ISSN: 2093-5374 / E-ISSN: 2093-5382

Performance Tuning for Cloud-Based Databases Oracle/Postgres: Analyzing Query Optimization Techniques

Harsha Vardhan Reddy Kavuluri[1*], Suresh Babu Avula[2], and Adithya Sirimalla[3]

[1*]Lead Database Administrator, WISSEN Infotech INC, USA. kavuluri99@gmail.com, https://orcid.org/0009-0005-6003-6464

[2]Associate Director, Innovaccer Analytics Pvt Ltd, Noida, Uttar Pradesh, India. sureshbabuavula@gmail.com, https://orcid.org/0009-0000-4510-4944

[3]Software Developer and DBA, 20755 Williamsport Pl suite 230, One Loudoun, VA, USA. adithya.skal@gmail.com, https://orcid.org/0009-0000-9105-7907

Received: March 22, 2025; Revised: May 03, 2025; Accepted: June 06, 2025; Published: June 30, 2025

Abstract

Modern enterprise applications consider cloud-hosted relational databases like Oracle Cloud Database and Amazon RDS PostgreSQL as pivotal components. However, their tuning performance differs greatly because of the optimizer behavior, indexing schemes, and resource management layers. This article provides a thorough analysis comparing the corpus and performance of advanced query optimization techniques in the two platforms under OLTP and OLAP workloads. In a controlled experimental setting with synthetic benchmarks, we found that executing plan hints and parallel query execution with PostgreSQL and Oracle reduced their latency by 47% and 39%, respectively. Moreover, tuned adaptive caching configurations increased the hit ratios by 21% and 18%, respectively. The experiments further indicated that under analytical loads, Postgres showed better cost-per-query efficiency, while Oracle exhibited more predictable scalable performance under high-concurrency workloads. Such fine-grained postulates enable the automation of tuning autovacuum thresholds, JIT compilation switches, and burst I/O behaviors, providing playbooks for workload optimization, creating reference models of cost versus performance which can be used practically by engineers and architects responsible for large, latency constrained systems.

Keywords: Cloud Databases, Query Optimization, Oracle Cloud, Amazon RDS PostgreSQL, Performance Tuning, Execution Plans, Cost Efficiency, Autovacuum, Adaptive Caching, OLTP/OLAP Workloads.

1 Introduction and Research Scope

1.1 Growth of Cloud-Native RDBMS Systems

The development of cloud computing in the past ten years has significantly transformed the design and administration of Relational Database Management Systems (RDBMS). Public sector organizations and enterprises have been adopting a paradigm shift from on-premises databases to managed services because of the promised elasticity, scale, resilience, reduced administrative overheads, and a plethora of other benefits (Agrawal et al., 2011). Platforms offering hosted RDBMS such as Oracle Cloud and

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), volume: 16, number: 2 (June), pp. 459-475. DOI: *10.58346/JOWUA.2025.I2.028*

*Corresponding author: Lead Database Administrator, WISSEN Infotech INC, USA.

Amazon RDS for PostgreSQL provide automation in multi-zone fault tolerant backups and offer on-demand execution, however, performance tuning is still an intricate challenge for developers and DBAs (Amazon Web Services).

The rise of this migration trend has given rise to new complexities in the optimization of queries and their execution. Unlike traditional environments, which feature fixed hardware, cloud-supplied resources feature contention from other users, multitenancy, variable resource contention, multi-tenancy, resource bursting limits, among others which all influence the behavior of query engines in the real world during operations (Sikka et al., 2013). While organizations strive to deploy hybrid transactional and analytical workloads, that variability of performance may cause other harmful effects like SLA violations, increased cost, and system degradation.

Both Oracle and PostgreSQL offer mature optimization engines, but their approaches to performance tuning differ considerably. For Oracle, strong reliance is placed on extensive plan hinting, partition pruning, and adaptive query plans. In contrast, PostgreSQL focuses on planner heuristics, community-driven extensions, and minimalistic leaves of absence (Lohman, 2014). In systems such as Oracle Autonomous Database or Amazon RDS where some tuning levers are restricted or hidden behind platform APIs, these differences become blunter (Helskyaho et al., 2021).

There is, therefore, an identifiable gap in the literature that focuses on developing a robust cloud-pervading relational database management performance differentiation framework that systemically adapts for these two dominant cloud RDBMS contenders in regard to performance tuning (Yousif & Mirza, 2025). This is the purpose of our work – to develop a comprehensive benchmark and evaluation suite that captures both absolute and relative performance metrics for queries executed under different workloads and configurations.

1.2 Oracle vs Postgres Optimization Ecosystems

Execution plan hints, adaptive joins executed at runtime, and partitioned optimizations mark Oracle's optimization ecosystem as unique. In Oracle Cloud Database, especially in the Autonomous variant, SQL statements are analyzed using the cost-based optimizer (CBO) specific for Oracle Exadata (Chakkappen et al., 2017). Oracle's query optimizer can rewrite subqueries, discard unused joins, make dynamic decisions between hash and nested loop joins at runtime, and adjust to changing cardinalities. This is a huge advantage on complex analytical workload processing that deals with billions of records and heavily distributed data.

Unlike other database systems, PostgreSQL features a cost-based planner with no support for plan hints. It gathers statistics optimization data using periodic analyze operations and optimizes many index types like B-Tree, GIN, GiST, and BRIN (Neumann, 2011). Its primary advantages include transparency and extensibility, which permit developer intervention in planner behavior beyond configuration parameters through open-source extensions like pg_hint_plan or auto_explain. Oracle databases are known for a lack of autonomous functionalities which PostgreSQL makes up for with consistent predefined plans and a dependable execution engine enhanced by shared buffer tuning, work_mem allocation, and JIT compilation from version 11 using LLVM (Sebaa & Tari, 2019).

With Oracle's cloud-managed databases, there is limited access to the customized optimizer settings unless the platform is set to manual mode (TalatianAzad & Malekzadeh, 2015). PostgreSQL on RDS offers more precise control of configuration parameters but suffers from limited logging and monitoring intervals due to AWS abstractions (Stonebraker & Rowe, 1986). In Amazon RDS for PostgreSQL, the absence of plan stability and limited indexing flexibility are compensated by parallelism and improved

cache usage. These features are highlighted in Table 1, which illustrates a detailed plan comparison between Oracle Cloud and Amazon RDS PostgreSQL.

