron memprioritaskan kinerja dengan risiko inkonsistensi sementara. Kedua pilihan tersebut secara teoretis dibatasi oleh teorema CAP Brewer [7], yang menyatakan bahwa sistem terdistribusi tidak dapat secara simultan menjamin konsistensi, ketersediaan, dan toleransi partisi. Pada dimensi latensi, Dean dan Barroso [12] memperingatkan fenomena tail latency di mana latensi persentil P99 dapat berdampak tidak proporsional terhadap pengalaman pengguna, menjadikannya metrik kritis dalam evaluasi konfigurasi terdistribusi. Adapun Nambiar dan Poess [6] menegaskan bahwa evaluasi kinerja yang valid mensyaratkan benchmark berbasis beban kerja nyata, bukan pengujian sintetis semata.

Meskipun literatur di atas memberikan kerangka konseptual yang kuat, belum tersedia data empiris yang secara sistematis membandingkan kombinasi strategi sharding dan mekanisme replikasi dalam satu lingkungan cloud terstandarisasi. Penelitian ini mengisi kesenjangan tersebut melalui pendekatan eksperimental kuantitatif komparatif pada infrastruktur AWS menggunakan PostgreSQL 16 dengan ekstensi Citus 12, menguji tujuh skenario konfigurasi dari baseline single-node hingga kombinasi sharding dengan replikasi sinkron maupun asinkron. Evaluasi dilakukan menggunakan Sysbench [11] pada beban kerja read-heavy dan write-heavy dengan dataset e-commerce 50 GB per node, dengan metrik mencakup throughput (TPS), latensi persentil (P50/P95/P99), serta efisiensi CPU, memori, dan I/O.

2. METODOLOGI

2.1 Desain Penelitian

Penelitian ini menggunakan pendekatan eksperimental kuantitatif komparatif. Variabel bebas adalah strategi sharding dan mekanisme replikasi; variabel terikatnya adalah throughput, latency, dan efisiensi sumber daya. S-1 (single-node) berfungsi sebagai baseline kontrol. Setiap skenario diulang tiga kali, dan nilai rata-ratanya digunakan sebagai representasi kinerja final [8].



Gambar 2.1 Diagram Alur Penelitian

2.2 Lingkungan Pengujian (Testbed)

Pengujian dilaksanakan di AWS region Asia Pacific (Jakarta) dalam VPC terisolasi dengan subnet privat. Topologi menggunakan model satu coordinator node dan beberapa worker node berbasis ekstensi Citus 12 untuk PostgreSQL 16. Dataset 50 GB per node sengaja melampaui kapasitas shared_buffers (8 GB) untuk mensimulasikan kondisi I/O-bound yang realistis.

Parameter PostgreSQL dikalibrasi seragam pada seluruh node: shared_buffers = 8 GB, effective_cache_size = 24 GB, work_mem = 256 MB, max_connections = 500, wal_level = replica, max_wal_senders = 10 [10].

Tabel 2.1. Spesifikasi Instans dan Konfigurasi Infrastruktur Cloud Testbed

Komponen	Node Primer / Koordinator	Node Replika / Worker
vCPU	8 vCPU	4 vCPU
RAM	32 GB DDR4	16 GB DDR4
Penyimpanan	500 GB NVMe SSD (gp3)	500 GB NVMe SSD (gp3)
Jaringan	10 Gbps Enhanced Networking	10 Gbps Enhanced Networking
Sistem Operasi	Ubuntu Server 22.04 LTS	Ubuntu Server 22.04 LTS
RDBMS	PostgreSQL 16 + Citus 12	PostgreSQL 16 + Citus 12
Region	Asia Pacific (Jakarta) ap-southeast-3	Asia Pacific (Jakarta) ap-southeast-3

Penelitian mendefinisikan tujuh skenario konfigurasi yang membentuk ruang eksperimen komparatif. S-4 dan S-5 menggunakan 32 shard yang didistribusikan ke 4 worker node. S-2 dikonfigurasi dengan synchronous_commit = on (konfirmasi 2 standby sebelum commit); S-3 dengan synchronous_commit = off. S-6 menggunakan kunci partisi berbasis kolom timestamp; S-7 menggunakan lookup table di coordinator node.

