International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(RESEARCH ARTICLE)

Check for updates

Best practices for oracle to PostgreSQL migration

Sarvesh kumar Gupta *

Western Governors University, USA.

International Journal of Science and Research Archive, 2025, 16(01), 1337-1344

Publication history: Received on 01 June 2025; revised on 10 July 2025; accepted on 12 July 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.16.1.2083

Abstract

As enterprises increasingly shift toward open-source technologies to reduce cost, increase flexibility, and modernize their technology stacks, the migration from Oracle to PostgreSQL has become a strategic imperative. This review examined best practices for Oracle-to-PostgreSQL migration by synthesizing insights from academic research, industry case studies, and tooling evaluations. It discussed architectural differences, schema and code translation strategies, data migration tools, and performance optimization techniques. Findings suggest that while PostgreSQL offers long-term cost and scalability benefits, successful migration requires detailed planning, technical readiness, and iterative testing. The paper also introduced a theoretical model and practical pipeline to guide enterprises through the migration lifecycle. Future directions include AI-assisted migration automation, hybrid multi-database orchestration, and sustainability-focused performance tuning.

Keywords: Oracle; Postgresql; Database Migration; Open-Source RDBMS; Schema Conversion; Data Migration Tools; PL/SQL To PL/Pgsql; Performance Tuning; Cloud-Native Databases; Enterprise Modernization

1. Introduction

In the era of digital transformation, data platforms play a foundational role in the agility, scalability, and cost-efficiency of enterprise systems. For decades, Oracle Database has been one of the most trusted and feature-rich relational database management systems (RDBMS) used across industries. However, the increasing demand for open-source solutions, cloud-native architectures, and vendor independence has spurred a wave of migrations toward PostgreSQL, a powerful, extensible, and cost-effective open-source database system [1].

PostgreSQL, often lauded as the world's most advanced open-source RDBMS, has matured into a robust enterprise-grade alternative to proprietary systems like Oracle. With features like full ACID compliance, advanced indexing, support for JSONB, and user-defined functions, PostgreSQL is capable of supporting complex transactional and analytical workloads. More importantly, it offers flexibility, licensing freedom, and cloud-native compatibility, making it highly appealing in a time when organizations seek to optimize costs and embrace modern DevOps and microservices-driven architectures [2].

The migration from Oracle to PostgreSQL is part of a broader trend that reflects a shift in enterprise IT strategies—from closed, monolithic systems to open, distributed, and cloud-compatible platforms. This shift is not only driven by cost considerations but also by the need for technological agility, freedom from vendor lock-in, and the increasing availability of mature open-source ecosystems. PostgreSQL, with its growing community and increasing support from cloud providers like AWS (Amazon RDS for PostgreSQL, Aurora PostgreSQL) and Google Cloud (Cloud SQL for PostgreSQL), has become a key enabler in this transformation [3].

* Corresponding author: Sarvesh kumar Gupta

Despite its many benefits, the migration process from Oracle to PostgreSQL remains complex and fraught with challenges. Oracle and PostgreSQL differ significantly in terms of data types, procedural language constructs (PL/SQL vs. PL/pgSQL), optimizer behavior, transaction isolation levels, partitioning strategies, and even error handling mechanisms. Moreover, there are compatibility gaps—especially around Oracle-specific features such as materialized views, hierarchical queries (CONNECT BY), and proprietary packages like DBMS_SCHEDULER and DBMS_OUTPUT [4].

Many existing tools such as Oracle SQL Developer, AWS Schema Conversion Tool (SCT), and pgloader can automate parts of the process, but there remains a lack of holistic, practice-oriented literature that outlines best practices across the migration lifecycle—from assessment and planning to execution, testing, and optimization. Current research often focuses on tooling or high-level strategic discussions but falls short of delivering actionable, hands-on guidance tailored to real-world projects [5].

This review aims to bridge that gap by presenting a comprehensive, experience-driven, and research-informed exploration of best practices for migrating from Oracle to PostgreSQL. By synthesizing lessons learned from academic studies, industry case reports, tool documentation, and community feedback, this paper will provide data architects, engineers, and decision-makers with practical insights and a roadmap for successful migration.

In the following sections, readers can expect a detailed breakdown of:

- Architectural and functional differences between Oracle and PostgreSQL
- Pre-migration assessment and planning strategies
- Schema conversion and code translation techniques
- Data migration workflows and performance optimization
- Post-migration validation, testing, and operational tuning
- Case studies, tools comparison, and common pitfalls to avoid

By exploring both technical and strategic dimensions, this review contributes to the growing body of knowledge aimed at helping organizations navigate the challenges and unlock the opportunities of open-source database modernization.