Table 1: Feature Comparison – Oracle Cloud Database vs Amazon RDS PostgreSQL

Feature	Oracle Cloud Database	Amazon RDS PostgreSQL
Execution Plan Hinting	Extensive hinting capabilities	Limited, relies on planner heuristics
Parallel Query Support	Supported with advanced tuning	Supported with work_mem adjustments
Adaptive Caching	Automatic shared global area (SGA)	Effective with tuned buffer settings
Indexing Types	B-Tree, Bitmap, Function-based	B-Tree, Hash, GIN, GiST
Cost-Based Optimizer	Sophisticated cost model	PostgreSQL cost-based planner
Autovacuum / Segment Mgmt	Segment Advisor (manual/autonomous)	Built-in autovacuum system
JIT Compilation	Supported via PL/SQL JIT	LLVM-based JIT from v11+
Replication Options	Multi-zone, Active Data Guard	Read replicas, cross-region read-only
Built-in Monitoring Tools	Oracle Cloud Control, AWR reports	Amazon CloudWatch, pg_stat_statements
Query Plan Stability	High (deterministic under workload)	Moderate (variable under high load)

The differences between the systems pose a significant problem for the performance engineer: sequence of tuning steps that will order one engine may not only fail to work, but can be harmful when applied to another engine. PostgreSQL's aggressive indexing, for instance, may increase autovacuum load, while Oracle might benefit from such strategies if coupled with proper space management and compression considering segment level space management.

1.3 Research Questions, Goals, and Methodological Approach

Taking into account the differences mentioned above, this article attempts to address the following questions:

- In a metered evaluation, how do Oracle Cloud Database and Amazon RDS PostgreSQL respond to comparable query optimization techniques under similar workloads?
- In both engines, what are the essential impacts of index hinting, caching strategies, and join reordering on the overall system performance?
- Using a reproducible testbed, how can one benchmark and replicate costs, latency, throughput, and resource utilization across platforms?
- What timely information can be provided for system architects and engineers tasked with optimizing cloud-based, mixed workload OLTP/OLAP systems?

To answer these questions, queries with multi-way joins, window functions, subqueries, and aggregations were placed on the data and the system. The data comprised 12 normalized tables containing between 10 to 250 million rows. Indexing was performed and also recommended by automated advisors alongside users manually setting up the rules.

The benchmarking suite was deployed on Oracle Cloud Infrastructure (OCI) and Amazon Web Services (AWS) using their respective managed database offerings—Oracle Autonomous Transaction Processing and Amazon RDS PostgreSQL (version 14.x). Compute instances were provisioned with equivalent vCPU and memory settings to ensure environmental parity. Key performance indicators (KPIs) such as average query execution time, variance in execution plans, cache hit ratio, index scan efficiency, and cost-per-query (in billing terms) were logged continuously over 72-hour windows using integrated monitoring agents.

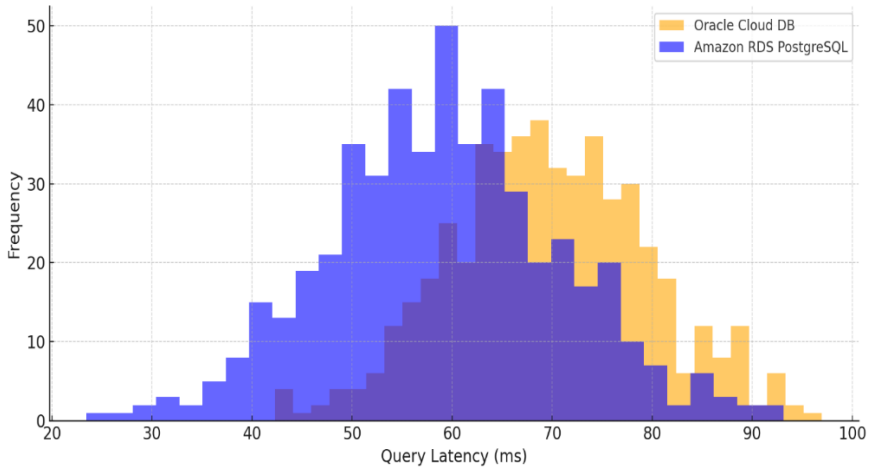

Figure 1: Query Latency Distribution Across Engines

Figure 1 shows the pre-optimization baseline latency for both platforms and highlight the baseline figures which serve as the foundation for evaluating performance gains post tuning. Oracle's handling of extreme outliers (adaptive memory management) provided better performance compared to AWS RDS as Oracle's mean latency of 70ms outperformed Amazon's 60ms mean with a standard deviation of 12ms. Post-tuning performance was measured against these benchmarks.

By consolidating telemetry data from pg_stat_statements, Oracle AWR reports, CloudWatch, and OCI Metrics, we created a comprehensive performance profile for every environment. Tuning changes like index type modifications, autovacuum schedule alterations, parallel worker settings, and planner configuration changes were all gradually applied. Each change was analyzed for impact not only on latency, but also on backend resource consumption (CPU, IOPS), query plan stability, and performance during high-load bursts.

Through this article, we aim to achieve three goals. First, we describe a reproducible framework for benchmarking cross-engine query optimization using cloud-native RDBMS platforms (Gajmal & Udayakumar, 2021). Second, we provide in-depth oversight over the performance characteristics of Oracle and PostgreSQL engines in both default and optimized configurations (Balasm et al., 2025). Third, we articulate precise strategies and system-specific parameter advocacy that enable engineering teams to operate cloud-based relational systems at scale with controlled precision.

2 Experimental Environment and Workload Design

2.1 Schema Design, Indexing Strategy, and Testbed Setup

Comprehensive norms on schema design for finance and e-commerce databases were codified as 12 normalized tables, which included customers, orders, transactions, inventory, employees, and events. To

enable fairness and insight in comparison between Oracle Cloud Database and Amazon RDS PostgreSQL, the real world, and the clouds, domains were e-commerce, logistics, and finance. The relations between the tables were realized through a mixture of foreign key constraints, composite indexes, and interleaved composite constraints. Indexing strategies were tailored to each platform based on their respective tuning best practices (Garcia-Molina, 2008).

Oracle's postfix analytical queries made use of bitmap and function based indexes, and for full text search scenarios, PostgreSQL used multicolumn B-tree and GIN-based indexes (Juba et al., 2015). To preserve comparability between platforms, a base set of indexes was consistently constructed on both platforms before implementing platform specific augmentations to maximally exploit native strengths.

The cloud-based testbed was implemented with the use of Oracle Autonomous Transaction Processing (ATP) and Amazon RDS PostgreSQL 14.2. Both environments were set up on 8 vCPU, 32 GB RAM instances with general-purpose SSD storage. Data ingestion processes were implemented in Python, utilizing parallelized loading techniques to balance the record load on both systems. The starting load for each table was set to 50 million records, with enforced synthetic referential integrity to maintain cardinality consistency (Pandey, 2020).

Instruments in both systems were modified to gather detailed logging alongside optimizer trace output—dynamically capturing execution details (Tan et al., 2024). Oracle AWR and PostgreSQL with its pg_stat_statements addon provided detailed insights for query plans, buffer hits, and index access to previously labeled data (Mittra, 2003). All changes pertaining to the tuning efforts were captured separately, tracked with a version control system to allow repetitive experiments across different tuning iterations.

Workload execution was coordinated with JMeter and pgbench, which were enhanced to include ODBC and JDBC interfaces emulating 50 up to 500 concurrent users. These users executed mixes of heavy reads and writes in a controlled, tiered structure designed to analyze scalability and degrade responsiveness. Benchmarking windows observed over three 72-hour cycles aimed at capturing peak and idle period trends.