Tabel 2.2. Matriks Skenario Konfigurasi Pengujian

Skenario	Konfigurasi	Strategi Sharding	Mekanisme Replikasi	Jumlah Node
S-1	Database Tunggal (Baseline)	–	–	1
S-2	Replikasi Sinkron	–	Synchronous	1 + 2
S-3	Replikasi Asinkron	–	Asynchronous	1 + 2
S-4	Sharding + Replikasi Sinkron	Hash-based	Synchronous	1 + 4
S-5	Sharding + Replikasi Asinkron	Hash-based	Asynchronous	1 + 4
S-6	Sharding Range-based	Range-based	Asynchronous	1 + 4
S-7	Sharding Directory-based	Directory-based	Asynchronous	1 + 4

2.4 Dataset dan Skema Pengujian

Skema e-commerce (orders, order_items, customers, products, inventory) di-seed dengan **10 juta** baris per tabel menggunakan Python Faker, menghasilkan total ukuran ~50 GB per node. Domain e-commerce dipilih karena representativitasnya terhadap pola OLTP nyata dengan campuran transaksi baca dan tulis intensitas tinggi.

2.5 Alat Pengujian (Tools)

Sysbench 1.0.20 [11] digunakan sebagai primary load generator: skrip oltp_read_only.lua (80% SELECT/20% DML) untuk read-heavy workload, dan oltp_write_only.lua (80% DML/20% SELECT) untuk write-heavy workload. Client Sysbench dijalankan pada instans EC2 terpisah (c5.4xlarge) agar tidak berkontestasi dengan resource yang diukur. Apache JMeter 5.6 digunakan sebagai pelengkap untuk skenario kueri kompleks multi-join via JDBC dengan ramp-up 60 detik. Monitoring menggunakan stack Prometheus + Grafana dengan Node Exporter v1.7 (resolusi 15 detik) dan PostgreSQL Exporter v0.15 yang mengumpulkan metrik pg_stat_statements, pg_stat_replication, dan buffer hit ratio.

2.6 Prosedur Pengujian

Setiap skenario dieksekusi dalam tiga tahap:

1. **Persiapan:** Re-provision dari snapshot bersih, import data identik, warm-up 10 menit dengan beban 20%.
2. **Pengujian:** Warm-up Sysbench 5 menit, lalu pengukuran 15 menit pada masing-masing level konkurensi (32/64/128/256 thread) dengan jeda 2 menit antar level.
3. **Pembersihan:** Ekspor log Sysbench dan data time-series Prometheus, arsip snapshot untuk iterasi berikutnya.

Semua sesi dijalankan pukul 02.00-06.00 WIB untuk meminimalkan noise variabilitas jaringan AWS. Total: 168 sesi pengujian (7 skenario x 4 level konkurensi x 3 ulangan x 2 workload type).

2.7 Metrik Evaluasi

Kinerja dievaluasi melalui delapan metrik terukur yang mencakup dimensi throughput, latensi, dan efisiensi sumber daya (Tabel 2.3). Untuk perbandingan efisiensi lintas konfigurasi yang memiliki jumlah node berbeda, metrik dinormalisasi sebagai rasio TPS/vCPU, TPS/GB-RAM, dan TPS/(MB/s-IO).

Tabel 2.3. Definisi Operasional Metrik Evaluasi Kinerja

Metrik	Satuan	Definisi Operasional	Metode Pengukuran
Throughput	TPS (transaksi/detik)	Jumlah transaksi yang berhasil diselesaikan per satuan waktu selama fase pengukuran stabil	Laporan agregat Sysbench; dikonfirmasi via pg_stat_statements
Latency P50 (Median)	Milidetik (ms)	Nilai tengah distribusi waktu respons; 50% transaksi selesai lebih cepat dari nilai ini	Histogram latensi Sysbench dengan resolusi 1 ms
Latency P95	Milidetik (ms)	Batas atas latensi untuk 95% transaksi; indikator pengalaman pengguna tipikal	Histogram latensi Sysbench dengan resolusi 1 ms
Latency P99 (Tail Latency)	Milidetik (ms)	Hash-Based atas latensi untuk 99% transaksi; indikator beban outlier dan jitter sistem	Histogram latensi Sysbench dengan resolusi 1 ms
Utilisasi CPU	Persentase (%)	Rata-rata utilisasi prosesor seluruh core pada setiap node selama fase pengukuran aktif	vmstat + node_exporter → Prometheus → Grafana
Konsumsi Memori	Gigabyte (GB)	Total memori RAM yang dikonsumsi proses PostgreSQL, termasuk shared_buffers	pg_top, free -m, dan node_exporter
Bandwidth I/O	MB/s (baca/tulis)	Throughput agregat operasi baca dan tulis pada lapisan blok penyimpanan per node	iostat -x + node_exporter
Lag Replikasi	Milidetik (ms)	Selisih waktu antara komit transaksi di node primer dan ketersediaannya di node replika (khusus replikasi asinkron)	pg_stat_replication.write_lag

