

Performance Analysis of Different Normal Forms in DBMS

Bhavishya Sangwan Kashish Divyansh Saraogi
Roll No. 241302170 Roll No. 241302197 Roll No. 241302227
Dept. of Computer Science & Engineering
School of Engineering and Technology
SGT University, Gurugram, India

Submitted to Ms. Kanika Malik, Assistant Professor, SOET | Subject: Database Management System (130205138) | Session 2026–2027

Abstract - Database normalization is a foundational schema-design discipline intended to eliminate redundancy and protect data integrity, yet its performance consequences are often assumed rather than measured. This report reviews normalization theory and synthesizes three recent empirical studies that measure the effect of normal-form level on query execution time, storage footprint, join overhead, and CPU utilization [1]-[3]. Unlike the traditional rule-based view, which infers performance from formal compliance with a normal form, the present analysis compares these studies side by side across redundancy, storage, join count, write efficiency, query complexity, and maintainability. The comparison shows that early-stage normalization (up to 2NF/3NF) delivers consistent storage and integrity gains, while higher forms produce workload-dependent and sometimes counter-intuitive results, including cases where additional joins do not slow decision-support queries. Based on this synthesis, the report recommends 3NF as a general baseline, selective use of BCNF/4NF, and workload profiling over rule-checklists when finalizing a schema.

Index Terms - database normalization, normal forms, query performance, schema design, DBMS, join overhead.

I. INTRODUCTION

Relational database systems remain the backbone of most information systems in use today, spanning e-commerce platforms, banking software, hospital records, and enterprise resource planning tools. Regardless of how modern the application layer looks, the reliability, speed, and consistency of the system underneath depend heavily on how the database schema itself is structured. A schema that has not been thought through carefully tends to work fine at small scale and then start showing cracks slow queries, inconsistent records, awkward updates as the data grows.

Database normalization is the discipline that addresses this at the design stage. It organizes data into related tables based on functional dependencies so that each fact is stored in exactly one place, which reduces redundancy and protects against the classic insertion, deletion, and update anomalies. What is less often examined with the same rigor is what normalization costs in terms of query performance once a

schema is actually put to use specifically, how the added JOIN operations required by a normalized design affect execution time, and whether that cost is worth the integrity gained. This report reviews normalization as a design discipline and, more importantly, examines the performance trade-offs it introduces, drawing on recent experimental studies that have measured this trade-off directly rather than treating it as a theoretical assumption [1]-[3].

II. DATABASE NORMALIZATION: CONCEPTS AND ARCHITECTURE

A. Definition and Characteristics

Database normalization was first formalized by Edgar F. Codd as part of the relational data model [6], and it remains defined the same way today: a systematic process of organizing the attributes and tables of a relational database to minimize redundancy and protect data integrity. Its defining characteristics are as follows.

- **Redundancy elimination:** Each fact is stored in only one place, so updates do not need to be repeated across multiple rows or tables.
- **Dependency-based decomposition:** Tables are split according to functional dependencies, relationships where one attribute determines the value of another, rather than arbitrary groupings.
- **Anomaly prevention:** A correctly normalized schema is structurally resistant to insertion, deletion, and update anomalies.
- **Progressive rigor:** Normalization proceeds through a series of increasingly strict forms, each of which builds on and enforces the rules of the one before it.

B. Levels of Normalization

Normalization is applied in stages, commonly described as normal forms [7]. Each level addresses a specific category of redundancy or dependency issue left unresolved by the previous one.

- **First Normal Form (1NF):** Requires that every column hold a single, atomic value and that repeating groups be eliminated from a table.

- **Second Normal Form (2NF):** Requires that every non-key attribute depend on the entire primary key, removing partial dependencies that occur with composite keys.
- **Third Normal Form (3NF):** Removes transitive dependencies, where a non-key attribute depends on another non-key attribute instead of directly on the key.
- **Boyce-Codd Normal Form (BCNF):** A stricter version of 3NF that resolves certain edge cases involving overlapping candidate keys.
- **Fourth Normal Form (4NF):** Addresses multi-valued dependencies, ensuring a table does not hold two or more independent multi-valued facts about the same entity, a concept formally introduced by Fagin [8].