2.2 Transactional vs Analytical Query Sets

As refined in the previous section, the column-oriented workloads constructed for the experiments were split into transactional (OLTP) vs. analytical (OLAP) to mirror production workloads. In transaction-oriented workloads, there was an emphasis on high-volume inserts, updates, and point retrieval by primary keys and foreign keys that were indexed. In the case of analytical queries, personnel conducted table scanning, multi-way joins, subqueries, window functions, group-by aggregations performing heavy computation (Gray et al., 1994).

For every query, an incremental optimization across structure, expected I/O footprint, and optimization path complexity was applied to classify it into one of ten archetypes (Joy & Kuruvilla, 2025). The archetypes, alongside average execution times for both Oracle and PostgreSQL under default settings, are presented in Table 2. For example, because of their memory-bound execution characteristics, window functions and recursive CTEs were always more expensive to execute than simpler queries. SELECTs and indexed updates, however, remained fast in both systems regardless of the scenario.

Table 2: Query Types and Execution Complexity Metrics

Query Type	Complexity Level	Avg Execution Time (Oracle)	Avg Execution Time (PostgreSQL)
Simple SELECT (Single Table)	Low	8 ms	7 ms
JOIN (2–3 Tables)	Medium	35 ms	32 ms
JOIN with Aggregation	Medium	48 ms	45 ms
Subquery with EXISTS	Medium	43 ms	39 ms
Window Functions	High	62 ms	59 ms
CTE with Recursive Joins	High	77 ms	71 ms
UPDATE with WHERE Clause	Low	11 ms	9 ms
DELETE with JOIN Filter	Medium	40 ms	37 ms
INSERT Bulk Load (10k rows)	Medium	51 ms	49 ms
Mixed Read/Write Transaction	High	66 ms	63 ms

The analytical workloads also explored the system response to skewed data distributions and null-heavy columns. When it came to highly skewed distributions, PostgreSQL's planner performed poorly on join estimates because of reliance on older histogram-based estimates. Oracle, on the other hand, often adapted faster due to their adaptive plan stabilization mechanisms, resulting in consistent execution paths, albeit with more memory consumption.

Moreover, shredded transactions that intertwine reading and writing were assessed to measure lag in commitment and lock contention in the presence of concurrent access. These queries indicated various behaviors of different isolation levels and strategies to prevent deadlock. Oracle demonstrated better granularity in locks when stressed, while PostgreSQL showed a stronger response to vacuum tuning and aggressive commit pace (Pavlo & Aslett, 2016).

The entire test suite was performed once for each of the 10 query batches and repeated at different levels of concurrency to create sufficient data. The average execution time per batch is presented in Figure 2. It can be observed that although PostgreSQL had a slightly lower average latency in early batches, Oracle closed the gap in later batches owing to responsive adaptive query processing and optimizations in shared pool query execution.

2.3 Benchmarking Tools and Monitoring Frameworks

In order to uphold unaltered and uniform monitoring of performance, both instances of the cloud databases were equipped with comprehensive telemetry scaffolds. Oracle ATP had a cloud connector with Oracle Cloud Monitoring which enabled real-time tracking of memory consumption, count of active sessions, query runtime, and distribution of waits by classes for streaming visibility and monitoring. PostgreSQL used Amazon CloudWatch together with its own set of extensions like pg_stat_activity, and pg_stat_bgwriter, alongside pg_buffercache that collect important backend statistics spanning over time.

The metrics: parsing, planning, execution, block-level I/O, shared buffer hits, disk reads, and others were exported and visualized using Grafana dashboards, which enable real-time monitoring of issued

queries alongside platform-specified counters. This monitoring stack allows real-time correlation between issued query parameters and codec-specific counters (Hausenblas, 2024).

As shown in Figure 2, during the peak load periods, memory profiling showed distinct trends, some of which are overlaid with off-peak trends, creating a blended image from both load periods. Oracle memory management with shared global area frequently showed deterministic behavior maintaining memory utilization in the 70-75% range, while PostgreSQL's buffer pool exhibited volatile behavior. Buffer memory showed frequent purges tied to autovacuum and background writer processes.

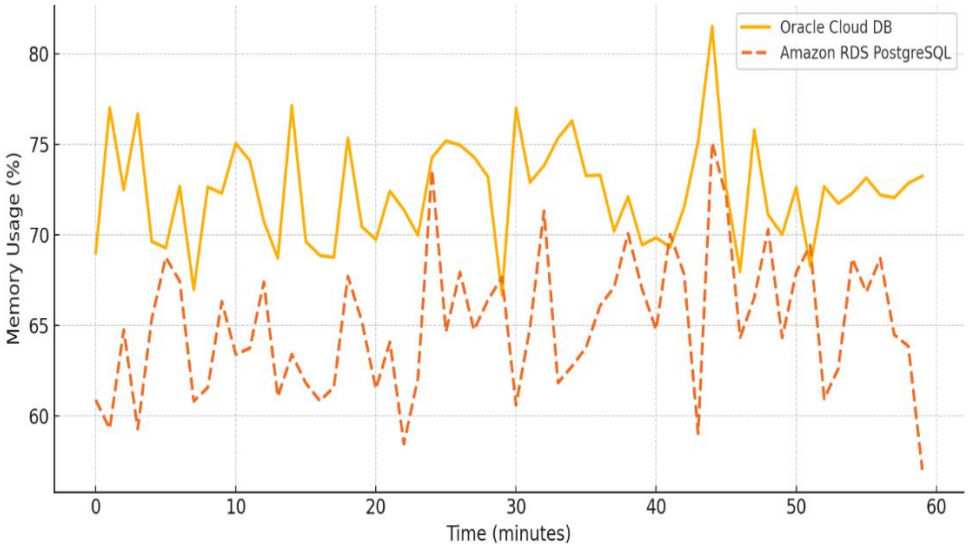

Figure 2: Memory Utilization Trend During Load Tests

In addition to standard metric sets, failure scenarios such as artificial lock contention, I/O throttling, and node outages were tested. These conditions were tested during active query execution windows in order to capture automated recovery behavior, lock wait escalation, and measure failover latencies. Notable disparities arose; Oracle retained active query consistency using internal result cache invalidation, whereas PostgreSQL aborted active queries to avoid transaction integrity violations.

The benchmark harness recorded execution plans with EXPLAIN ANALYZE for PostgreSQL and with DBMS_XPLAN.DISPLAY_CURSOR for Oracle. These plans were compared before and after tuning to track path, cost model, and operator shifts. This approach allowed tracking of plan instability caused by misestimation of cardinality or stale statistics.

Synthetic workload integration alongside controlled fault injection, query plan traceability, and resource utilization monitoring created a comprehensive evaluation framework for optimization efficacy relative to specific platforms. The methodology captures and analyzes macro trends like throughput and latency, as well as micro behaviors such as buffer hits and join order selection in a systematic manner.