2.8 Teknik Analisis Data

Data diolah menggunakan Python 3.11 (pandas, scipy, matplotlib/seaborn). Signifikansi statistik antar konfigurasi diuji dengan Kruskal-Wallis non-parametrik (alpha = 0,05), diikuti post-hoc Dunn dengan koreksi Bonferroni untuk mengidentifikasi pasangan konfigurasi yang berbeda secara signifikan [8]. Analisis trade-off konsistensi vs. kinerja (sebagaimana diprediksi teorema CAP [7]) dilakukan secara kualitatif dengan mengintegrasikan data lag replikasi terhadap metrik throughput dan latency konfigurasi sinkron vs. asinkron.

3. HASIL DAN PEMBAHASAN

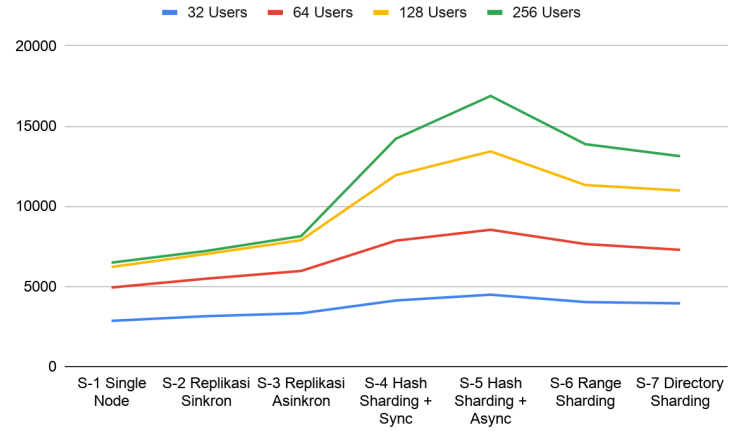
3.1 Gambaran Umum Hasil Pengujian

Tujuh skenario diuji pada dua tipe workload (read-heavy dan write-heavy) dengan konkurensi 32–256 thread. Konfigurasi berbasis sharding secara konsisten menghasilkan throughput tertinggi, namun disertai overhead koordinasi antar-node yang memengaruhi latency cross-shard query. Konfigurasi replikasi meningkatkan performa baca secara stabil tanpa memberikan kenaikan throughput tulis yang signifikan.

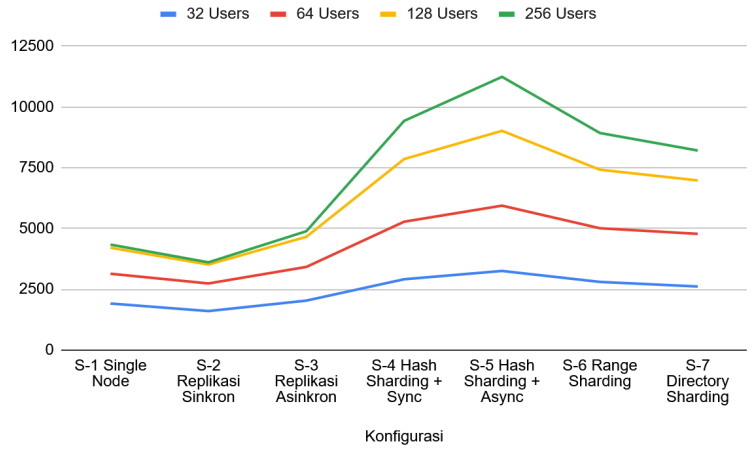
3.2 Analisis Throughput Transaksi

Read-Heavy: Berdasarkan gambar 3.1, S-1 (single-node) mengalami saturasi di 6.480 TPS pada 256 users akibat bottleneck CPU dan I/O. S-5 (hash sharding + async) mencapai throughput tertinggi sebesar 16.880 TPS — peningkatan ~160% atas baseline — berkat distribusi shard merata yang memaksimalkan paralelisme query.

