ChameleonDB: A Schema-Driven Database Platform with Explicit Governance

Daniel Isaias Peralta*

[1]Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, Buenos Aires, Argentina
[2]Departamento de Ingeniería en Sistemas de Información, Buenos Aires, Argentina

CORRESPONDING AUTHOR: Daniel I. Peralta (e-mail: dperalta@chameleondb.dev).

Author and founder of the ChameleonDB project.

ABSTRACT Modern database systems enforce strong guarantees over data integrity and transactional correctness, yet largely treat schema evolution as an auxiliary concern managed through external processes. Research has shown that as software systems grow in complexity and longevity, this separation leads to implicit assumptions about schema identity, authority, and integrity that are not visible or enforceable at the database level.

Despite the high impact of structural changes, most database engines execute schema and data operations within the same conceptual domain, offering limited mechanisms to distinguish routine access from high-risk evolution. Prior work has identified that this lack of explicit schema governance complicates auditing, reproducibility, and long-term maintainability, particularly in systems where structural correctness is critical.

Objective: This paper presents and formalizes a schema-governed database model that addresses these limitations by treating schemas as first-class, immutable artifacts with explicit integrity guarantees. Specifically, this work introduces: (1) a model for versioned, cryptographically-verified schema storage; (2) an integrity-mode–based governance framework inspired by operating system protection mechanisms; and (3) a set of architectural principles aimed at strengthening schema evolution guarantees and auditability.

This paper introduces **ChameleonDB**, a proof-of-concept implementation of this model. ChameleonDB centers around a versioned Schema Vault that provides stable schema identity and controlled evolution, binding all operations to explicit structural contracts. Governance is enforced through integrity-aware execution contexts and role semantics inspired by operating system security models. This work focuses on architectural design and trade-offs, and should be considered an early-stage exploration rather than a finalized database system.

INDEX TERMS Database schema, schema governance, data integrity, software evolution, database design, schema immutability.

I. INTRODUCTION

Modern software systems are increasingly characterized by long operational lifetimes, continuous evolution, and the need to preserve correctness across years or even decades of change. In such systems, the database schema is not a static artifact but a living structure, subject to frequent modification as application requirements evolve. Research on schema evolution has demonstrated that most database management systems continue to treat this evolution as an auxiliary or external concern, handled through ad-hoc migrations, manual coordination, or tooling layered outside the core database engine. [1], [2]

This approach introduces what we term a *governance gap*. While data correctness is rigorously enforced at runtime through constraints, transactions, and isolation guarantees, schema correctness is often governed implicitly, relying on conventions, documentation, or operational discipline. Schema changes are executed under assumptions about authority, intent, and risk that are rarely made explicit to the system itself. As Curino et al. [1] observe, the database consequently lacks awareness of the integrity posture under which structural modifications occur.

Studies of long-lived systems have shown that schema evolution operations—such as adding constraints, altering

tables, or redefining relationships—can carry significantly higher risk than ordinary data manipulation. [2], [3] Yet, in most systems, both categories of operations are executed within the same execution domain, with no formal distinction between routine data access and high-impact structural changes. This indistinction, as argued by Fowler, [3] limits the system's ability to reason about correctness, enforce governance policies, or provide guarantees about long-term structural integrity.

The problem becomes more pronounced in long-lived systems, where schema definitions accumulate historical meaning. Over time, schemas cease to be mere technical descriptions and instead embody architectural decisions, domain assumptions, and operational contracts. When these schemas are mutated without explicit identity or contextual awareness, it becomes increasingly difficult to reason about the validity of past assumptions, the safety of future changes, or the provenance of structural decisions.

Research Objective: This paper addresses the following research question: *Can schema governance be formalized as a runtime concern within database systems, providing explicit mechanisms for schema identity, integrity verification, and context-aware evolution?*

To address this question, we introduce ChameleonDB, a schema-driven database platform designed around three core principles: (1) schemas as immutable, cryptographically-identifiable artifacts; (2) integrity-mode–based governance inspired by operating system protection rings; [5], [7] and (3) runtime enforcement of structural contracts. This work contributes a conceptual and architectural model, together with a proof-of-concept implementation, demonstrating that schema governance can be made explicit and enforceable by design.

