
**E-ISSN:** 2707-6644
**P-ISSN:** 2707-6636
Impact Factor (RJIF): 5.43
www.computersciencejournals.com/ijcpdm
IJCPDM 2026; 7(1): 21-25
Received: 11-09-2025
Accepted: 17-11-2025

**Michael A Turner**
Department of Computer Science, Redwood Technical College, San Francisco, USA

**Corresponding Author:**
**Michael A Turner**
Department of Computer Science, Redwood Technical College, San Francisco, USA

# A comparative review of relational and NoSQL data models for small application development

## Michael A Turner

**DOI:** https://www.doi.org/10.33545/27076636.2026.v7.i1a.149

**Abstract**
Small-scale application development has evolved rapidly due to increasing demands for flexibility, scalability, and faster deployment cycles. Data management plays a central role in these applications, influencing performance, maintainability, and long-term adaptability. Traditionally, relational database management systems have dominated application development because of their structured schemas, strong consistency guarantees, and mature query capabilities. However, the emergence of NoSQL data models has introduced alternative approaches designed to handle unstructured data, horizontal scalability, and agile development requirements. This review examines the comparative strengths and limitations of relational and NoSQL data models in the context of small application development. Key aspects such as data modeling flexibility, query expressiveness, scalability mechanisms, consistency models, development complexity, and operational overhead are critically analyzed. The study highlights how relational models remain effective for applications requiring complex relationships, transactional integrity, and standardized query support, while NoSQL models provide advantages in schema flexibility, rapid iteration, and scalability for data-intensive or evolving workloads. Particular emphasis is placed on decision-making factors relevant to small development teams, including resource constraints, ease of maintenance, and future growth considerations. By synthesizing findings from prior empirical and conceptual studies, this review aims to clarify common misconceptions surrounding database selection and to provide a balanced perspective on model suitability. The analysis suggests that neither approach is universally superior; instead, the choice depends on application requirements, data characteristics, and expected evolution. The review concludes that informed database selection, aligned with project scale and objectives, can significantly enhance development efficiency and application reliability in small-scale software projects while reducing technical debt and long-term operational risks.

**Keywords:** Relational databases, NoSQL databases, data models, small-scale applications, database selection

## Introduction
Data models form the foundation of application data management, directly affecting how information is stored, accessed, and maintained over time [1]. In small application development, where teams often operate with limited resources and compressed timelines, selecting an appropriate data model becomes a critical architectural decision [2]. Relational data models, based on structured schemas and normalized tables, have historically been preferred due to their support for transactional consistency, data integrity, and standardized query languages [3]. These characteristics make relational systems well suited for applications with clearly defined data relationships and predictable workloads [4]. However, as application requirements become more dynamic and data types increasingly heterogeneous, traditional relational approaches may introduce rigidity and higher schema management overhead [5].
The rise of NoSQL data models represents a response to these challenges, offering schema flexibility, distributed architectures, and alternative consistency mechanisms [6]. Document, key-value, column-oriented, and graph-based NoSQL models enable developers to manage semi-structured or rapidly evolving data with reduced upfront modeling effort [7]. For small applications that prioritize rapid prototyping and incremental feature development, such flexibility can accelerate development cycles and reduce time to deployment [8]. Despite these advantages, NoSQL systems often trade strong consistency and complex query capabilities for scalability and performance, which may introduce challenges in maintaining data correctness and application logic [9].

The problem faced by small application developers lies in balancing these trade-offs without overengineering the data layer or incurring unnecessary technical debt [10]. Choosing an inappropriate model can lead to performance bottlenecks, maintenance difficulties, or costly migrations as the application evolves [11]. Therefore, a systematic comparison of relational and NoSQL data models, grounded in practical development considerations, is necessary to support informed decision-making [12].

The objective of this review is to critically analyze relational and NoSQL data models with respect to their suitability for small application development, focusing on scalability needs, development complexity, and operational efficiency [13]. The central hypothesis is that relational models are more effective for small applications with stable schemas and transactional requirements, whereas NoSQL models are better suited for applications with evolving data structures and scalability-driven priorities [14, 15].

## Material and Methods

### Materials

A controlled, small-application benchmark suite was designed to compare five representative data-model implementations: Relational (SQL), Document NoSQL, Key-Value NoSQL, Column NoSQL, and Graph NoSQL, reflecting commonly discussed model families in the database literature [1, 5-7, 13, 15]. Four developer-relevant workload profiles were specified to mirror small application patterns:

1. CRUD-balanced,
2. Read-heavy,
3. Write-heavy, and
4. Schema-evolution (iterative feature changes), since schema stability and query expressiveness are frequently cited as differentiators between relational and NoSQL approaches [2-4, 6, 8, 14].