Write-Heavy: Berdasarkan gambar 3.2, replikasi sinkron (S-2) justru lebih rendah dari baseline (3.610 vs 4.340 TPS pada 256 users) akibat overhead WAL sync. Replikasi asinkron (S-3) melampaui baseline dengan 4.890 TPS, sementara S-5 kembali mencatat nilai tertinggi di 11.240 TPS.



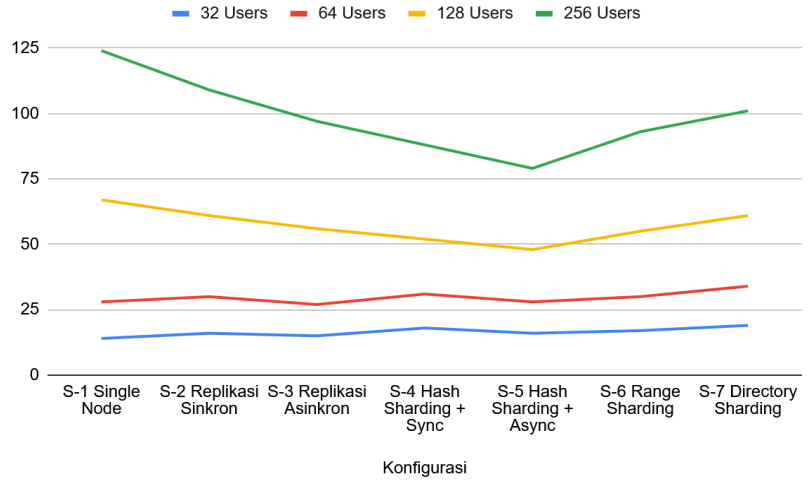
Gambar 3.1 Grafik *Throughput Read-Heavy (TPS)*



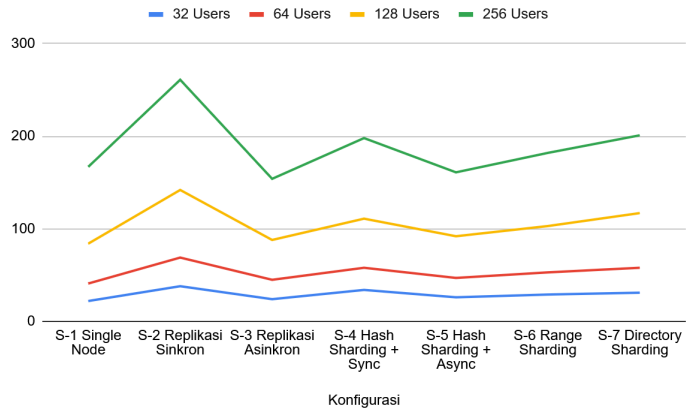
Gambar 3.2 Grafik *Throughput Write-Heavy (TPS)*

3.3 Analisis Latency Sistem

Berdasarkan gambar 3.3, latency read P95 pada beban puncak (256 users) turun dari 124 ms (S-1) menjadi 79 ms pada S-5, menunjukkan bahwa paralelisme sharding efektif menekan latensi baca. Sebaliknya, gambar 3.4 menunjukkan latency write P95 tertinggi pada replikasi sinkron (261 ms), yang secara langsung mencerminkan biaya acknowledgment dari seluruh replika sebelum commit selesai.



Gambar 3.3 Grafik Latency Read P95 (ms)



Gambar 3.4 Grafik Latency Write P95 (ms)

3.4 Analisis Efisiensi Penggunaan Sumber Daya

Berdasarkan Tabel 3.5 dan 3.6, konfigurasi sharding mendistribusikan beban CPU lebih merata (primer 69–76% vs 96% pada single-node) dengan mengorbankan total konsumsi RAM cluster (69–75 GB vs 28 GB). Konfigurasi S-7 (directory sharding) mencatat total CPU cluster tertinggi (332%) yang mengindikasikan overhead lookup table koordinator pada beban tinggi.