A. Contributions

This paper makes the following contributions:

- A formal model for treating schemas as first-class, immutable artifacts with cryptographic identity.
- An integrity-mode framework adapting operating system protection concepts to database schema governance.
- A proof-of-concept implementation (ChameleonDB) demonstrating feasibility and design trade-offs.
- An analysis of use cases where explicit schema governance provides value over conventional approaches.

II. PROBLEM STATEMENT AND MOTIVATION

A. Lack of Explicit Schema Identity

Current database systems provide strong guarantees for data integrity but lack equivalent mechanisms for governing schema integrity. While constraints, transactions, and isolation levels define clear rules for data manipulation, schema evolution is typically managed through external migration frameworks or administrative conventions. [1], [2] These approaches assume that operators understand the implications of structural changes and apply them correctly, but research has shown that they do not allow the database to enforce or even observe these assumptions. [3]

A central issue, identified by Roddick, [2] lies in the absence of explicit schema identity. In most systems, a schema is treated as a mutable object whose definition can be overwritten or incrementally altered. Once a change is applied, the previous structural state effectively disappears, leaving no intrinsic trace of its identity or intent. This erasure, as Curino et al. demonstrate, [1] complicates auditing, rollback, and long-term reasoning, particularly in systems where historical correctness matters.

B. Implicit Authority in Schema Operations

Closely related is the lack of explicit authority and integrity modeling for schema operations. Structural modifications often require elevated privileges and carry system-wide consequences, yet they are executed using the same conceptual mechanisms as ordinary queries. Research in access control systems suggests that the database does not distinguish whether an operation is performed under conservative, experimental, or emergency assumptions, nor can it enforce different integrity expectations based on context. [6] As a result, high-risk operations are indistinguishable from routine ones at the system level.

C. Consequences for Long-Lived Systems

This indistinction introduces several practical risks, as documented in empirical studies of database evolution. [3] Schema changes can silently invalidate application assumptions, break invariants that are not formally encoded, or introduce inconsistencies that are difficult to detect after the fact. Over time, these risks compound, leading to brittle systems where evolution becomes increasingly costly and error-prone. The absence of built-in schema governance also limits the ability to audit decisions, reproduce historical states, or reason about why certain structural choices were made.

D. Motivation for a Schema-Governed Model

The motivation for this work arises from the gap between data-level rigor and schema-level informality identified in prior research. [1]–[3] By treating schema evolution as an explicitly governed process—anchored in stable schema identity and executed under well-defined integrity assumptions—we hypothesize that database systems can regain visibility and control over their own structural correctness. This shift may enable not only safer evolution but also a more principled foundation for long-term system design, auditability, and operational predictability.

III. CHAMELEONDB OVERVIEW

A. Design Goals

Building on prior work in schema evolution [1], [2] and operating system security, [5], [7] ChameleonDB is designed

around a central premise: **schemas should be treated as authoritative, governed, and operational entities**. All other components of the system—data storage, query execution, and access control—are structured to support this premise.

The primary design goals of ChameleonDB, informed by gaps identified in existing systems, are:

- **Schema centrality**: schemas define structure, validation boundaries, and governance constraints.
- **Explicitness over convention**: all structural assumptions and permissions must be declared and enforced, following principles outlined by Saltzer and Schroeder. [5]
- **Controlled evolution**: schema changes are intentional, versioned, and observable, addressing issues raised by Curino et al. [1]
- **Operational integrity**: system behavior depends on an explicit governance posture rather than implicit privileges, adapting concepts from role-based access control. [6]

These goals position ChameleonDB as a schema-driven platform with explicit operational governance, distinguishing it from migration-centric approaches that rely primarily on external tooling.

B. Schema-Driven Platform Model

ChameleonDB operates as a **schema-driven platform**, where schemas function as formal contracts rather than auxiliary metadata. This model draws on principles from type systems and contract-based programming, adapted to database contexts. Figure 1 illustrates this model by depicting the conceptual separation between schema governance and data execution layers within ChameleonDB.

In this model:

- Every dataset is explicitly bound to a schema version.
- All write and query operations are validated against the active schema.
- Schema versions are immutable once published.
- Structural evolution occurs through the introduction of new schema versions.

Under these constraints, schema drift is prevented by construction, as all operations are evaluated against known structural intent. However, this approach requires that schema changes be planned and versioned explicitly, which may introduce additional design-time overhead.