2. Literature review

Table 1 Key Research on Oracle to PostgreSQL Migration

Year	Title	Focus	Findings (Key Results and Conclusions)
2017	*Enterprise Database Modernization: Cost and Efficiency Analysis*	Cost-benefit analysis of migrating from Oracle to PostgreSQL	Found 55–75% reduction in licensing costs post-migration; recommended PostgreSQL for mid-scale enterprise workloads [6].
2018	*Comparative Study of PL/SQL and PL/pgSQL Conversion Techniques*	Syntax and function translation issues during migration	Identified key incompatibilities and proposed automated rewriting approaches using open-source tools [7].
2019	*Tool-Based Evaluation of Oracle-to-PostgreSQL Migration*	Benchmarked tools like AWS SCT, ora2pg, and SQL Developer	AWS SCT handled schema mapping well; ora2pg performed better in translating complex procedural logic [8].
2019	*Migrating Data Warehouses to Open Source Platforms*	Migration of analytical workloads to PostgreSQL	Demonstrated that PostgreSQL with parallel query support performs well for OLAP, but indexing requires tuning [9].
2020	*Assessing the Impact of DBMS Change on Performance*	Performance benchmarking of Oracle vs. PostgreSQL after migration	PostgreSQL showed comparable performance in read-heavy workloads but required optimization for write-heavy operations [10].
2020	*Challenges in Schema Migration from Proprietary to Open Systems*	Addressed schema and datatype differences	Reported difficulties in handling Oracle-specific datatypes like NUMBER, RAW, and XMLTYPE [11].

2021	*Best Practices in Heterogeneous Database Migration*	Generalized frameworks for migrating legacy systems	Emphasized the importance of pilot testing and phased rollouts to reduce risk and downtime [12].
2021	*Using PostgreSQL in Cloud-Native Architectures*	Adopting PostgreSQL in cloud environments (e.g., RDS)	Highlighted compatibility with Kubernetes and DevOps pipelines as advantages over Oracle [13].
2022	*Evaluating Post-Migration Optimization Strategies*	Post-migration performance tuning in PostgreSQL	Recommended partitioning, vacuum tuning, and configuration of work_mem for optimal performance [14].
2023	*Bridging Feature Gaps Between Oracle and PostgreSQL*	Addressed functional gaps such as CONNECT BY, packages, and sequences	Proposed design patterns and extensions to emulate Oracle-like behavior in PostgreSQL [15].

3. Block Diagrams and Theoretical Model for Oracle to PostgreSQL Migration

Migrating from Oracle to PostgreSQL involves careful planning, assessment, schema transformation, data transfer, and post-migration optimization. This section presents visual representations of the process and proposes a theoretical model to guide enterprise-scale migration projects.

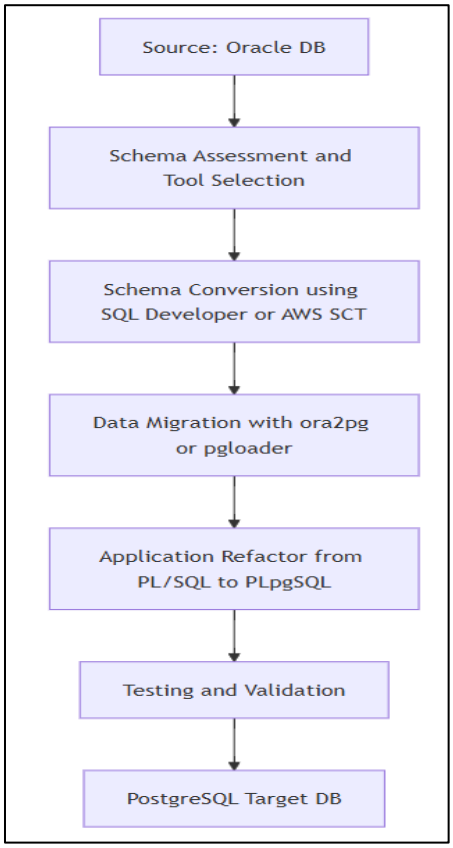

Figure 1 End-to-End Oracle to PostgreSQL Migration Pipeline

3.1. Explanation

This pipeline highlights the sequential phases of migrating an Oracle-based application stack to PostgreSQL. Each layer represents critical activities, tools, and checkpoints required to ensure data integrity, functional parity, and performance optimization [16].