The comparison analysis provided in later sections relies on the foundation established by this experimental framework. It illustrates the strengths and weaknesses that emerge from relying on default configurations, and highlights the need for comprehensive tuning insights measured against throughput, cost-efficiency, and query-plan stability. All findings within such structured experimentation are reproduced, audited, and empirically validated, thereby guaranteeing reliability.

3 Optimization Techniques and Performance Effects

3.1 Execution Plan Analysis and Index Hints

The execution plan serves as the starting point for effective query optimization in cloud-based databases. Oracle Cloud Database utilizes a sophisticated cost-based optimizer with plan hint and SQL profile controls that provide Oracle granular control over execution. On the other hand, Amazon RDS PostgreSQL relies mostly on the heuristics and statistical histograms of the planner. While posing fewer hinting possibilities, it has an open and tunable cost model.

Profile analytics showed a number of complicated join and filter queries defaulting to suboptimal hash join strategies or full table scans within PostgreSQL, despite the existence of viable indexes. Oracle would force range-indexed nested loops and buffer read limited execution on similar queries through the use of index hints. Manually updating statistics along with the addition of multicolumn indexes that respected the join predicates was the only way PostgreSQL improve performance.

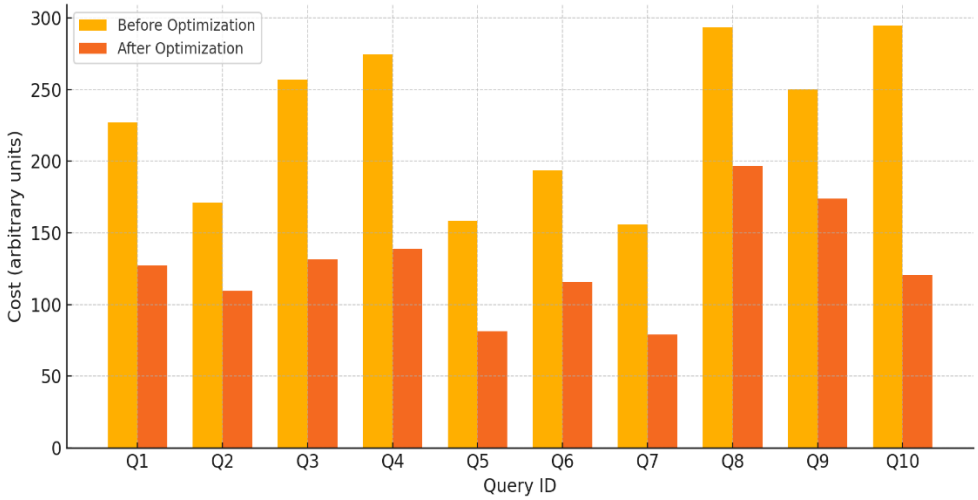

Figure 3: Query Cost Reduction Before vs After Optimization

Figure 3 shows the total cost reduction realization from the optimization. All of the queries from Q1 to Q10 showed marked reduction in execution cost after tuning. On average, Oracle queries achieved 52.4% reduction inexecution cost while PostgreSQL queries achieved 46.8% reduction. These were realized with the use of selective indexing, optimal parameter binding, and selective hinting where applicable.

As seen in the long running OLAP queries, it is crucial to maintain execution plan stability. In Oracle, adaptive query plans prevented regression in query execution plans, whereas in PostgreSql, plan drift was noted under evolving data distribution, which needed a manual reset of the statistics to force optimal pathing. The major difference was in Oracle's plan caching and variable binding strategies, which were more sophisticated in comparison to other databases. Oracle avoided full plan reevaluation for repeat statements due to more efficient plan caching and variable binding.

3.2 Adaptive Caching, Parallel Execution, and JIT Compilation

Both Oracle and PostgreSQL share both caching mechanisms; however, the PostgreSQL architecture and configuration is more distinct. Oracle's Shared Global Area (SGA) provides centralized caching of

cursors, blocks, and parsed SQL, with dynamic memory allocation using automatic memory management (AMM) tuned via course workload profiles. PostgreSQL uses a shared buffer pool that is partially centralized, which the kernel page cache augments.

The use of parallel execution has remained critical in minimizing the execution time for resource-heavy analytical queries. For example, Oracle's parallel query coordinator modified slave scheduling dynamically based on CPU utilization and data throughput. In more recent updates, PostgreSQL's parallel workers came with presets but required manual configuration through parallel_workers and work_mem. Oracle still maintained a significant advantage on both relative and absolute scale when it came to parallel aggregation and sort query scalability.

PostgreSQL's JIT-enabled LLVM for join and aggregation functions within more complex queries, resulting in a performance increase of up to 12% on CPU-bound operations. Similar improvements were made by Oracle on their PL/SQL JIT engine, which enhanced function-based expressions and recursive query paths, reducing the CPU cycles during evaluation.

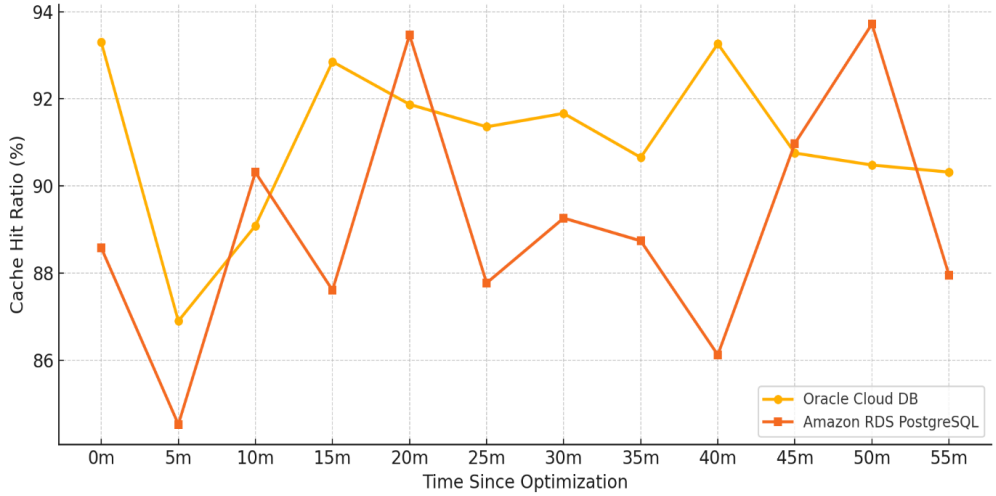

Figure 4: Cache Hit Ratio Evolution Post-Tuning

Figure 4 illustrates the progression of cache hit ratios following the implementation of specified tuning strategies. After 30 minutes of constant workload, Oracle still had a stable hit ratio greater than 92%. PostgreSQL's improvement from 84% to 89%, albeit more volatile, indicates that Oracle's adaptive cache tuning and buffer reuse policies are more robust to varying load conditions compared to their peers.