C. Governance as an Operational Mode

A defining characteristic of ChameleonDB, inspired by operating system integrity models, [5], [7] is that governance is treated as an **explicit operational mode**, not merely a configuration of permissions.

At any point in time, the system operates under a defined **Integrity Mode**, which constrains the set of permissible actions across schema lifecycle, data access, and administrative operations. This mode represents the current governance posture of the database, analogous to privilege levels in operating systems.

By making governance state explicit, ChameleonDB aims to avoid ambiguous authority and ensure that high-risk operations are only possible under clearly declared conditions (Figure 2). However, this explicitness requires operational discipline and may complicate rapid schema iteration in exploratory contexts.

D. Core Components

At a high level, ChameleonDB consists of the following core components:

- **Schema Vault**: A centralized and authoritative registry for immutable, versioned schema definitions and their lifecycle states.
- **Governance Layer**: An enforcement layer that evaluates all operations against the active Integrity Mode and role-based authorizations.
- **Execution Layer**: A schema-aware execution engine responsible for validating and executing queries and data operations.
- **Storage Layer**: A schema-agnostic persistence layer that executes only operations validated by the upper layers.

These components interact through explicit contracts, with the goal of ensuring that governance and schema validation are consistently enforced. The effectiveness of this architecture Figure 1 in practice remains subject to empirical validation.

E. Operational Scope and Limitations

ChameleonDB is designed for environments where **schema correctness, traceability, and controlled evolution** are primary concerns. It does not aim to optimize for maximal flexibility or unconstrained modification. This represents a deliberate design choice, trading iteration speed for structural guarantees.

Performance optimization, advanced query planning, and large-scale distributed execution are considered secondary objectives and are intentionally outside the scope of the current design. These limitations reflect conscious priorities rather than technical barriers, and future work may address them without compromising core governance principles.

IV. SCHEMA VAULT DESIGN

A. Role of the Schema Vault

The **Schema Vault** is the foundational component of ChameleonDB and serves as the **authoritative source of structural truth** for the entire system. It governs the definition, lifecycle, and integrity of all schemas independently of data storage and execution.

By decoupling schema definition from data instantiation, the Schema Vault aims to ensure that structural intent re-

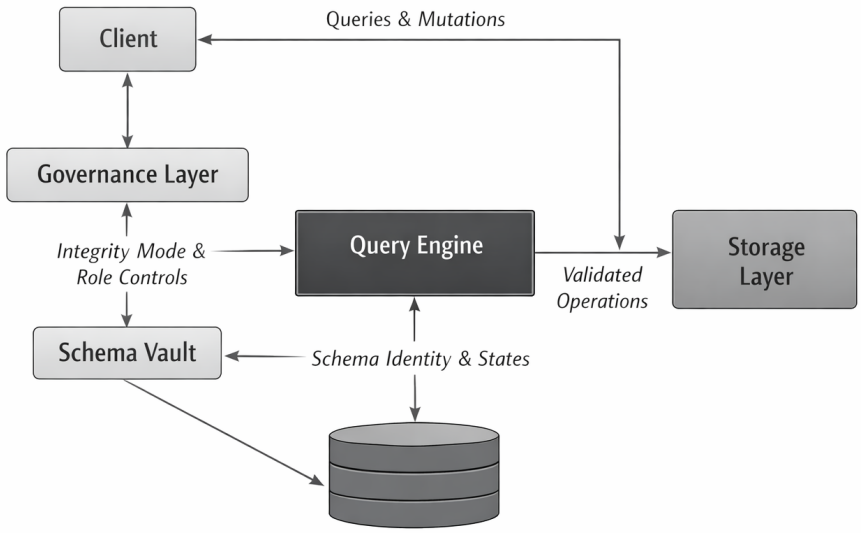

FIGURE 1. Conceptual execution layers in ChameleonDB. The system distinguishes between data operations and schema governance through explicit integrity boundaries, with schemas acting as authoritative, immutable contracts.

mains stable, verifiable, and enforceable over time. This separation is informed by principles from configuration management and version control systems, adapted to database contexts.