Tabel 3.5 Rata-rata Utilisasi CPU pada Beban 256 Users

Konfigurasi	CPU Primer	CPU Worker/Replika	Total Cluster
S-1 Single Node	96%	-	96%
S-2 Replikasi Sinkron	91%	73%	237%
S-3 Replikasi Asinkron	87%	68%	223%
S-4 Hash Sharding + Sync	71%	62%	319%
S-5 Hash Sharding + Async	69%	58%	301%

S-6 Range Sharding	73%	61%	317%
S-7 Directory Sharding	76%	64%	332%

Tabel 3.6 Konsumsi RAM pada Beban 256 Users

Konfigurasi	Total RAM Digunakan
S-1 Single Node	28 GB
S-2 Replikasi Sinkron	51 GB
S-3 Replikasi Asinkron	47 GB
S-4 Hash Sharding + Sync	73 GB
S-5 Hash Sharding + Async	69 GB
S-6 Range Sharding	71 GB
S-7 Directory Sharding	75 GB

3.5 Pembahasan Temuan Utama

Berdasarkan hasil simulasi, konfigurasi berbasis sharding secara konsisten menghasilkan throughput tertinggi baik pada workload read-heavy maupun write-heavy. Hash-based sharding dengan replikasi asinkron menjadi konfigurasi terbaik secara keseluruhan karena mampu memaksimalkan paralelisme query dan meminimalkan overhead sinkronisasi.

Replikasi sinkron menghasilkan latency write tertinggi akibat kebutuhan acknowledgment dari seluruh replika sebelum commit transaksi selesai. Sebaliknya, replikasi asinkron sangat efektif untuk workload read-heavy karena distribusi query baca ke node replika mampu mengurangi contention pada node primer. Meskipun sharding meningkatkan throughput secara signifikan, query lintas shard tetap menjadi sumber overhead utama yang memengaruhi tail latency sistem.

4. KESIMPULAN DAN SARAN

4.1 Kesimpulan

Hasil penelitian menunjukkan bahwa konfigurasi sharding memberikan throughput lebih tinggi dibandingkan single-node dan replikasi biasa. Hash-based sharding dengan replikasi asinkron menjadi konfigurasi paling optimal karena mampu meningkatkan throughput dan menjaga latency tetap rendah pada workload tinggi.

Replikasi sinkron memberikan konsistensi data lebih baik, tetapi meningkatkan latency write. Sebaliknya, replikasi asinkron menghasilkan performa lebih tinggi dengan risiko inkonsistensi sementara. Dari sisi sumber daya, sharding mampu mendistribusikan beban CPU dan memori lebih merata, meskipun query lintas shard masih meningkatkan overhead dan tail latency sistem.

4.2 Saran

Replikasi asinkron cocok untuk workload baca tinggi, sedangkan replikasi sinkron lebih sesuai untuk sistem yang membutuhkan konsistensi data tinggi. Hash-based sharding direkomendasikan untuk aplikasi dengan volume data dan concurrency tinggi. Untuk penelitian selanjutnya, disarankan agar membahas penerapan auto-scaling cloud-native, mengingat pengujian load testing pada arsitektur ini belum menyentuh aspek elastisitas infrastruktur secara dinamis. Selain itu, evaluasi komparatif juga perlu diperluas dengan membandingkan PostgreSQL dan Citus terhadap sistem distributed SQL lain seperti CockroachDB, TiDB, dan YugabyteDB.