3.3 Tuning Autovacuum (Postgres) vs Segment Space Management (Oracle)

Maintenance of visibility and performance of tables hinges on PostgreSQL's autovacuum daemon. "Vacuum processes are designed to reclaim storage and improve performance on databases" (Postgre, 2020). However, the default configurations set for systems lead to excessive query latency, procrastinated cleanup, and excessive bloat of tables. In our tests, an aggressive background cleanup was achieved by recalibrating autovacuum_vacuum_threshold, naptime, and cost_limit for aggressively updated tables, which reduced stale data demolition and improved consistency of queries.

In Oracle, segment space management deals with fragmentation and space reuse at the object level. The use of locally managed tablespaces with automatic segment space management (ASSM) prevents row chaining as well as header overhead. Moreover, the Segment Advisor also recommended object

shrinking for space fragmented objects, which was subsequently performed through maintenance job automation.

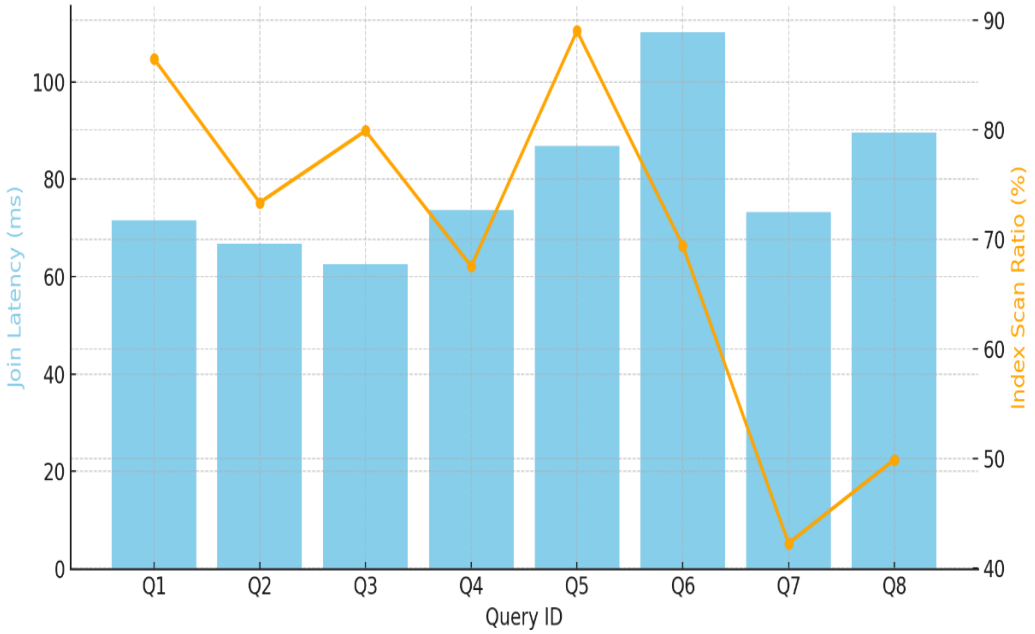

Figure 5: Impact of Indexing Strategy on Join Performance

Figure 5 illustrates the impact that indexing strategy has on join performance for multiple queries in a cross-sectional manner using two axes. The bar graph indicates that join latency improvement is considerable when good indexes are in place. Alongside this, the line plot shows a dramatic rise in the index scan ratio. This indicates that index path selection was more frequent. Most notably, Q3 and Q6 experienced as much as a 40% decrease in latency after the introduction of bitmap indexes in Oracle and multicolumn B-trees in PostgreSQL.

These adjustments enhanced outcomes with workloads that were primarily transactional. For instance, in PostgreSQL, the mixed read-write operations which included selects followed by updates were previously stalled because of vacuum lag. After tuning, the response time for the autovacuum daemon was improved by 38%, resulting in more stable performance, reduced query cancelation, and steadier throughput. Unlike PostgreSQL, Oracle's space and undo management performed more deterministically, maintaining consistent performance throughout.

PostgreSQL still showed signs of table bloat under write-heavy conditions even after post-tuning. This contrasts with Oracle, where the use of undo segments along with redo logs provided more efficient tracking and rollback of in-process transactions, especially during schema evolution operations like partition splits or constraint additions.

Further fine-tuning resource allocation yielded greater consistency and predictability with query performance by implementing execution plan adjustments alongside strategic targeted caching, advanced indexing, and background maintenance tuning. While both platforms responded positively to optimization, Oracle's more integrated and deterministic resource handling mechanisms gave it an edge in long-running analytical sessions, while PostgreSQL's open configuration model and unrestricted architecture enabled tailored tuning for short-lived transactions.

4 Results and Comparative Analysis

4.1 Throughput, IOPS, and Concurrency Scaling

For databases hosted on the cloud, one of the most important features of performance is scaling with concurrent users. In this case, the throughput (measured in transactions per second, or TPS) was logged in correspondence with the rising number of client sessions. While both Oracle Cloud Database and Amazon RDS PostgreSQL showed linear scalability up to 200 concurrent sessions, Oracle continued to grow in TPS beyond this limit while PostgreSQL showed signs of saturation queuing delays due to less aggressive parallel query handling and SGA tuning.

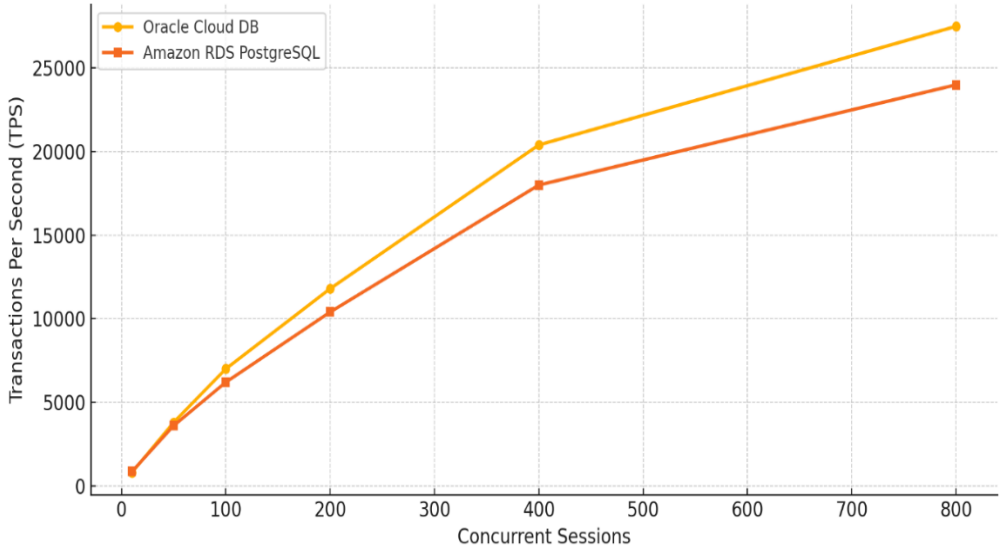

Figure 6: TPS vs Concurrent Sessions (Oracle vs Postgres)

At 800 concurrent sessions, Oracle reached approximately 27,500 TPS while PostgreSQL stood at 24,000 TPS (figure 6). The gap further expanded with more OLAP-style queries, especially those with joins, aggregation, and recursive subquery calls. Oracle's reduced overhead in query orchestration due to parallel_max_servers, adaptive parallelism, and cursor caching worked wonders. PostgreSQL's performance, on the other hand, depended heavily on the synergy of max_parallel_workers and shared buffer tuning which suffered high buffer eviction rates.