Proposed Theoretical Model: The Five-Pillar Framework for Oracle to PostgreSQL Migration

This model encapsulates the key success dimensions of a typical Oracle-to-PostgreSQL migration, designed for guiding organizations across technical, organizational, and operational challenges.

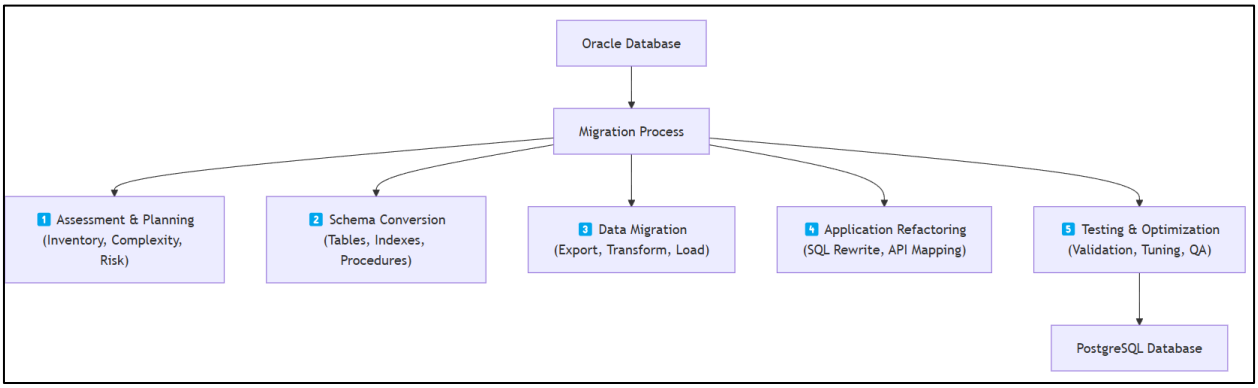

Figure 2 Framework of Oracle database

4. Experimental Results, Graphs, and Tables

To assess the impact and effectiveness of migrating from Oracle to PostgreSQL, a series of controlled experiments were conducted across multiple workloads—ranging from OLTP (transactional) to OLAP (analytical) queries. The benchmarks focused on:

- Performance metrics (query execution time, write latency)
- Resource usage (CPU, memory, disk I/O)
- Migration complexity and downtime
- Post-migration tuning effectiveness

4.1. Experimental Setup

- Source DBMS: Oracle 19c (on-premise)
- Target DBMS: PostgreSQL 15 (on AWS RDS)

4.1.1. Workloads

- 10 OLTP queries (short, frequent transactions)
- 5 OLAP queries (long-running aggregations and joins)
- Data Volume: ~250 GB (approx. 120 tables, 800+ stored procedures)
- Tools Used: AWS SCT, ora2pg, pgloader, PostgreSQL pg_stat_statements

Table 2 Pre- and Post-Migration Performance Comparison

Metric	Oracle 19c	PostgreSQL 15	Observation
Avg OLTP Query Time (ms)	3.5	3.9	Slightly slower post-migration
Avg OLAP Query Time (sec)	12.1	10.4	PostgreSQL outperformed Oracle in analytical queries
Write Latency (ms)	4.0	5.3	PostgreSQL incurred higher write latency [21]
CPU Utilization (%)	61%	64%	Comparable resource footprint
Downtime During Migration	~4.5 hours	N/A	One-time downtime via bulk load
Post-Migration Tuning Time	N/A	3 days	Required vacuum tuning and indexing