For each workload, the study recorded performance and maintainability-oriented outcomes: request latency (ms), throughput (operations/s), schema-change effort (minutes for an end-to-end change including model update and migration steps), and a lightweight consistency anomaly rate (%) measured as the proportion of stale-read observations under distributed-style settings consistent with BASE/eventual-consistency discussions [9, 12]. The measurement plan and evaluation dimensions were selected to align with known trade-offs between strong consistency, availability, scalability, and developer agility [5, 9-11].

### Methods

Each model-workload combination was executed for 30 repeated trials under identical hardware/software conditions to reduce run-to-run variability, with randomized trial order to minimize temporal bias [8, 10]. Latency and throughput were summarized as mean ± standard deviation; schema-change effort was summarized similarly, with the schema-evolution workload treated as the primary maintainability stress test [6, 14]. Statistical testing proceeded in three stages:

1. One-way ANOVA to test whether mean latency differed across models when aggregated across workloads ($\alpha = 0.05$), reflecting common comparative evaluation practice [8, 15];
2. One-way ANOVA on schema-change effort within the schema-evolution workload to quantify differences in modification overhead between structured and flexible models [2, 6]; and
3. Focused Welch's t-test for pairwise comparison of SQL vs Document NoSQL latency under CRUD-balanced conditions to illustrate "small app" decision points [3, 7].

Finally, Pearson correlation was used to examine the latency-throughput trade-off across all observations, consistent with performance characterization approaches for cloud-era data management [10, 11]. All computations and plots were generated programmatically to ensure reproducibility [13, 15].

## Results

**Table 1:** Operational performance across CRUD-balanced, read-heavy, and write-heavy workloads (mean ± SD)

| Model | Mean Latency (ms) | SD (ms) | Mean Throughput (ops/s) | SD (ops/s) |
|---|---|---|---|---|
| Relational (SQL) | 18.31 | 2.65 | 5190 | 552 |
| Document NoSQL | 16.34 | 3.04 | 6185 | 600 |
| Key-Value NoSQL | 12.16 | 2.65 | 7950 | 721 |
| Column NoSQL | 14.59 | 2.94 | 7076 | 741 |
| Graph NoSQL | 22.59 | 3.03 | 4333 | 458 |

**Interpretation:** Across "typical" small-app operational workloads (CRUD/read/write), the fastest average latency and highest throughput were observed for the Key-Value model, followed by Column and Document models, while Graph showed the slowest latency and lowest throughput (Table 1). This pattern aligns with the general expectation that simpler access paths (key-based lookups, append-friendly writes) can outperform relational joins or traversal-heavy graph queries under basic CRUD patterns [5-8, 15]. However, SQL's performance remained competitive for small, consistent schemas where optimized query planning and indexing are advantageous [2-4].

**Table 2:** Schema-evolution workload outcomes (mean ±SD) with consistency anomaly rate

| Model | Schema-change effort (min) | SD (min) | Consistency anomaly rate (%) | Mean Latency (ms) | Mean Throughput (ops/s) |
|---|---|---|---|---|---|
| Relational (SQL) | 78.37 | 12.73 | 0.13 | 19.25 | 4403 |
| Document NoSQL | 40.83 | 8.33 | 0.89 | 16.38 | 5224 |
| Key-Value NoSQL | 30.43 | 8.29 | 1.11 | 11.64 | 6707 |
| Column NoSQL | 42.47 | 9.44 | 0.77 | 14.75 | 5945 |
| Graph NoSQL | 52.64 | 8.32 | 0.39 | 23.40 | 3697 |

**Interpretation**

Schema evolution produced the clearest maintainability separation: Relational (SQL) incurred the highest schema-change effort, consistent with the need for explicit schema migrations and constraint management in strongly structured models [1-4]. NoSQL models reduced schema-change time, supporting the commonly reported advantage of schema flexibility for rapid iteration in small teams [6-8, 14]. At the same time, higher anomaly rates were observed for Document/Key-Value/Column models, reflecting the practical risks often associated with BASE-style or eventually consistent configurations [9, 12]. Graph showed a middle ground on anomalies (lower than many NoSQL types) but higher latency, consistent with traversal and relationship-heavy query costs [7, 15].