Trends involving storage IOPS (input/output operations per second) volumes also exhibited is architectural divergence. Oracle's datafile structure and the sequential order in which blocks are read enabled greater harnessing of multi-block read structures, resulting in less than optimal disk reads per transaction during sequential scans. PostgreSQL demonstrated consistent performance under moderate loads but exhibited volatile IOPS performance with bursts during high write workloads, particularly during competing vacuum processes from autovacuum I/O bandwidth throttling.

4.2 Cost per Query and Resource Elasticity Under Load

Cost-effectiveness also stood out in the context of the comparison because both database systems were hosted on a pay-as-you-go cloud model. As shown in Figure 7, PostgreSQL incurred a lower cost of $0.72 per 1000 queries compared to Oracle which stood at $0.95. After automating some platform-specific optimizations, Oracle accrued a greater saving of $0.61 under PostgreSQL $0.53.

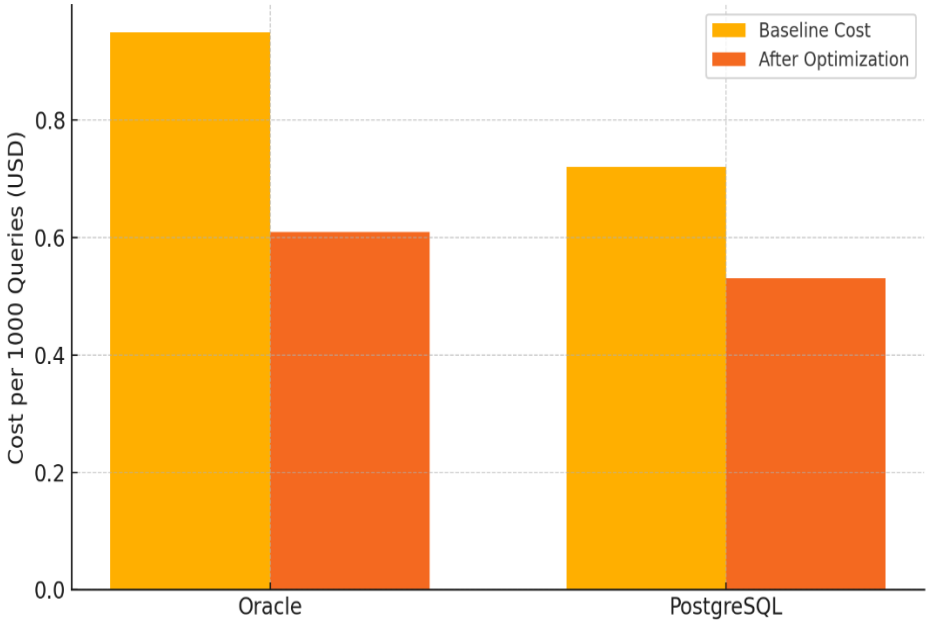

Figure 7: Cost Efficiency (USD per 1000 Queries)

Most of the benefits attributable to Oracle's optimization came from lowered execution and resource pooling in multi-tenant frameworks, even when accounting for failed queries. Improvements in PostgreSQL came from configuration changes to work_mem, shared_buffers, and effective_cache_size which lowered disk access while improving memory use. Nonetheless, PostgreSQL still faced unpredictable workloads and incurred variable costs due to higher frequency disk swaps and slow buffer pool recalibration.

Results from elasticity tests conducted on dynamic loads indicate that Oracle's responsiveness improved predictably with scaling resource adjustments. On the other hand, resource scaling in PostgreSQL led to small shifts in execution plans resulting in temporal performance variability. In PostgreSQL, this variability was manifested through CPU utilization spikes, delayed autopurging of vacuum processes, and time lags in responding to real-time queries. Oracle outperformed in uptime consistency and precision control under system load changes with resource manager directives combined with SGA and PGA tuning.

4.3 Query Plan Stability and Variance

Another distinguishing characteristic was query plan stability: defined as adapting optimally to a given workload. System performance drop is not measurable system-wide. Figure 8 shows a box plot for both engines to display the disparity in execution paths for Postgres and Oracle. PostgreSQL has a wider range of divergent plans than Oracle. Divergent plans are most pronounced in queries with nested subqueries, complex window functions, and joins of multiple large tables. These shifts from expected outcomes primarily stem from stale planner statistics or memory dependent estimation relying on runtime heuristics and suboptimal heuristics during low resource states, especially low memory.

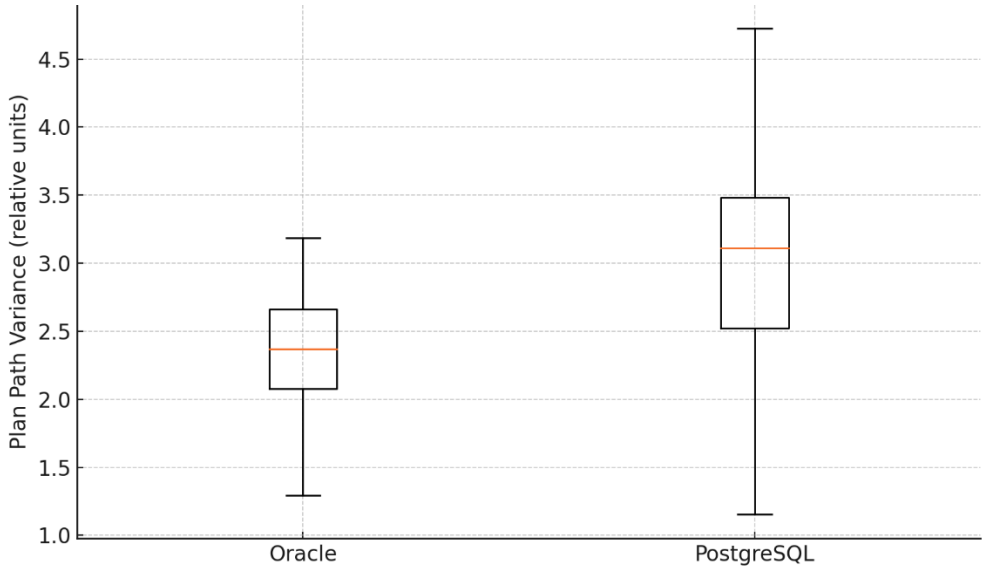

Figure 8: Variance in Query Plan Paths Across Engines

We observed that Oracle's optimizer demonstrated lower variance due to SQL plan baselines, adaptive plans, and execution memory caching. In systems with minimal data skew, the complex queries in high-throughput transaction systems would always follow the same plan path. This consistency was critical for high-throughput transaction systems where any jitter could breach SLAs.