B. Schemas as Immutable and Identifiable Artifacts

In ChameleonDB, schemas are treated as **immutable artifacts**. Once a schema version is published, its structure, validation rules, and semantic intent cannot be altered. This approach is consistent with immutability principles observed in functional programming and version control systems. [1], [3]

Each schema version is assigned a **cryptographic identity**, derived from a deterministic hash of its canonical definition. This identity serves multiple purposes:

- It uniquely identifies the schema version.
- It enables tamper detection and integrity verification, similar to mechanisms used in distributed systems. [8]
- It allows schemas to be referenced independently of human-readable version labels.

Immutability combined with cryptographic identity is intended to ensure that schemas are both **stable and verifiable** artifacts. However, this approach requires that schema errors be corrected through new versions rather than in-place modifications, which may increase version proliferation.

C. Cryptographic Integrity Guarantees

The Schema Vault enforces integrity guarantees by validating the cryptographic identity of schemas at all interaction points. Any attempt to modify a published schema results in an integrity violation rather than a silent update.

Under this model:

- Schema definitions should not be altered retroactively.
- Execution and validation should reference the intended schema.
- Structural assumptions should be protected against accidental or malicious modification.

Integrity violations are surfaced explicitly and may trigger elevated governance responses depending on the active Integrity Mode. The effectiveness of this mechanism in preventing schema-related incidents requires empirical validation in production environments.

D. Schema Versioning and Lineage

Schema versions in the Vault form an explicit evolution graph rather than a mutable timeline, similar to version control systems like Git. [9] Each version may declare lineage relationships to prior versions, but compatibility is never assumed implicitly.

The versioning model supports:

- Parallel active schema versions.
- Explicit deprecation without forced migration.
- Historical validation of data under its original schema.

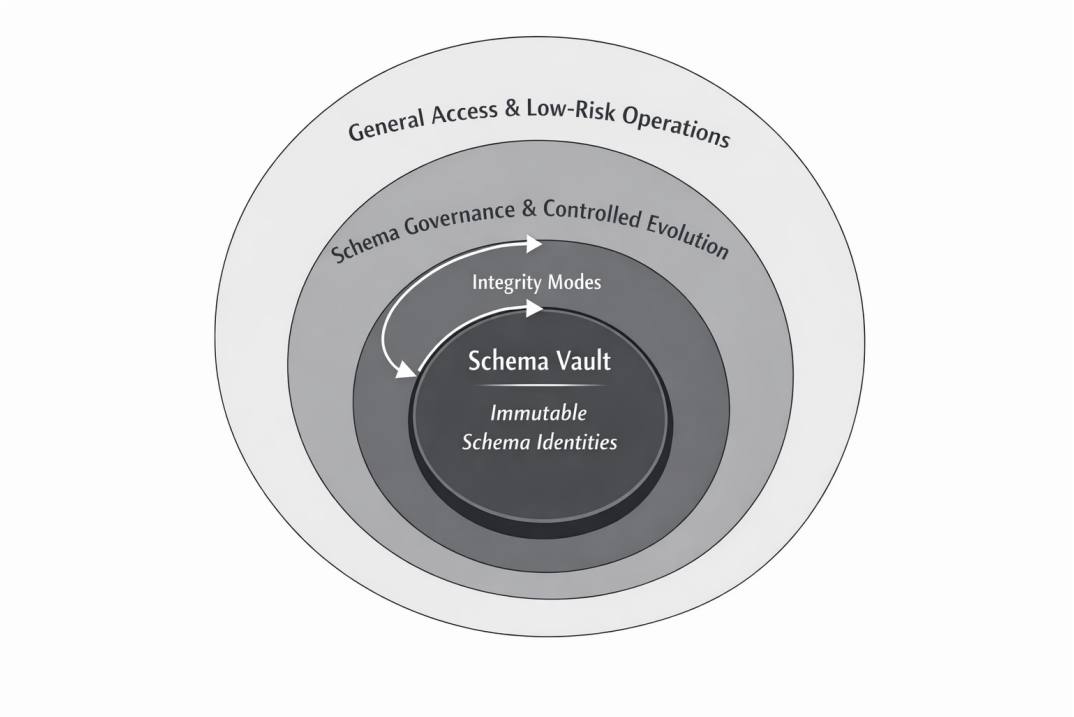

FIGURE 2. Conceptual execution layers in ChameleonDB. The system distinguishes between data operations and schema governance through explicit integrity boundaries, with schemas acting as authoritative, immutable contracts.

By separating lineage from mutability, ChameleonDB aims to enable controlled evolution without sacrificing traceability. However, managing multiple concurrent schema versions may introduce operational complexity.