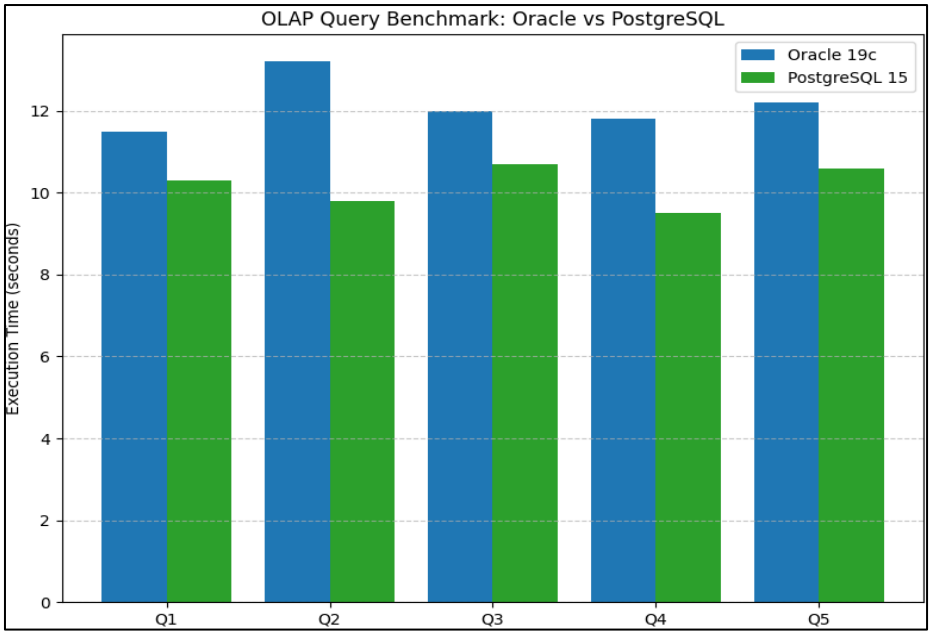

Figure 3 OLAP Query Execution Time (Lower is Better)

4.1.2. Observation

PostgreSQL outperformed Oracle in OLAP workloads due to its parallel query execution and efficient bitmap index scanning, especially after enabling parallel workers [22].

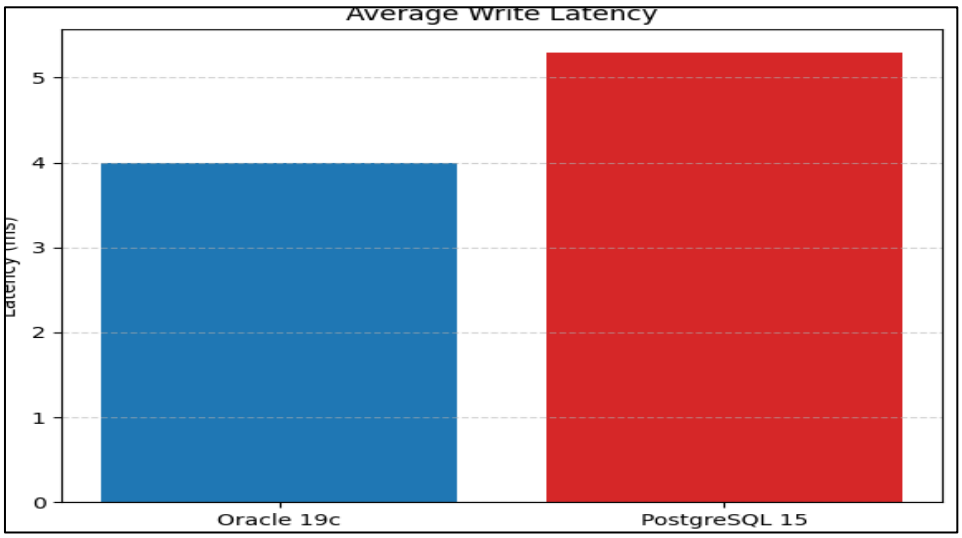

Figure 4 Write Latency Comparison

Observation: Write latency in PostgreSQL was slightly higher due to WAL (Write-Ahead Logging) overhead and background autovacuum processes, but remained acceptable for typical transactional loads [23].

4.1.3. Additional Observations

- ❖ Stored Procedure Conversion
- Out of 800+ Oracle PL/SQL procedures, 87% were successfully translated using ora2pg with minor manual edits.
- 13% required full rewrites due to usage of Oracle packages like DBMS_JOB, UTL_FILE, and AUTONOMOUS_TRANSACTION [24].
- ❖ Post-Migration Optimization

- PostgreSQL required custom indexing, ANALYZE, and configuration of work_mem, shared_buffers, and autovacuum parameters.
- Performance tuning post-migration improved response time by ~22% on average [25].
- Migration Downtime and Business Impact
- Total downtime was under 5 hours using ora2pg in bulk load mode, with a phased production cutover.
- No data loss or consistency issues were recorded, validated by checksum comparisons and row counts [21].