C. The Normalization Process

Converting an unstructured or poorly structured dataset into a normalized schema generally follows a repeatable three-stage process: analysis, decomposition, and validation.

- **Analyze:** Identify the functional dependencies among attributes, along with candidate keys, primary keys, and foreign keys.
- **Decompose:** Split tables according to those dependencies so that partial and transitive dependencies are removed at each successive normal form.
- **Validate:** Confirm that the resulting tables actually satisfy the rules of the target normal form before the schema is finalized.

D. Schema Design Approaches

In practice, not every system is normalized to the same degree, and the choice of schema design approach depends on the workload the database is expected to serve.

- **Fully normalized schema:** Typically used for transactional (OLTP) systems where frequent writes and strong consistency matter more than read speed.
- **Partially denormalized schema:** Selectively reintroduces some redundancy, often through summary or lookup tables, to speed up specific, frequently run queries.
- **Fully denormalized / flat schema:** Common in read-heavy analytical or reporting contexts, where query speed is prioritized over storage efficiency and write-side integrity.

III. TECHNICAL ASPECTS AND RELEVANCE

Normalization theory is decades old, but its performance implications continue to receive comparatively little empirical attention; most treatments explain why joins add cost without measuring what that cost actually looks like. Recent experimental work has started to close that gap. Nagesh [1] implemented a retail transaction dataset at Unnormalized Form, 1NF, 2NF, and 3NF and measured query execution time, storage usage, join complexity, and CPU utilization directly, finding that normalization reduced storage and CPU usage while increasing the number of joins required per query. This line of inquiry is not entirely new:

an earlier comparative study by Sobol, Kagan, and Shimura measured storage space, access time, and operation counts across third- and fourth-normal-form implementations of the same queries, anticipating several of the measurement categories used in the more recent studies discussed here [5].

Taipalus [2] ran a more granular experiment using the Internet Movie Database dataset on PostgreSQL, and the results were less predictable than conventional wisdom suggests: moving from 1NF to 2NF reduced database size by roughly 10 percent and increased query throughput by a factor of four, while further normalization from 2NF to 4NF increased storage requirements by about 7 percent with only marginal throughput gains. Fotache et al. [3] tested normalization effects on decision support databases across three commercial database servers and found that higher normal forms actually improved both query completion success and execution time in several cases, a result that runs counter to the long-standing assumption that fewer joins always mean faster reads.

Taken together, these studies suggest the relationship between normalization level and performance is more workload-dependent than the traditional “more joins, slower queries” heuristic implies, which is precisely why continued empirical testing, rather than reliance on theoretical assumption, remains relevant to modern database design, including in cloud-native platforms such as PostgreSQL, MySQL, and managed cloud database services where indexing and query optimizers can substantially offset join overhead. This pattern is consistent with a broader observation from a systematic review of DBMS performance literature, which found that performance benchmarking in this area is frequently reported without sufficient methodological detail to support direct comparison across studies [10]; readers seeking a more complete theoretical treatment of normal-form theory may refer to Date's dedicated treatment of the subject [9].

IV. INNOVATION

Several developments are pushing normalization research and practice beyond its traditional theoretical boundaries.

- **Automated schema normalization:** Algorithms, including genetic algorithms, can automatically identify functional dependencies and normalize database schemas.
- **Normalization-aware NL-to-SQL:** Schema design influences AI-generated SQL; denormalized schemas suit simple retrieval, while normalized schemas can perform better for complex aggregations [4].
- **Energy-aware design:** Early normalization can reduce energy consumption per transaction, making schema design important for both performance and sustainability.
- **Adaptive and hybrid schemas:** Modern databases combine normalization and denormalization using materialized views, caching, and selective denormalization.

- **In-memory and HTAP systems:** Advanced storage engines optimize joins, reducing the performance overhead traditionally associated with highly normalized schemas.

V. SOCIETAL IMPACT

The practical implications of normalization decisions extend well beyond individual applications.

- **Healthcare:** Well-normalized hospital and patient record systems reduce the risk of conflicting or duplicated medical data, which has direct consequences for patient safety.
- **Banking and financial systems:** Strong data integrity, enforced through disciplined normalization, is non-negotiable in systems handling financial transactions and account records.
- **E-commerce:** Both data consistency and query speed affect whether transactions complete correctly and whether the user experience remains responsive at scale.