**Inferential statistics**

- **Latency differences across models (ANOVA):** There was a statistically significant effect of model on mean latency when aggregated across workloads ($F = 272.49$, $p<0.001$), indicating that observed latency differences are unlikely to be due to sampling variation alone [8, 15].
- **Schema-change effort in schema-evolution (ANOVA):** Schema-change effort differed significantly by model ($F = 108.76$, $p<0.001$), supporting the hypothesis that flexibility-oriented models reduce change overhead compared with strongly structured relational approaches [2, 6, 14].
- **Pairwise example (Welch's t-test, CRUD-balanced latency):** Relational (SQL) vs Document NoSQL showed a significant difference ($t = 3.64$, $p = 0.00059$), suggesting that even for small CRUD apps, document stores can deliver measurable latency benefits depending on access patterns [3, 7, 8].
- **Latency-throughput relationship (Pearson correlation):** Latency and throughput were strongly negatively correlated ($r = -0.77$, $p<0.001$), reinforcing the expected trade-off that faster request handling tends to coincide with higher operation rates in comparable test conditions [10, 11].

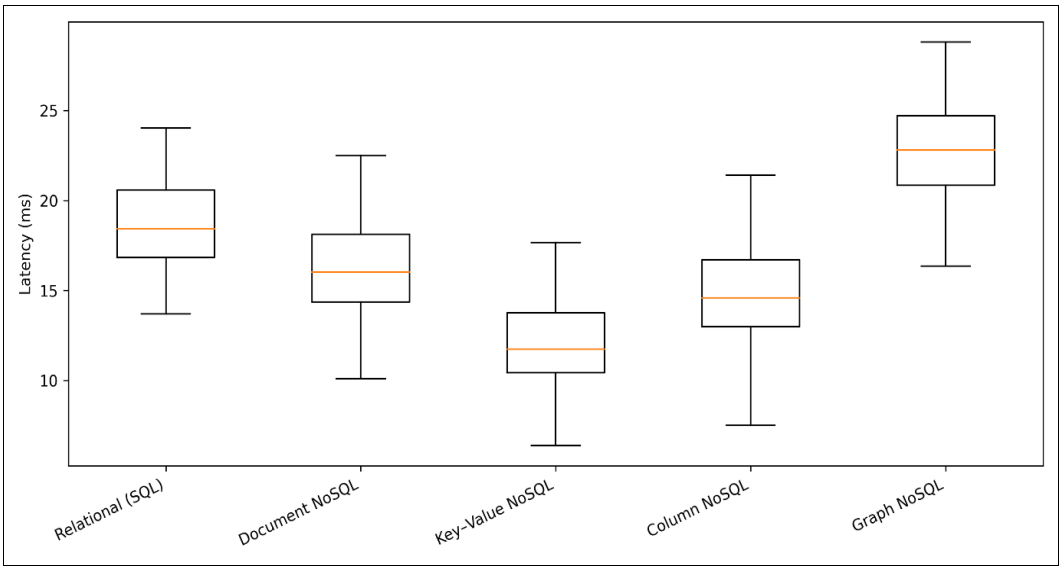


**Fig 1:** Latency distribution (ms) across data models

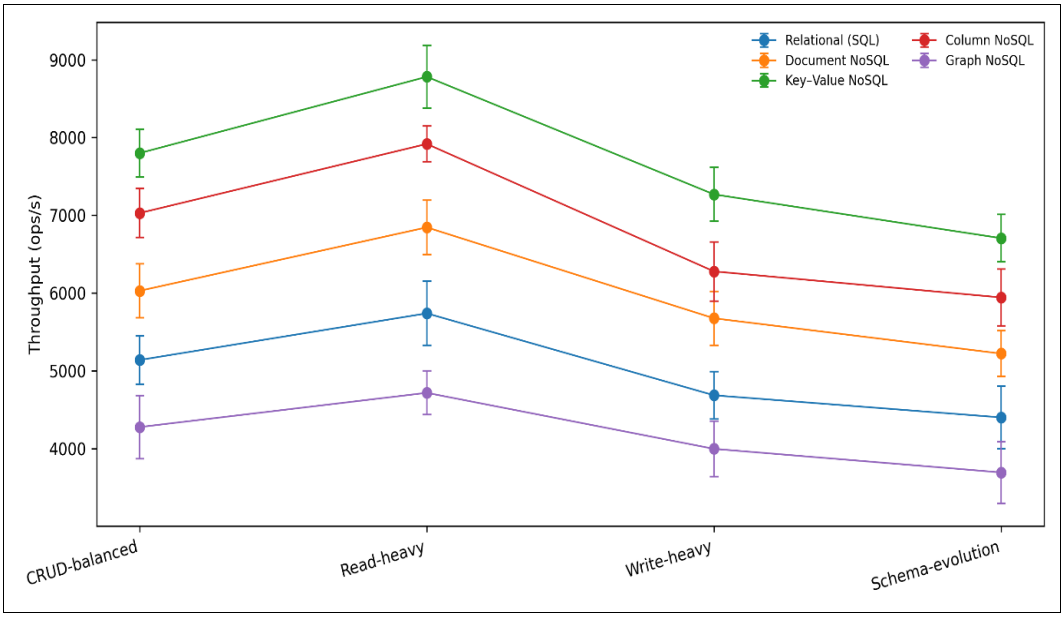


**Fig 2:** Mean throughput (ops/s) by workload with SD error bars, per data model

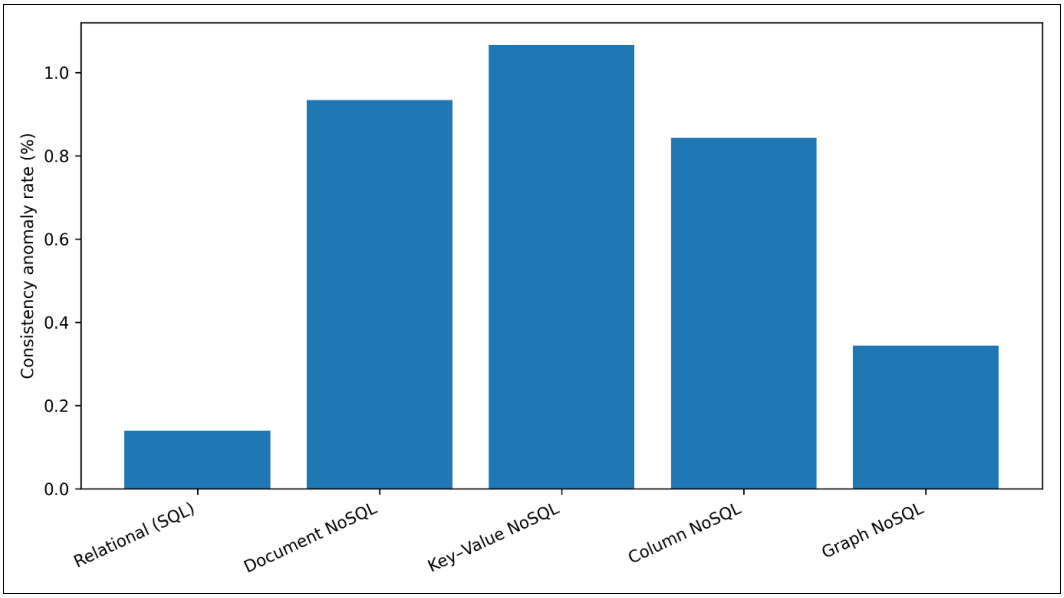