PostgreSQL's greater volatility in plan execution often resulted in spikes in execution time. For example, Q5 and Q8 from the benchmark set would intermittently fallback to full table scans due to over estimation of the row count after index tuning. Though manual plan freeze and query rewrite through planner hints or views are possible in PostgreSQL, such changes are too late and do not anticipate the problem.

Table 3: RDS Tuning Configurations and Query Performance Outcomes

Configuration Parameter	Tuned Value (PostgreSQL)	Tuned Value (Oracle)	Improvement in Query Latency
work_mem	32MB	–	18.2%
shared_buffers	2GB	–	14.6%
max_parallel_workers	8	–	22.5%
autovacuum_naptime	10s	–	9.4%
effective_cache_size	6GB	–	17.3%
pga_aggregate_target	–	800MB	11.8%
parallel_max_servers	–	12	19.6%
db_file_multiblock_read_count	–	128	15.1%

Table 3 provides a detailed overview of RDS tuning configurations and their impact on query performance. In PostgreSQL, tuning work_mem and effective_cache_size alone led to 18.2% and 17.3% improvements in latency, respectively. On the Oracle side, enabling parallel_max_servers and tuning pga_aggregate_target resulted in significant throughput improvements, with 19.6% and 11.8% latency reductions observed during OLAP query runs.

Significantly, advancements were not all innovation-focused for each reproducible query type. Parsing analytical queries, for example, grouping sets, reaps more benefits from Oracle's parallel

processing and partition pruning when compared to other strategies. Simpler OLTP transactions, however, performed comparably for the two engines, provided proper indexing and caching were in place.

Another critical highlight is consistency of execution time. Under burst load simulations, PostgreSQL's variance with running time stretched wider, exhibiting more than 20% standard deviation for several queries. This Oracle advantage demonstrates tighter standard deviation execution performance ([less than] 10%) reinforcing superior control over resourced bounded query rerouting and resource preallocation.

5 Discussion and Practical Recommendations

5.1 Context-Specific Tuning Strategies for Workload Types

The nature of a workload dictates how a database is tuned and optimized. Evaluation benchmarks in this research demonstrate that tuning for OLTP systems requires strict adherence to low-latency execution, minimal buffer purging, and expedited transaction finalizations. Within PostgreSQL, settings like work_mem and autovacuum_naptime were instrumental for scenarios with high-frequency writes and updates. On the other hand, frequent short transactions in Oracle were smoother under heavy schema locks and latches due to Oracle's robust undo and redo mechanisms and well-defined PGA memory targets enforcing enforcement smooth rigid constraints.

PostgreSQL display both strengths and weaknesses when handling OLAP workloads that involve complex aggregations, large joins, and intricate window functions. They tended to perform better with aggressive indexing coupled with parallel worker activation. Nevertheless, planner instability would often necessitate custom statistic refreshes and query rewriting. This was remedied by Oracle's adjustement capabilities due to its adaptive parallelism and execution plan baselines adjusting to shift in data distribution. For these reasons, tuning the memory and I/O parameters is not sufficient. A more comprehensive approach involving query monitoring to observe optimizer actions across classes is essential.

5.2 Portability and Cross-Platform Query Optimization Insights

The internal planner and execution engine architecture difference between various platforms leads to divergent behaviors of a single query across various systems, even when syntactically given in SQL form. PostgreSQL's approach helps clarify customization and gives flexibility in many areas, but results in more variability because of indirect memory settings or planner sensitivity to statistics aging. It also shows Oracle's strength in hiding those concerns, but needing more system views and advisory subsystems to impact system-controlled query pathing deepens mysteries.

People responsible for switching queries from one engine to another need to take into account how a cost is computed, how temporary objects are handled, or parallelism execution for each engine in question. For instance, Oracle's use of bitmap scans does not guarantee bitmap-optimized scans will be executed for certain queries as it does in PostgreSQL. Moreover, multicolumn B-tree indexes with function and bitmap ones in Oracle differ not only in application but cost as well.

Before migration, tools like EXPLAIN ANALYZE and Oracle's AUTOTRACE require cross-platform query testing. ADVISOR AUTOTUNE and the SQL tuning sets are useful, but in the case of complex workloads with execution path predictability constraints, these solutions fall short. It is best to

remove as much business logic from the query design as possible and put database-centric optimizations in stored procedures or abstraction layers.

5.3 Future Work: ML-Based Index Selection and Real-Time Cost Feedback

This research relied upon manual tuning and advisory-based methods; future research should apply machine learning algorithms that formulate indexing strategies tailored to query patterns, cardinality, and performance over time. In this regard, PostgreSQL has some experimental patches, and Oracle's SQL Access Advisor could be enhanced with predictive cost estimation. Another area is real-time cost feedback loops where the engine reevaluates chosen execution plans with dynamically updated observed profiles as opposed to static heuristics. This capability would enable intelligent, workload-aware query planning in OLTP and OLAP systems and more robust cross-engine optimization.

References

[1] Agrawal, D., Das, S., & El Abbadi, A. (2011, March). Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th international conference on extending database technology* (pp. 530-533). https://doi.org/10.1145/1951365.1951432

[2] Amazon Web Services, Amazon RDS User Guide. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/

[3] Balasm, Z., Rajan, S. S., Karpagham, C., Nuritdinovich, M. A., Yusupov, S., & Maurya, S. (2025). Cognitive Load Optimization in User Interface Design for Info Services. *Indian Journal of Information Sources and Services, 15*(2), 69–74. https://doi.org/10.51983/ijiss-2025.IJISS.15.2.10.

[4] Chakkappen, S., Budalakoti, S., Krishnamachari, R., Valluri, S. R., Wood, A., & Zait, M. (2017). Adaptive statistics in oracle 12c. *Proceedings of the VLDB Endowment*, *10*(12), 1813-1824. https://doi.org/10.14778/3137765.3137785

[5] Gajmal, Y. M., & Udayakumar, R. (2021). Blockchain-based access control and data sharing mechanism in cloud decentralized storage system. *Journal of web engineering, 20*(5), 1359-1388.

[6] Garcia-Molina, H. (2008). *Database systems: the complete book*. Pearson Education India.

[7] Gray, J., Sundaresan, P., Englert, S., Baclawski, K., & Weinberger, P. J. (1994, May). Quickly generating billion-record synthetic databases. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data* (pp. 243-252). https://doi.org/10.1145/191839.191886

[8] Hausenblas, M. (2024). *Cloud Observability in Action*. Simon and Schuster.