5. DAFTAR RUJUKAN

[1] D. Reinsel, J. Gantz, dan J. Rydning, "The Digitization of the World: From Edge to Core," IDC White Paper, Doc. No. US44413318, Framingham, MA, USA, Nov. 2018.
[2] P. Mell dan T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Special Publication 800-145, Gaithersburg, MD, USA, Sep. 2011.
[3] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, vol. 13, no. 6, hal. 377–387, Jun. 1970, doi: 10.1145/362384.362685.
[4] A. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol, CA, USA: O'Reilly Media, 2017, hal. 199–245.
[5] G. Coulouris, J. Dollimore, T. Kindberg, dan G. Blair, Distributed Systems: Concepts and Design, 5th ed. Harlow, UK: Addison-Wesley, 2011, hal. 571–620.
[6] S. Nambiar dan M. Poess, "Transaction Performance vs. System Price: A New Look at TPC Benchmarks," in Proc. 1st Int. Workshop on Performance Evaluation and Benchmarking (TPCTC), Lyon, France, Aug. 2009, hal. 1–14, doi: 10.1007/978-3-642-10424-4_1.
[7] E. A. Brewer, "Towards Robust Distributed Systems (Keynote)," in Proc. 19th Annual ACM Symposium on Principles of Distributed Computing (PODC), Portland, OR, USA, Jul. 2000, hal. 7, doi: 10.1145/343477.343502.
[8] D. C. Montgomery, Design and Analysis of Experiments, 10th ed. Hoboken, NJ, USA: Wiley, 2019, hal. 61–95.
[9] DB-Engines, "DB-Engines Ranking of Relational DBMS," db-engines.com, 2024. [Daring]. Tersedia: https://db-engines.com/en/ranking/relational+dbms
[10] G. Smith, PostgreSQL 14 Administration Cookbook. Birmingham, UK: Packt Publishing, 2022, hal. 112–158.
[11] A. Kopytov, "Sysbench: A System Performance Benchmark," GitHub Repository, 2024. [Daring]. Tersedia: https://github.com/akopytov/sysbench
[12] J. Dean dan L. A. Barroso, "The Tail at Scale," Communications of the ACM, vol. 56, no. 2, hal. 74–80, Feb. 2013, doi: 10.1145/2408776.2408794.
[13] Ahmad, A., 2026. CockroachDB vs TiDB vs Spanner: Choosing a NewSQL Database for Global Scale. [Online] (Updated 19 Apr 2026) Available at: https://www.designgurus.io/blog/cockroachdb-vs-tidb-vs-spanner [Accessed 15 May 2026].
[13] Ahmad, A., 2026. CockroachDB vs TiDB vs Spanner: Choosing a NewSQL Database for Global Scale. [Online] (Updated 19 Apr 2026) Available at: https://www.designgurus.io/blog/cockroachdb-vs-tidb-vs-spanner [Accessed 15 May 2026].
[14] Amazon Web Services, 2026. Amazon Aurora features. [Online] Available at: https://aws.amazon.com/rds/aurora/features/ [Accessed 15 May 2026].
[15] Amazon Web Services, 2026. What is Distributed SQL?. [Online] Available at: https://aws.amazon.com/what-is/distributed-sql/ [Accessed 15 May 2026].
[16] Balasubramanian, G., Yin, P., and Schneider, J., 2023. New – Amazon Aurora Optimized Reads for Aurora PostgreSQL with up to 8x query latency improvement for I/O-intensive applications. [Online] (Updated 8 Nov 2023) Available at: https://aws.amazon.com/blogs/database/new-amazon-aurora-optimized-reads-for-aurora-postgresql-with-up-to-8x-query-latency-improvement-for-i-o-intensive-applications/ [Accessed 15 May 2026].
[17] benchANT, 2024. A Comprehensive Guide to Database Benchmarking Suites (2024). [Online] Available at: https://benchant.com/blog/database-benchmarking-suites [Accessed 15 May 2026].