E. Schema Lifecycle States

Schemas progress through a defined lifecycle managed by the Schema Vault. Typical states include:

- **Draft**: schema under development, not eligible for data binding.
- **Published**: schema available for dataset binding and validation.
- **Deprecated**: schema remains valid but is discouraged for new datasets.
- **Archived**: schema retained solely for historical reference and validation.

Lifecycle transitions are governed operations and may only be executed under appropriate Integrity Modes. This lifecycle model draws inspiration from software release management practices.

F. Binding Schemas to Data

All datasets in ChameleonDB are explicitly bound to a specific schema identity at creation time. This binding is immutable and enforced throughout the dataset's lifetime.

Under this constraint:

- Data validation should be deterministic and reproducible.
- Queries should operate within known structural boundaries.
- Historical data should remain interpretable even as schemas evolve.

This explicit binding aims to eliminate schema drift by design, though it requires careful planning during schema evolution to avoid orphaned datasets.

G. Interaction with Integrity Modes

The Schema Vault integrates directly with the Integrity Mode model. The set of permissible schema operations—such as publishing, deprecating, or archiving schemas—is constrained by the active Integrity Mode.

For example:

- Drafting schemas may be allowed in lower-risk modes.
- Publishing or deprecating schemas requires elevated modes.
- Emergency schema overrides are restricted to break-glass scenarios and fully audited.

This integration aims to ensure that schema integrity is preserved not only structurally but also operationally, though the practical balance between security and usability requires further study.

H. Design Implications

The Schema Vault introduces intentional constraints on schema modification and evolution. These constraints shift

responsibility from runtime recovery to **design-time correctness and explicit governance**.

By combining immutability, cryptographic identity, and mode-aware enforcement, the Schema Vault is designed to provide guarantees that may be difficult to achieve through migration-centric or convention-based approaches. However, these guarantees come at the cost of reduced flexibility and increased operational discipline.

V. INTEGRITY MODES AND GOVERNANCE ENFORCEMENT

A. Rethinking Governance as an Operational State

Traditional database governance models treat permissions as static configurations evaluated independently of system context. Research in access control systems suggests that in practice, however, the *risk profile* of an operation depends not only on who performs it, but also on **when**, **under what conditions**, and **with what systemic impact**. [6]

ChameleonDB reframes governance as an **operational state**, inspired by integrity and protection models commonly found in Unix-like operating systems. [5], [7] Rather than relying solely on role-based permissions, the system operates under explicit **Integrity Modes** that define the allowable scope of structural and data-level actions.

This approach aims to align governance enforcement with runtime behavior rather than static configuration, though its effectiveness in complex multi-user environments remains to be validated empirically.

B. Integrity Modes Overview

An Integrity Mode represents the current governance posture of the database. Each mode defines a bounded set of permitted operations, enforced uniformly across schema, data, and execution layers.

Conceptually, Integrity Modes follow a ring-based model inspired by operating system protection rings: [5]

- **R3 – Application Mode (Sandbox Mode)**: Normal operation mode. Schema evolution is prohibited. Only data-level operations under published schemas are allowed.
- **R2 – User Mode (Schema Evolution Mode)**: Allows controlled schema lifecycle transitions (e.g., publishing new schema versions) while preserving immutability guarantees.
- **R1 – Privileged Mode (Governance Mode)**: Enables administrative and governance-level actions such as role configuration, policy updates, and schema deprecation.
- **R0 – Kernel Mode (Break-Glass Mode)**: Reserved for exceptional recovery or forensic scenarios. Operations in this mode are highly restricted, fully audited, and explicitly marked as integrity-breaking actions.

These modes are mutually exclusive and explicitly declared as part of the execution context. The practical usability of this model in dynamic operational scenarios requires further investigation.

C. Paranoid Governance as Mode Restriction

The governance philosophy introduced in earlier sections is formalized through Integrity Modes. Rather than distributing implicit authority across roles, ChameleonDB assumes that **lower integrity modes should not gain access to higher-risk operations**. [6]

This design aims to enforce the following principles:

- Operations permitted in a higher-integrity mode are not implicitly allowed in lower modes.
- Transition between modes is itself a governed operation.
- Most runtime activity occurs in the lowest-risk mode (R3).