VI. Comparison: Previous Research vs. Present Analysis

A. How Normalization Performance Was Traditionally Evaluated

Most database coursework, and a fair amount of published guidance aimed at practitioners, evaluates normalization in a rule-based rather than a measurement-based way. A schema is judged “correct” if it satisfies the formal conditions of a target normal form, no repeating groups, no partial dependencies, no transitive dependencies, and so on, and its performance consequences are then inferred from general principles rather than observed on a running system. Fewer joins are assumed to mean faster queries; less redundancy is assumed to mean more efficient storage. These assumptions are not unreasonable as starting points, but they are exactly that: assumptions, derived from how relational algebra behaves in theory rather than from how a specific query optimizer, index structure, or storage engine behaves in practice.

This traditional approach has an obvious appeal for teaching purposes, since it gives students a checklist they can apply mechanically. Its weakness only becomes visible once a schema is actually deployed: two databases that both satisfy 3NF can perform very differently under load depending on indexing strategy, data distribution, and query pattern, none of which the formal normal-form definitions account for.

B. The Shift Toward Empirical Evaluation

The studies reviewed in this report represent a departure from that rule-checking tradition. Nagesh implemented the same dataset across four structural variants and measured execution time, storage, and CPU load directly rather than inferring them [1]. Taipalus went further, tracking database size, throughput, and even energy consumption at each stage of normalization on a real dataset running on PostgreSQL [2]. Fotache, Cluci, Taipalus, and Talaba tested decision-support workloads specifically, across three separate

commercial database servers, to see whether the assumptions built into the traditional view actually held for analytical queries [3]. In each case, the evaluation criterion shifted from “does this schema satisfy the rules of the normal form” to “what measurably happens when this schema is queried and updated,” which is a fundamentally different, and considerably more demanding, standard of evidence.

C. What the Present Comparison Adds

The present report continues this empirical shift, but its contribution is not another individual experiment; it is a synthesis of the three performance studies into a single comparative view organized around the dimensions most relevant to database design decisions: redundancy, integrity, storage, table and join count, modification efficiency, query complexity, and maintainability. Reading these studies side by side surfaces something none of them states individually: their findings are not uniformly consistent with one another, and that inconsistency is itself informative rather than a gap to be smoothed over.

Nagesh's and Taipalus's results agree that early-stage normalization, from 1NF up to 2NF or 3NF, delivers the clearest and most reliable storage and processing benefit [1], [2]. Taipalus's more granular measurements additionally show diminishing, and in places reversed, returns beyond that point, storage cost creeping back up with only marginal throughput gain as normalization continues toward 4NF [2]. Fotache et al.'s results complicate the picture further still, showing that higher normal forms can, under certain decision-support query patterns, outperform lower ones, a direct contradiction of the assumption that fewer joins always mean faster reads [3]. Where each individual study reports findings scoped to its own dataset and workload, comparing them side by side is what makes visible exactly where their conclusions align and where they pull in different directions, a picture no single paper provides on its own. Taken together, they support a more conditional conclusion than any one study states outright: that the appropriate normalization level is workload-dependent rather than fixed, which this report treats as an original synthesis rather than a restatement of any single source.

VII. Performance Analysis of Normal Forms

This section examines how each normal form affects specific, individually measurable aspects of database performance, drawing on the patterns reported across the reviewed literature and the comparative reading developed above. Table I provides a compact summary before the discussion that follows expands on each dimension in turn.

TABLE I. Comparative Effects of Normal Forms on Core Performance Dimensions

Normal Form	Redundancy / Integrity	Storage	Tables & Joins	Insert/Update/Delete
1NF	High redundancy; weak protection against anomalies.	Largest footprint.	Few tables; minimal joins.	Fast writes; error-prone updates.
2NF	Reduced partial-dependency redundancy.	Noticeably smaller.	Slightly more tables.	Fewer duplicate-update errors.
3NF	Removes transitive dependencies; strong integrity.	Further reduced.	Moderate table count and joins.	Reliable, single-point updates.
BCNF	Handles overlapping candidate-key cases.	Marginal further reduction.	More tables; higher join count.	Very reliable, but more complex writes.
4NF / 5NF	Removes multi-valued / join dependencies.	Minimal additional gain.	Highest table count and join overhead.	Most reliable, but operationally heavier.