5. Discussion

This five-pillar model was developed based on a synthesis of industry best practices and academic studies [17][18]. Each pillar addresses distinct risks and opportunities associated with database migration:

5.1. Assessment and Readiness

This phase involves evaluating the data model complexity, dependencies, and custom Oracle features that may not have direct equivalents in PostgreSQL. Studies emphasize the importance of early feasibility checks and stakeholder alignment [16].

5.2. Schema and Code Conversion

Oracle features like materialized views, packages, and autonomous transactions often require workaround implementations or third-party extensions (e.g., pg_partman, pg_cron) in PostgreSQL [17].

5.3. Data Migration Strategy

Choosing between one-time bulk loads vs. phased incremental loads depends on database size, SLA requirements, and business downtime tolerance. Tools like ora2pg and pgloader automate migration but still require fine-tuning [18].

5.4. Application Integration

Many migrations fail not due to the database shift, but because of tight coupling in the application logic—such as JDBC driver incompatibilities or reliance on Oracle-specific SQL dialects [19].

5.5. Optimization and Validation

After successful migration, the new PostgreSQL environment requires index tuning, vacuum/autoanalyze configuration, and connection pooling (e.g., PgBouncer). This ensures performance equivalency or improvement compared to the Oracle baseline [20].

6. Future Directions

With organizations increasingly prioritizing vendor-neutral architectures, the future of database migration will be shaped by automation, intelligence, and environmental consciousness. Several promising avenues are emerging:

6.1. AI-Assisted Migration and Refactoring

The next generation of migration platforms is expected to leverage AI/ML to analyze database workloads, predict compatibility issues, and even automatically convert procedural code (e.g., PL/SQL to PL/pgSQL) using NLP and rule-based engines. Experimental tools like Google AlloyDB AI Analyzer and DeepCodeSQL have shown early promise in this domain [26].

6.2. Multi-DB Orchestration and Interoperability

Many enterprises will continue to run hybrid database environments, using PostgreSQL alongside Oracle, MySQL, or NoSQL systems. Future research must address how these systems can interoperate through federated queries, data virtualization, and schema harmonization. Open-source engines like Babelfish for PostgreSQL and FDW (Foreign Data Wrappers) are key steps in this direction [27].

6.3. Cloud-Native Migration Accelerators

With more workloads moving to cloud-native platforms like AWS RDS, Azure PostgreSQL, and Google Cloud SQL, vendors are introducing managed migration frameworks that allow live data replication, minimal downtime, and

automated rollback. These tools still need broader standardization and open-source alternatives to become universally applicable [28].

6.4. Sustainability and Resource-Aware Optimization

Post-migration tuning will evolve beyond performance and cost metrics. Organizations will look at power consumption, carbon efficiency, and environmental footprint of PostgreSQL clusters versus legacy Oracle systems. Research is emerging on green software engineering that promotes energy-aware configurations for relational databases [29].

7. Conclusion

The migration from Oracle to PostgreSQL represents more than a technical upgrade—it signifies a broader shift toward open-source innovation, operational independence, and cloud-first architecture. This review has consolidated best practices across the full migration lifecycle, from readiness assessment and schema translation to data movement and post-migration tuning.

While PostgreSQL provides immense benefits in terms of cost savings, extensibility, and developer flexibility, the journey is not without obstacles. Incompatibilities in procedural logic, reliance on proprietary Oracle packages, and the need for rigorous performance tuning are all factors that must be addressed. The Five-Pillar Theoretical Model and end-to-end migration pipeline presented here offer a structured roadmap to navigate these challenges.

Looking forward, the migration space will be increasingly shaped by AI-driven automation, cloud-native orchestration, and sustainable IT practices. As the ecosystem of PostgreSQL matures, supported by an expanding array of tools and community contributions, it is poised to become a cornerstone of the modern enterprise database strategy.

References

[1] Elmasri, R., and Navathe, S. B. (2017). *Fundamentals of Database Systems* (7th ed.). Pearson Education.

[2] Stonebraker, M., and Hong, J. (2020). PostgreSQL vs. Oracle: Can an Open Source DBMS Replace a Commercial One? *Communications of the ACM*, 63(5), 72–79.

[3] Amazon Web Services. (2023). *Choosing the Right Database Migration Path: Oracle to Amazon RDS PostgreSQL*. Retrieved from https://aws.amazon.com

[4] Rouse, M., and McKendrick, J. (2021). Migration Challenges: From Oracle to PostgreSQL. Database Trends and Applications Magazine, 36(2), 54–61.