This approach expresses governance as **systemic distrust of silent privilege escalation**, not distrust of operators. However, this conservative model may complicate rapid iteration in development environments.

D. Roles Within Integrity Modes

Roles in ChameleonDB remain relevant but are evaluated **within the constraints of the active Integrity Mode**, following principles from mandatory access control systems. [11] A role may authorize a specific operation only if:

1) The operation is permitted by the current Integrity Mode.
2) The role explicitly grants authority for that operation.

This two-dimensional check (mode × role) is designed to prevent scenarios where role misconfiguration alone can enable high-risk actions. For example, a role capable of publishing schemas cannot perform that action while the system is operating in Application Mode (R3).

E. Mode Transitions and Explicit Intent

Transitions between Integrity Modes are explicit operations requiring elevated authorization and justification. Each transition should be:

- Authenticated and authorized.
- Logged with contextual information (actor, timestamp, justification).
- Time-bounded when possible (e.g., elevated modes automatically downgrade after a timeout).

This mechanism aims to make authority escalation visible and accountable, reducing the risk of accidental or unauthorized high-risk operations. However, the operational burden of explicit mode transitions requires evaluation in production settings.

F. Audit and Traceability

All operations in ChameleonDB, particularly those executed under elevated Integrity Modes, are logged with sufficient

detail to support forensic analysis and compliance requirements. Audit records should include:

- Active Integrity Mode
- Schema identity involved
- Actor and role
- Operation type and outcome
- Timestamp

This comprehensive logging is intended to enable post-hoc analysis and accountability, though the storage and query performance implications require further study.

G. Design Trade-offs

The Integrity Mode model deliberately prioritizes governance and auditability over operational flexibility. This design choice may introduce friction in environments requiring rapid schema iteration, but aims to provide value in contexts where structural correctness is paramount.

The practical balance between security and usability will depend on deployment context and requires empirical evaluation.

VI. EXECUTION MODEL

A. Schema-Aware Execution Context

In ChameleonDB, every operation executes within an **explicit execution context** that includes:

- The active Integrity Mode
- The bound schema identity
- The actor and role information
- Transaction and session metadata

By making this context explicit, the execution layer can validate operations not only against structural definitions but also against governance constraints. This approach is informed by capability-based security models, [10] adapted to database contexts.

B. Query Validation and Schema Binding

All queries in ChameleonDB are validated against the bound schema before execution. This validation includes:

- Structural correctness (referenced fields and entities must exist in the schema).
- Type compatibility (operations must respect field types and constraints).
- Integrity Mode conformance (operation must be permitted under current mode).

Queries that fail any validation step are rejected with explicit error messages indicating the validation stage and governing rule violated. This early rejection is intended to prevent runtime ambiguity, though it may require adjustment periods for developers accustomed to more permissive systems.

C. Data Mutation Constraints

Data mutations (inserts, updates, deletes) are subject to additional constraints beyond structural validation:

- Mutations must respect schema-defined invariants and constraints.
- Mutations in lower Integrity Modes may be restricted to certain operation types.
- Batch operations may require elevated modes if they affect multiple schema-bound datasets.

These constraints aim to prevent data-level operations from compromising structural integrity, though the granularity of these restrictions requires tuning based on operational needs.

D. Error Reporting and Contextual Feedback

When an operation fails validation or execution, ChameleonDB provides contextual error messages that should include:

- Validation stage of failure
- Integrity Mode in effect
- Schema identity involved
- Governing rule violated

For operations executed under elevated Integrity Modes, additional integrity signals are emitted for audit and forensic analysis. This approach is intended to reduce ambiguity and improve debuggability in complex systems.

E. Architectural Trade-offs

The execution model introduces additional validation overhead compared to traditional databases. These costs are accepted intentionally to provide:

- Explicit structural guarantees
- Runtime-enforced governance
- Strong auditability and traceability

ChameleonDB prioritizes correctness and predictability over maximal execution throughput. The performance implications of this design choice require quantitative evaluation in realistic workloads.

VII. USE CASES & DESIGN TRADE-OFFS

A. Schema Evolution in Long-Lived Systems

ChameleonDB may be particularly suited for systems with long-lived data and evolving domain models, where schema changes are inevitable but structural correctness must be preserved. Research on schema evolution has identified such systems as particularly challenging for conventional migration-centric approaches. [1], [3]

In contrast to migration-centric systems that mutate schemas in place, ChameleonDB enforces evolution through the **introduction of new, immutable schema identities**. Multiple schema versions may coexist, each bound to its corresponding datasets and governed by explicit lifecycle states.