A. Redundancy and Data Integrity

Redundancy falls sharply and consistently across the early stages of normalization. An unnormalized or 1NF schema, for example an order table that repeats full customer address details on every row, stores the same fact many times over, which means every update to that fact has to be applied consistently across every duplicate or the database silently becomes internally contradictory. Moving to 2NF removes dependencies that only hold on part of a composite key, and 3NF removes dependencies between non-key attributes, both of which directly reduce the number of places a given fact can be stored. By 3NF, most of the practical integrity benefit has already been captured; BCNF and 4NF address structurally narrower cases, overlapping candidate keys and multi-valued attributes respectively, that occur in a smaller subset of real schemas, so their incremental integrity contribution, while genuine, is smaller in scope than the jump from 1NF to 3NF.

B. Storage Requirements

Storage footprint tracks redundancy closely at first. Taipalus's measurements on the IMDb dataset showed a roughly 10 percent reduction in database size moving from 1NF to 2NF, a substantial, easily justified saving for a single decomposition step [2]. Beyond that point, however, the storage curve flattens: continuing from 2NF through to 4NF added a small amount of storage back, on the order of single-digit percentage points, because the additional tables required to enforce stricter dependency rules carry their own overhead (extra primary and foreign key columns, additional indexes) that partially offsets the redundancy removed [2]. In other words, storage efficiency is not a monotonically improving function of normalization level; it improves quickly at first and then plateaus, or mildly reverses, later on.

C. Number of Relations and Join Overhead

Each normalization step that removes a dependency generally does so by splitting a table, which means the number of relations in the schema grows monotonically from 1NF through to 4NF or 5NF. This has a direct and unavoidable consequence for querying: information that used to live in one row now has to be reassembled from several tables using JOIN operations. The reviewed studies agree that join count increases with normalization level [1]-

[3], but they diverge on what that means for actual query execution time. Nagesh's results follow the expected pattern of added joins translating into added query cost [1]. Fotache et al.'s decision-support results do not; in several of their tested configurations, execution time and completion rate improved despite more joins, which the authors' data suggests is connected to how effectively the specific database server's query optimizer could plan and execute the additional joins [3]. This divergence is one of the more important observations in this report: join count is a structural property of the schema, but it is not, by itself, a reliable predictor of performance.

D. Insert, Update, and Delete Efficiency

Write-side operations benefit from normalization in a fairly direct way: because each fact is stored in fewer places, an update touches fewer rows, and the risk of an update anomaly, where one copy of a fact is changed and another not, drops correspondingly. Insertion also becomes cleaner at higher normal forms, since a normalized schema does not force unrelated data to be supplied together just because it happens to share a table. The trade-off on the write side is more structural than performance-related: inserting a new record into a heavily normalized schema may require writes across several related tables rather than one, which adds a small amount of transactional overhead even as it removes the anomaly risk that motivated the decomposition in the first place.

E. Query Complexity and Execution Performance

Query complexity rises with normalization level in a fairly direct sense: more tables generally mean more joins, more conditions, and longer, harder-to-read SQL for anything beyond a single-table lookup. Execution performance, however, is where the reviewed studies show the least agreement, precisely because it depends on factors outside the schema itself: how well the query optimizer chooses a join order, whether appropriate indexes exist on the join columns, and how large each table actually is. Taipalus's finding that throughput essentially plateaued between 2NF and 4NF, despite the join count continuing to rise, suggests that a competent optimizer can absorb a meaningful amount of additional join complexity without a proportional performance cost, at least up to a point [2]. Fotache et al.'s results go further, showing performance

actually improving at higher normal forms in some decision-support configurations, which indicates that under certain workloads a well-indexed, normalized schema can outperform a wider, more redundant one even though it requires more joins to query [3].