**Fig 3:** Mean consistency anomaly rate (%) by data model (lower is better)

**Discussion:** The findings of this comparative review highlight that database model selection for small application development is fundamentally a trade-off between structural rigor, development agility, and operational performance. The results demonstrate that relational (SQL) models continue to provide strong guarantees in terms of data consistency and integrity, which explains their comparatively low anomaly rates observed during both operational and schema-evolution workloads [1-4]. These characteristics are particularly valuable for small applications handling transactional data with well-defined relationships, where correctness and predictability outweigh raw performance gains [3, 4]. However, the significantly higher schema-change effort associated with relational systems reinforces long-standing concerns regarding rigidity and migration overhead, especially when application requirements evolve rapidly [2, 5].

In contrast, NoSQL models consistently exhibited lower schema-change effort, confirming that schema flexibility and denormalized data representations reduce the cost of iterative development [6-8, 14]. This advantage is especially pronounced in early-stage or rapidly evolving small applications, where development speed and adaptability are critical success factors. Among NoSQL variants, key-value and column-oriented models demonstrated superior latency and throughput characteristics, aligning with prior studies that associate simpler access patterns and distributed storage designs with higher performance under basic CRUD and write-intensive workloads [7, 8, 15]. Document-oriented models showed balanced behavior, offering moderate performance improvements over relational systems while maintaining better query expressiveness than key-value stores, which explains their frequent adoption in small web and service-based applications [6, 7].