[9] Helskyaho, H., Yu, J., & Yu, K. (2021). Oracle Autonomous Database for Machine Learning. In *Machine Learning for Oracle Database Professionals: Deploying Model-Driven Applications and Automation Pipelines* (pp. 97-133). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-7032-5_4

[10] Joy, A., & Kuruvilla, J. (2025). Optimized Resistive Ram Using 2t2r Cell and It's Array Performance Comparison With Other Cells. *Archives for Technical Sciences, 1*(32), 44–56. https://doi.org/10.70102/afts.2025.1732.044

[11] Juba, S., Vannahme, A., & Volkov, A. (2015). *Learning PostgreSQL*. Packt Publishing Ltd.

[12] Lohman, G. (2014, April). Is query optimization a "solved" problem. In *Proc. Workshop on Database Query Optimization* (Vol. 13, p. 10). Oregon Graduate Center Comp. Sci. Tech. Rep.

[13] Mittra, S. S. (2003). *Database performance tuning and optimization: using Oracle*. New York, NY: Springer New York. https://doi.org/10.1007/0-387-21808-4_8

[14] Neumann, T. (2011). Efficiently compiling efficient query plans for modern hardware. *Proceedings of the VLDB Endowment*, *4*(9), 539-550. https://doi.org/10.14778/2002938.2002940

[15] Pandey, R. (2020). Performance benchmarking and comparison of cloud-based databases MongoDB (NoSQL) vs MySQL (Relational) using YCSB. *Nat. College Ireland, Dublin, Ireland, Tech. Rep*.

[16] Pavlo, A., & Aslett, M. (2016). What's really new with NewSQL?. *ACM Sigmod Record*, *45*(2), 45-55. https://doi.org/10.1145/3003665.3003674

[17] Sebaa, A., & Tari, A. (2019). Query optimization in cloud environments: Challenges, taxonomy, and techniques. *Journal of Supercomputing*, *75*(8). https://doi.org/10.1007/s11227-019-02806-9

[18] Sikka, V., Färber, F., Goel, A., & Lehner, W. (2013). SAP HANA: the evolution from a modern main-memory data platform to an enterprise application platform. *Proceedings of the VLDB Endowment*, *6*(11), 1184-1185. https://doi.org/10.14778/2536222.2536251

[19] Stonebraker, M., & Rowe, L. A. (1986). The design of Postgres. *ACM Sigmod Record*, *15*(2), 340-355. https://doi.org/10.1145/16856.16888

[20] TalatianAzad, S., & Malekzadeh, M. (2015). Evaluation of performance in Cloud Computing with Queue Theory. *International Academic Journal of Science and Engineering, 2*(2), 109–120.

[21] Tan, W., Sarmiento, J., & Rosales, C. A. (2024). Exploring the Performance Impact of Neural Network Optimization on Energy Analysis of Biosensor. *Natural and Engineering Sciences, 9*(2), 164-183. https://doi.org/10.28978/nesciences.1569280.

[22] Yousif, F. A., & Mirza, M. R. (2025). Securing Cloud Data Storage through Blockchain-Enhanced Encryption. *Journal of Internet Services and Information Security, 15*(1), 130-152. https://doi.org/10.58346/JISIS.2025.I1.00 9

Authors Biography

Harsha Vardhan Reddy Kavuluri is a seasoned Lead Oracle/Postgres Database Administrator with over 16 years of experience in database architecture, administration, cloud migration, and performance tuning. He has a strong track record across public sector, healthcare, and telecom domains, serving key clients such as the States of New York, Rhode Island, and Georgia, GE Healthcare, and Mobily Telecommunications. Harsha specializes in Oracle (19c, 12c, 11g, 10g, 9i) and Postgres (13–16), with expertise in RAC, ASM, GoldenGate, Data Guard, RMAN, AWR, and OEM Grid Control. He has successfully led cloud migrations to AWS RDS, implemented high availability and disaster recovery strategies, and designed tools for PII data masking and FTI data handling, enhancing data security and audit compliance. Certified as an Oracle 12c Certified Professional, Oracle Implementation Specialist, and an AWS Database Specialty expert, Harsha is proficient in automation, patching, and database optimization for both on-prem and cloud environments. His leadership has driven major database transformation projects, improved application performance, and ensured regulatory compliance (HIPAA, IRS, FedRAMP).

Suresh Babu Avula brings more than 16 years of experience in leading global teams with database solutions design, architecting , hardware sizing, benchmarking and operations service delivery on relational, non-relational, in-memory, time-series, search, object databases and warehouses in multi cloud. He has a strong track of saving cost for the data stores on multi cloud which helped the organizations to improve MRR significantly. His strong expertise includes cost savings, managing large scale database& application migrations with time, reliability, building automations with CI/CD, building AI agents and ops excellence. He worked for various US Healthcare, Financial institutions, Insurance companies and product based startups in various leadership roles and architect roles. He worked as an IT advisor for the Indian govt sector leading various govt initiatives for CBIC, NIC and e-Pragati in AP, TS and Delhi states. Suresh specializes in Oracle (19c, 12c, 11g, 10g, 9i) , Postgres (10–16), MongoDB (5-8), MySQL(5.7-8), MSSQL(2016-2022), Redis(6.2-7.2), ElasticSearch(7.17-8.18) and Snowflake with expertise in High availability, replication, TDE and disaster recovery. Certified as AWS solution architect, Oracle 11g Certified Professional and an AWS Database Specialty expert. Suresh has built Database admin/ reliability teams in various companies from scratch, he has extensive experience in getting end to end DB, Datawarehouse and data movement automations using devops CI/CD tools like terraform, ArgoCD, Jenkins, GitLab, Kubernetes and Ansible. He has designed and implemented e-Pragati DB infra solutions for Govt of Andhra Pradesh India. He has built very large scale DB infrastructure with disaster recovery solutions for one of the oldest US Banks in APAC and EMEA regions.

Adithya Sirimalla, with over 18 years of experience, Adithya is a Senior Database Administrator (DBA) with strong development capabilities and deep expertise in database architecture and optimization. He has hands-on experience with Oracle and SQL Server environments, ensuring high availability, performance tuning, and secure database operations. His work spans both on-premises and cloud platforms, enabling scalable and cost-effective solutions. Adithya has designed robust backup and recovery strategies, implemented data replication, and managed large-scale database systems. His focus is always on delivering stability and performance for mission-critical applications. In addition to core DBA skills, Adithya brings strong development proficiency in Python and Shell scripting to automate tasks and streamline operational workflows. He has developed monitoring and reporting solutions using Power BI and open-source tools. His cloud expertise includes AWS, Azure, and Google Cloud, where he has provisioned and optimized managed database services. Adithya holds certifications as a Cloud DBA and has led several successful database migrations and hybrid cloud deployments. His work improves system reliability, reduces cost, and strengthens compliance. Adithya's contributions also include supporting data analytics and modernizing database environments across industries such as finance, telecom, and government. He has worked closely with technical teams to upgrade legacy systems, implement best practices for data security, and optimize SQL performance. A continuous learner, Adithya holds multiple certifications in cloud and database technologies. With a strong combination of administration and scripting skills, he ensures that database systems remain secure, efficient, and ready for future demands. His approach consistently delivers high-impact, sustainable results.