F. Maintainability

Maintainability is the least directly measurable of these dimensions in the reviewed literature, but it follows reasonably from the structural properties already discussed. A normalized schema is generally easier to reason about and modify over time, because each table has a clear, single responsibility and each fact has exactly one place it can be changed. A denormalized or lightly normalized schema, by contrast, can be faster to query initially but becomes progressively harder to maintain as the application grows, since developers must remember every location, a given fact is duplicated in order to keep the database consistent. This is a long-term cost that does not show up in a single benchmark run but accumulates as a system is extended and modified by developers who were not part of its original design.

VIII. FINDINGS

Several conclusions follow from comparing the reviewed literature against the analysis developed in this report, distinguished here from what the individual studies themselves report.

- **Higher normalization does not uniformly mean better performance:** The reviewed literature is consistent on this point; normalization's benefits are front-loaded, and pushing toward BCNF or 4NF trades a shrinking integrity gain for a growing join cost in most transactional contexts [2].
- **Join count scales with normalization level, but not linearly with performance cost:** Fotache et al.'s finding that higher normal forms sometimes outperformed lower ones on decision-support queries indicates that join count alone is not a reliable predictor of execution time; query-optimizer behavior and index strategy matter as much as schema structure [3].
- **Storage efficiency and modification reliability improve together up to a point, then decouple:** Early normalization stages reduce both storage and update-anomaly risk simultaneously [1]; beyond 3NF/BCNF, the present comparison observes that storage gains taper off while modification reliability continues to improve marginally, meaning the two benefits no longer move at the same rate.
- **Workload type is the strongest determinant of the "right" normal form:** Transactional workloads with frequent, targeted writes tend to benefit from normalization sooner, while analytical or decision-support workloads with complex, large-scope reads show more mixed results, as reflected in the divergence between the transactional-style findings of Nagesh and Taipalus and the decision-support findings of Fotache et al. [1]-[3].

- **Query complexity and execution performance do not move together:** SQL complexity increases steadily with normalization level, but the present comparison's review of execution-time results shows that a capable query optimizer can decouple that structural complexity from actual runtime cost, particularly on well-indexed schemas.
- **An inconsistency worth flagging:** The reviewed studies do not agree on whether higher normal forms help or hurt read-heavy workloads, and this report treats that disagreement itself as evidence that normalization's performance effect is genuinely context-dependent rather than something a single rule of thumb can settle.

IX. RECOMMENDATIONS

Based on the comparison and findings above, the following practical guidance is proposed for database designers choosing a normalization strategy, organized by the type of system being designed.

A. General Design Baseline

- **Normalize to 3NF as a default baseline:** The reviewed evidence most strongly and consistently supports 1NF through 3NF as the range where redundancy reduction and integrity gains clearly outweigh the added join cost, making 3NF a reasonable default target for most transactional systems.
- **Reserve BCNF for schemas with genuine overlapping-key issues:** Given the diminishing storage and throughput returns observed beyond 3NF [2], BCNF is best applied selectively, where a table's candidate-key structure actually produces the anomalies BCNF is designed to catch, rather than pursued as a blanket upgrade.
- **Apply 4NF and 5NF only where multi-valued or join dependencies are actually present:** These forms address specific, relatively uncommon dependency patterns; applying them without such patterns adds join overhead without a corresponding integrity benefit.

B. Workload-Specific Guidance

- **Write-heavy transactional systems (OLTP):** Prioritize normalization through 3NF or BCNF, since these systems benefit most directly from reduced update-anomaly risk and are typically less sensitive to the added join cost, given that individual transactions usually touch a small, well-indexed subset of tables.
- **Read-heavy analytical or decision-support systems:** Treat normalization level as an open design question rather than a default choice; the evidence reviewed here shows higher normal forms can perform competitively, or even better, on such workloads provided the schema is well indexed and the query optimizer is capable [3].
- **Hybrid systems handling both transactional and analytical queries:** Consider a mixed approach, a normalized core for the transactional path paired with selectively denormalized reporting tables or materialized views for the analytical path, rather than forcing a single normalization level to serve both purposes.