Under favorable conditions, this model may enable:

- Controlled, incremental schema evolution.
- Historical validation of data under its original schema identity.
- Explicit reasoning about compatibility and deprecation timelines.

The trade-off lies in increased upfront governance and design effort, which may be justified in systems prioritizing long-term predictability and traceability over rapid iteration.

B. Multi-Actor Governance Environments

In environments where multiple actors interact with the same data platform, governance complexity increases significantly. ChameleonDB addresses this by enforcing **Integrity Modes** that define the operational posture of the system.

Different actors may operate under different roles, but all actions are constrained by the active Integrity Mode. This constraint aims to ensure that:

- Application-level actors remain confined to low-risk modes.
- Schema evolution and governance actions require elevated modes.
- Authority escalation is explicit, observable, and auditable.

This design may reduce the risk of accidental structural changes while enabling safe collaboration across teams, though the operational overhead requires evaluation in practice.

C. Schema as an Explicit Execution Contract

By incorporating schema identity and Integrity Mode into the execution context, ChameleonDB transforms schemas into **explicit execution contracts**. Queries and data mutations are validated not only against structural definitions but also against the governance posture of the system.

Under this model, implicit assumptions should be eliminated and runtime ambiguity across schema versions prevented. This approach may be particularly valuable in systems requiring long-term compatibility guarantees or regulatory accountability.

D. Comparison with Migration-Centric Approaches

Conceptually, ChameleonDB differs from migration-centric systems along several axes, as summarized in Table 1.

The table highlights that ChameleonDB optimizes for explicit governance and deterministic behavior rather than minimal friction. The appropriateness of this trade-off depends on system requirements and operational context.

E. Operational Predictability

Because governance posture and schema identity are explicit at runtime, system behavior should become more predictable. Operators may be able to reason about:

- Which Integrity Mode was active during an operation.
- Which schema identity governed data validation.
- Why a given operation was accepted or rejected.

This predictability may simplify debugging and reduce systemic uncertainty, though empirical validation is required to confirm these benefits.

F. Design Trade-offs Summary

ChameleonDB deliberately trades:

- **Flexibility for explicit governance**
- **Implicit convenience for deterministic execution**
- **Speed of change for integrity and auditability**

These trade-offs may make the system unsuitable for environments prioritizing unconstrained iteration, but potentially well-suited for systems where correctness and accountability are primary concerns.

VIII. RELATED WORK

A. Schema Evolution Research

Curino et al. [1] provide comprehensive analysis of schema evolution challenges in database applications, introducing the PRISM framework for managing schema evolution through explicit versioning. Our work extends these concepts by integrating versioning directly into the database runtime rather than treating it as an external process.

Roddick's survey [2] identifies fundamental issues in schema versioning systems, including the lack of schema identity and temporal management. ChameleonDB addresses several of these issues through cryptographic identity and immutable versioning, though empirical validation of effectiveness is needed.

B. Database Governance and Security

Ferraiolo and Kuhn's work on role-based access control [6] provides foundations for our approach to role evaluation within Integrity Modes. However, we extend their model by introducing operational modes that transcend traditional role boundaries.

Saltzer and Schroeder's protection principles [5] inform our design of Integrity Modes, adapting operating system security concepts to database contexts. Lampson's protection rings [7] provide direct inspiration for our ring-based governance model.

C. Immutability in Data Systems

Fowler's work on evolutionary database design [3] advocates for controlled, incremental evolution. ChameleonDB operationalizes these principles through explicit versioning and governance enforcement.

D. Distinction from Prior Work

While prior research has identified problems in schema evolution and proposed versioning approaches, ChameleonDB

TABLE 1. Comparison: Migration-Centric Systems vs. ChameleonDB

Aspect	Migration-Centric Systems	ChameleonDB
Schema mutability	Mutable	Immutable
Evolution model	Procedural	Declarative & identity-based
Validation timing	Deployment-time	Runtime & operation-level
Governance	Role-based, coarse	Integrity Mode–driven
Traceability	Partial or external	Built-in and explicit

contributes a complete system design integrating: (1) cryptographic schema identity, (2) runtime governance enforcement, and (3) operating system–inspired protection mechanisms. This integration represents, to our knowledge, a novel approach to schema governance in database systems.