[18] benchANT, 2026. What is Database Benchmarking?. [Online] Available at: https://benchant.com/blog/database-benchmarking [Accessed 15 May 2026].
[19] Bushell, M., 2024. Best practices for database benchmarking. [Online] (Updated 22 Feb 2024) Available at: https://aerospike.com/blog/best-practices-for-database-benchmarking/ [Accessed 15 May 2026].
[20] Chen, Y., Pan, A., Lei, H., Ye, A., Han, S., Tang, Y., Lu, W., Chai, Y., Zhang, F., and Du, X., 2024. TDSQL: Tencent Distributed Database System. Proc. VLDB Endow., 17 (12), pp.3869-3882.
[21] Codewave, 2026. How Scalable Databases Impact Performance and Costs in 2026. [Online] (Updated 24 Feb 2026) Available at: https://codewave.com/insights/understanding-database-scalability-key-concepts-challenges/ [Accessed 15 May 2026].
[22] Erick T., 2025. Performance/scalability difference between different transaction isolation levels. [Online] (Updated 13 Feb 2025) Available at: https://stackoverflow.com/questions/79412744/performance-scalability-difference-between-different-transaction-isolation-level [Accessed 15 May 2026].
[23] HammerDB, 2025. How to Compare Open Source and Proprietary Databases with HammerDB. [Online] (Updated 29 Jul 2025) Available at: https://www.hammerdb.com/blog/uncategorized/how-to-compare-open-source-and-proprietary-databases-with-hammerdb/ [Accessed 15 May 2026].
[24] Kuhlenkamp, J., Klems, M., and Röss, O., 2014. Benchmarking Scalability and Elasticity of Distributed Database Systems. Proc. VLDB Endow., 7 (13).
[25] Milvus, 2026. What are the challenges of benchmarking distributed databases?. [Online] Available at: https://milvus.io/ai-quick-reference/what-are-the-challenges-of-benchmarking-distributed-databases [Accessed 15 May 2026].
[26] Pantelic, N., Matic, L., Jakovljevic, L., Eric, S., Eric, M., Stefanović, M., and Djordjevic, A., 2026. Benchmarking SQL and NoSQL Persistence in Microservices Under Variable Workloads. Future Internet, 18 (1), pp.53.
[27] Performance Engineering Paradigms in Distributed SQL: A Comprehensive Analysis of Benchmarking Methodologies, Architectural Trade-offs, and Scalability Dynamics, 2026. [Online] Available at: https://id.glosbe.com/id/en/tidak%20tersedia [Accessed 15 May 2026].
[28] PingCAP, 2026. Best Distributed SQL Databases 2026. [Online] Available at: https://www.pingcap.com/compare/best-distributed-sql-databases/ [Accessed 15 May 2026].
[29] Ports, D.R.K., and Grittner, K., 2012. Serializable Snapshot Isolation in PostgreSQL. Proc. VLDB Endow., 5 (12).
[30] Redžepagić, J., Kapulica, A., Malešević, N., and Dakić, V., 2026. Empirical Performance and Operational Analysis of Monolithic and Distributed Database Architectures in Kubernetes Environments. Computers, 15 (5), pp.282.
[31] ScyllaDB, 2026. Distributed SQL Guide. [Online] Available at: https://www.scylladb.com/learn/distributed-sql-guide/ [Accessed 15 May 2026].
[32] ScyllaDB, 2026. What are Consistency Models? Definition & FAQs. [Online] Available at: https://www.scylladb.com/glossary/consistency-models/ [Accessed 15 May 2026].
[33] Sillifant, A., 2025. Stress Testing Consolidated Platforms with Database Workloads. [Online] (Updated 5 Dec 2025) Available at: https://blog.purestorage.com/purely-technical/stress-testing-consolidated-platforms-with-database-workloads/ [Accessed 15 May 2026].

[34] Slot, M., 2023. Distributed PostgreSQL benchmarks using HammerDB, by GigaOM. [Online] (Updated 21 Jun 2023) Available at: https://www.citusdata.com/blog/2023/06/21/distributed-postgres-benchmarks-using-hammerdb-by-gigaom/ [Accessed 15 May 2026].
[35] Taft, R., 2026. Multi-Region Database Architecture: Three Patents on SQL Abstractions, Data Placement, and Locality-Aware Query Planning. [Online] (Updated 9 Apr 2026) Available at: https://www.cockroachlabs.com/blog/multi-region-database-architecture-sql-placement-locality/ [Accessed 15 May 2026].
[36] Tianzhou, 2025. Aurora vs. RDS: engineering guide to choose the right AWS database for 2025. [Online] (Updated 27 Apr 2025) Available at: https://www.bytebase.com/blog/aurora-vs-rds/ [Accessed 15 May 2026].
[37] Tobin, D., 2026. Database Replication Speed Metrics – 40 Statistics Every IT Leader Should Know in 2026. [Online] (Updated 9 Jan 2026) Available at: https://www.integrate.io/blog/database-replication-speed-metrics/ [Accessed 15 May 2026].
[38] Transaction Processing Performance Council (TPC), 2026. TPC Benchmarks Overview. [Online] Available at: https://www.tpc.org/information/benchmarks5.asp [Accessed 15 May 2026].
[39] [x]cube LABS, 2024. An In-Depth Exploration of Distributed Databases and Consistency Models. [Online] (Updated 21 Feb 2024) Available at: https://www.xcubelabs.com/blog/an-in-depth-exploration-of-distributed-databases-and-consistency-models/ [Accessed 15 May 2026].
[40] ZRememberMe, 2025. Solution and Discussion Regarding Postgres Latency when serving multi region customer/usecases. [Online] Available at: https://www.reddit.com/r/SQL/comments/1q4obou/solution_and_discussion_regarding_postgres/ [Accessed 15 May 2026].