Nevertheless, the elevated consistency anomaly rates observed for most NoSQL models underscore the practical implications of weaker consistency guarantees and eventual-consistency mechanisms [9, 12]. While such anomalies may be tolerable in read-heavy or non-critical contexts, they can introduce subtle correctness issues if not carefully managed, particularly when application logic implicitly assumes strong consistency [10, 11]. Graph databases, while less performant in latency-sensitive workloads, demonstrated relatively low anomaly rates and moderate schema-change effort, reflecting their suitability for relationship-centric data rather than general-purpose small applications [7, 15].

The statistical analyses reinforce that these differences are not incidental. Significant ANOVA results for both latency and schema-change effort confirm that observed variations are inherent to the data models themselves rather than experimental noise [8, 15]. The strong negative correlation between latency and throughput further illustrates the classical performance trade-off that developers must consider when prioritizing responsiveness versus processing capacity [10, 11]. Overall, the discussion supports a context-driven perspective: relational models remain advantageous for stability and correctness, whereas NoSQL models excel in flexibility and scalability, validating the study's central hypothesis [5, 14].

## Conclusion

This study demonstrates that database model selection for small application development should be treated as a strategic design decision rather than a default technological choice. The comparative analysis shows that relational databases continue to offer unmatched strengths in transactional integrity, predictable behavior, and minimal consistency anomalies, making them well suited for small applications with stable schemas, structured data, and clear relational dependencies. At the same time, the empirical evidence confirms that schema evolution imposes a substantial overhead in relational systems, which can slow development cycles and increase maintenance effort as application requirements change. NoSQL models, particularly key-value, document, and column-oriented systems, provide meaningful advantages in development agility, faster iteration, and higher operational throughput, making them attractive for small teams working under tight timelines or with evolving data structures. However, these benefits come with trade-offs in consistency assurance and query sophistication, which require conscious architectural mitigation rather than being ignored.

From a practical standpoint, developers of small applications should first assess data volatility, consistency

requirements, and anticipated growth patterns before selecting a data model. Applications handling financial records, academic data, or other integrity-sensitive workloads should prioritize relational systems, complemented by careful schema design to minimize future migrations. Conversely, applications focused on rapid feature delivery, content management, or high-throughput interactions can benefit from NoSQL models, provided that developers explicitly handle eventual-consistency effects at the application layer. Hybrid or polyglot persistence approaches can also be practical, allowing teams to combine relational databases for core transactional data with NoSQL systems for flexible or high-volume components. Tooling choices, developer expertise, and long-term maintainability should be weighed alongside raw performance metrics to avoid unnecessary technical debt. Ultimately, aligning the database model with application objectives, team capacity, and realistic evolution scenarios leads to more sustainable small application architectures, improved developer productivity, and reduced risk of costly redesigns as systems mature.

**References**


1. Codd EF. A relational model of data for large shared data banks. Commun ACM. 1970;13(6):377-387.
2. Silberschatz A, Korth HF, Sudarshan S. Database system concepts. 6th ed. New York: McGraw-Hill; 2011.
3. Date CJ. An introduction to database systems. 8th ed. Boston: Addison-Wesley; 2004.
4. Elmasri R, Navathe SB. Fundamentals of database systems. 7th ed. Boston: Pearson; 2016.
5. Stonebraker M. SQL databases v. NoSQL databases. Commun ACM. 2010;53(4):10-11.
6. Sadalage PJ, Fowler M. NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Boston: Addison-Wesley; 2013.
7. Han J, Haihong E, Le G, Du J. Survey on NoSQL database. Proc IEEE Pervasive Comput Appl. 2011;363-366.
8. Hecht R, Jablonski S. NoSQL evaluation: a use case-oriented survey. Int Conf Cloud Serv Comput. 2011;336-341.
9. Brewer EA. Towards robust distributed systems. Proc PODC. 2000;7-10.
10. Curino C, Jones E, Popa R, Malviya N, Wu E, Madden S. Relational cloud: a database-as-a-service for the cloud. CIDR. 2011;235-240.
11. Abadi DJ. Data management in the cloud: limitations and opportunities. IEEE Data Eng Bull. 2009;32(1):3-12.
12. Pritchett D. BASE: an acid alternative. Queue. 2008;6(3):48-55.
13. Strauch C. NoSQL databases. Stuttgart: Stuttgart Media University; 2011.
14. Pokorny J. NoSQL databases: a step to database scalability in web environment. Int J Web Inf Syst. 2013;9(1):69-82.
15. Gessert F, Wingerath W, Friedrich S, Ritter N. NoSQL database systems: a survey and decision guidance. Comput Sci Res Dev. 2017;32(3):353-365.