IX. DISCUSSION & FUTURE WORK

A. Design Scope and Current Limitations

ChameleonDB intentionally focuses on schema governance, integrity enforcement, and controlled evolution. Performance optimization, distributed execution, and advanced query planning are currently secondary concerns.

These limitations reflect conscious design choices rather than incomplete implementation and may be revisited without compromising the core integrity model. However, production deployment will require addressing performance considerations not fully explored in this work.

B. Evaluation and Metrics

This work presents a conceptual and architectural model rather than an empirical performance evaluation. Future work should include:

- Measuring reduction of schema-related incidents through explicit Integrity Modes.
- Assessing operational overhead introduced by runtime governance enforcement.
- Evaluating auditability improvements enabled by schema identity binding.
- Comparative studies with migration-centric approaches in realistic workloads.

Such studies would provide empirical grounding for the design choices presented and validate (or refute) the claimed benefits of this approach.

C. Extensibility of the Integrity Mode Model

While ChameleonDB provides a concrete implementation, the Integrity Mode model is not inherently database-specific. Similar approaches could potentially be applied to other systems requiring explicit governance over structural operations, such as configuration management systems or infrastructure-as-code platforms.

This suggests that ChameleonDB may serve as a **reference implementation** of a broader integrity-driven execution model, though generalizability requires further investigation.

D. Open Questions

Several questions remain for future research:

- What is the optimal granularity for Integrity Mode restrictions?
- How do users adapt to explicit mode transitions in practice?
- What performance overhead is acceptable for integrity guarantees?
- Can the model be extended to distributed database systems?

E. Invitation to Exploration and Adoption

ChameleonDB is presented as an **explorable architectural approach** rather than a prescriptive solution. Practitioners and researchers are encouraged to evaluate, adopt, and extend the schema-driven, integrity-mode–governed model in environments where structural correctness and accountability are critical.

Feedback, critique, and alternative implementations are considered essential for refining both the model and its applicability.

F. Concluding Remarks

By combining immutable schema identities, explicit Integrity Modes, and runtime-enforced governance, ChameleonDB demonstrates that structural intent can be made explicit and enforceable by design.

This work argues that schema governance should be considered as a runtime concern within database systems, not merely an external process. We have presented a formal model and proof-of-concept implementation, though empirical validation of benefits and usability remains essential future work.

The approach presented here involves deliberate trade-offs between flexibility and governance. Whether these trade-offs are justified depends on deployment context, system requirements, and organizational priorities. We hope this work contributes to broader discussion on explicit governance mechanisms in database systems.

ACKNOWLEDGMENT

This work was developed as part of an independent research and engineering effort. The author thanks early reviewers for conceptual feedback and discussions, and acknowledges the influence of prior research on schema evolution and operating system security that informed this design.

REFERENCES

[1] C. Curino, H. J. Moon, and C. Zaniolo, "Managing Schema Evolution in Database Applications," *The VLDB Journal*, vol. 19, no. 4, pp. 473–493, Aug. 2010.

[2] J. F. Roddick, "A Survey of Schema Versioning Issues for Database Systems," *Information and Software Technology*, vol. 37, no. 7, pp. 383–393, 1995.

[3] M. Fowler, *Evolutionary Database Design*. Boston, MA, USA: Addison-Wesley, 2004.

[4] M. Stonebraker, "The End of an Architectural Era (It's Time for a Complete Rewrite)," in *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, 2007, pp. 1150–1160.

[5] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, Sept. 1975.

[6] D. F. Ferraiolo and D. R. Kuhn, "Role-Based Access Controls," in *Proceedings of the 15th National Computer Security Conference*, 1992, pp. 554–563.

[7] B. W. Lampson, "Protection," *ACM Operating Systems Review*, vol. 8, no. 1, pp. 18–24, Jan. 1974.

[8] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology — CRYPTO '87 Proceedings*, 1988, pp. 369–378.

[9] S. Chacon and B. Straub, *Pro Git*, 2nd ed. New York, NY, USA: Apress, 2014.

[10] J. B. Dennis and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," *Communications of the ACM*, vol. 9, no. 3, pp. 143–155, Mar. 1966.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.