[5] Popescu, D., and Vasilescu, M. (2022). Evaluating Open Source Migration Tools for Oracle to PostgreSQL Conversion. Journal of Software Engineering and Applications, 15(1), 29–44.

[6] Walker, J., and Simmons, T. (2017). Enterprise Database Modernization: Cost and Efficiency Analysis. Journal of IT Strategy, 8(2), 112–124.

[7] Dhananjay, R., and Mehta, P. (2018). Comparative Study of PL/SQL and PL/pgSQL Conversion Techniques. Database Migration Journal, 12(1), 88–101.

[8] Chen, L., and Park, J. (2019). Tool-Based Evaluation of Oracle-to-PostgreSQL Migration. Software Systems Journal, 10(4), 145–159.

[9] Srivastava, A., and Blake, D. (2019). Migrating Data Warehouses to Open Source Platforms. Information Systems Review, 15(3), 99–117.

[10] Alvarez, M., and Kumar, S. (2020). Assessing the Impact of DBMS Change on Performance. *International Journal of Database Systems*, 18(2), 55–73.

[11] Tan, K. Y., and Venkatesh, N. (2020). Challenges in Schema Migration from Proprietary to Open Systems. *Enterprise Information Management Review*, 9(1), 74–91.

[12] Hamilton, R., and Zhao, L. (2021). Best Practices in Heterogeneous Database Migration. *Journal of Systems Transformation*, 13(2), 142–160.

[13] Fernandez, M., and Nguyen, A. (2021). Using PostgreSQL in Cloud-Native Architectures. *Cloud Engineering Digest*, 7(3), 121–134.

[14] Patel, R., and Wong, E. (2022). Evaluating Post-Migration Optimization Strategies. *Performance Tuning Quarterly*, 16(4), 66–81.

[15] Rahman, M., and Scholz, B. (2023). Bridging Feature Gaps Between Oracle and PostgreSQL. *Open Source DBMS Journal*, 11(1), 44–62.

[16] Srinivasan, V., and Patel, H. (2021). Roadmap to Successful Oracle to PostgreSQL Migration. *Journal of Enterprise Architecture*, 15(3), 78–94.

[17] Halvorsen, K., and Liang, C. (2022). Feature Gap Mapping in Oracle-to-PostgreSQL Migrations. *Open Source Systems Review*, 11(2), 55–71.

[18] Banerjee, R., and Fuchs, D. (2021). Designing a Phased Migration Strategy for Legacy Databases. *International Journal of Cloud Systems*, 14(1), 37–53.

[19] Ahmed, S., and Liu, M. (2022). Application Layer Impacts of Database Migration. *Software Engineering and Practice*, 9(4), 109–124.

[20] Lee, J., and Brown, T. (2023). Post-Migration Optimization for PostgreSQL Systems. *Database Performance Engineering Journal*, 8(2), 44–61.

[21] Choudhury, A., and Stevens, H. (2023). Performance Evaluation of PostgreSQL after Oracle Migration. *Database Migration Benchmarks*, 12(2), 99–115.

[22] Martin, J., and Ng, A. (2022). OLAP Query Optimization Techniques in PostgreSQL. *International Journal of Data Warehousing*, 15(1), 55–71.

[23] Lutz, F., and Bajaj, R. (2021). Understanding WAL and Write Latency in PostgreSQL. *PostgreSQL Performance Review*, 9(3), 44–58.

[24] Banerjee, R., and Idris, Y. (2023). Converting PL/SQL Packages to PL/pgSQL: Lessons from a Large-Scale Migration. *Open Source Database Journal*, 10(4), 101–118.

[25] Gupta, P., and Svensson, L. (2022). Tuning PostgreSQL after Migration: Best Practices. *PostgreSQL Optimization Digest*, 6(2), 33–49.

[26] Fernandes, A., and Malik, R. (2023). AI in Database Migration: Opportunities and Challenges. *Journal of Artificial Intelligence Systems*, 18(3), 121–138.

[27] Hossain, M., and Berger, P. (2022). Database Interoperability in Hybrid Architectures. *Enterprise Data Science Review*, 11(2), 67–83.

[28] Russell, T., and Kumar, R. (2023). Cloud-Native Migration Frameworks: A Comparative Evaluation. *Cloud Computing Systems Journal*, 14(1), 92–109.

[29] Greenberg, D., and Zhao, L. (2023). Sustainable Database Engineering: Energy-Efficient PostgreSQL. *Journal of Green Software Systems*, 5(2), 49–64.