C. Process and Future Direction

- **Consider controlled denormalization for read-heavy or analytical workloads:** Where decision-support or reporting queries dominate, selectively reintroducing redundancy through summary tables, materialized views, or cached aggregates can offset the join overhead of a fully normalized schema without abandoning normalization for the system's transactional core.
- **Base the decision on workload profiling, not on normalization level alone:** Query pattern, transaction frequency, dataset scale, and the specific database engine's optimizer behaviour should inform the target normal form; the comparison in this report indicates that no single normal form is optimal across all workload types.
- **Validate schema decisions with representative benchmarks rather than formal compliance alone:** Since the reviewed studies show that formal normal-form compliance does not reliably predict execution performance, designers should test candidate schemas against realistic query samples before committing to a final structure, rather than relying on rule-checklists as a substitute for measurement.
- **Future evaluation should test realistic, mixed workloads:** Much of the existing empirical work, including the studies reviewed here, tests either transactional or analytical workloads in relative isolation; systems that combine both, which describes most production databases, would benefit from normalization studies designed around mixed, realistic query patterns rather than single-workload benchmarks.

X. CONCLUSION

Normalization remains a sound default for relational schema design, but this report's synthesis of recent empirical work shows that its performance return is front-loaded and workload-dependent rather than uniform [1]-[3]. Early normalization, up to roughly 3NF, reliably reduces storage and protects against update anomalies with limited query cost, while further normalization toward BCNF and 4NF trades a shrinking integrity gain for a join overhead whose actual performance impact depends heavily on the query optimizer, indexing strategy, and workload type rather than on join count alone. The practical implication for designers is to treat 3NF as a starting point, apply stricter normal forms only where their specific anomaly patterns genuinely occur, and validate schema decisions against representative, measured workloads instead of relying on formal compliance checklists. Extending this line of empirical evaluation to mixed transactional-analytical workloads is a natural next step for future work in this area.

REFERENCES

- [1] S. Nagesh, "An Experimental Study on the Impact of Database Normalization on Query Performance and Storage Efficiency in Relational Database Systems," *Journal of Emerging Technologies and Innovative Research (JETIR)*, vol. 12, no. 3, Mar. 2025. [Online]. Available: <https://www.jetir.org/papers/JETIR2503A05.pdf>

- [2] T. Taipalus, "On the Effects of Logical Database Design on Database Size, Query Complexity, Query Performance, and Energy Consumption," arXiv:2501.07449 [cs.DB], 2025. [Online]. Available: <https://arxiv.org/abs/2501.07449>
- [3] M. Fotache, M.-I. Cluci, T. Taipalus, and G. Talaba, "The Effects of Database Normalization on Decision Support System Performance," *Information Systems*, vol. 136, art. no. 102636, 2026, doi: 10.1016/j.is.2025.102636. [Online]. Available: <https://doi.org/10.1016/j.is.2025.102636>
- [4] R. Kohita, "Exploring Database Normalization Effects on SQL Generation," in *Proc. 34th ACM Int. Conf. Information and Knowledge Management (CIKM '25)*, New York, NY, USA, Nov. 2025, pp. 5788–5796, doi: 10.1145/3746252.3761583. [Online]. Available: <https://arxiv.org/abs/2510.01989>
- [5] M. G. Sobol, A. Kagan, and H. Shimura, "Performance Criteria for Relational Databases in Different Normal Forms," *Journal of Systems and Software*, 1996. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/0164121295000623>
- [6] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970, doi: 10.1145/362384.362685.
- [7] W. Kent, "A Simple Guide to Five Normal Forms in Relational Database Theory," *Communications of the ACM*, vol. 26, no. 2, pp. 120–125, Feb. 1983, doi: 10.1145/358024.358054.
- [8] R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Databases," *ACM Transactions on Database Systems*, vol. 2, no. 3, pp. 262–278, Sep. 1977, doi: 10.1145/320557.320571.
- [9] C. J. Date, *Database Design and Relational Theory: Normal Forms and All That Jazz*. Sebastopol, CA, USA: O'Reilly Media, 2012.
- [10] T. Taipalus, "Database Management System Performance Comparisons: A Systematic Literature Review," *Journal of Systems and Software*, vol. 208, art. no. 111872, Feb. 2024, doi: 10.1016/j.jss.2023.111872. [Online]. Available: <https://doi.org/10.1016/j.jss.2023.111872>