Qolqa: A Metrics-Driven Design Tool for Document-Oriented NoSQL Databases

Anghelo Alagon-Fernandez[1], **Yerson Joab Chirinos-Vilca**[1], **Maristela Holanda**[2], **Harley Vera-Olivera**[1]

[1]Department of Computer Science
Universidad Nacional de San Antonio Abad del Cusco (UNSAAC)
Cusco, Perú

[2]Department of Computer Science – University of Brasília (UnB)
Brasília, Brasil

{200518,182731,harley.vera}@unsaac.edu.pe, mholanda@unb.br

Abstract. *Document-oriented NoSQL databases, although considered schema-less, require design tools to understand the structural context of schemas, including document organization, attributes, and nesting levels. However, current tools present limitations as they rely heavily on the designer's experience, lack metrics to evaluate schema quality, and do not integrate queries as part of the design process. Additionally, many of these solutions are commercial or have restricted free versions, which hinders the development of optimal solutions. In this context, this work proposes Qolqa, a web-based tool for designing schemas in document-oriented NoSQL databases that incorporates evaluation metrics, integrates queries as a relevant component of the design, and supports multiple schemas within a single solution. The proposed approach aims to facilitate decision-making in schema design and improve the quality of solutions, regardless of the designer's level of experience.*

1. Introduction

Document-oriented NoSQL databases are considered schema-less [Mozaffari et al. 2024]; however, design tools are necessary to understand the structural context of their schemas. This includes the organization of documents, the identification of simple and complex attributes, as well as relationships defined through references or nesting at different levels [Luiz 2025][Vera et al. 2015]. These aspects are not adequately visualized in JSON format. Although these tools allow schemas to be represented using graphical notations, their effectiveness largely depends on the designer's expertise and does not provide objective criteria for assessing design quality [Vera-Olivera and Holanda 2024]. Despite the availability of various design tools [DbSchema 2025] [Datensen 2025] [Hackolade 2025] [Luiz 2025], many of them are either commercial in nature or offer limited functionality in their free versions. Furthermore, these tools do not explicitly incorporate queries as part of the design process, nor do they consider metrics that guide the designer in evaluating schemas, which hinders the identification of an optimal solution.

The objective of this work is to develop Qolqa[1,2] a web-based schema design tool

[1]https://github.com/QOLQA/frontend-qolqa
[2]https://youtu.be/gD1SmnvSrcI

for document-oriented NoSQL databases that incorporates metrics for schema evaluation. In addition, the study aims to integrate queries as a core component of the design process and to consider solutions involving multiple schemas, enabling a comprehensive evaluation and optimization of the proposed designs.

Data schema design is one of the critical stages in software development. Although it constitutes only a single phase, its impact on the final solution is often greater than that of any other stage. Therefore, identifying the most appropriate schema for a given problem is essential. However, this is not a trivial task, highlighting the need for a metrics-driven tool that enables designers, regardless of their level of expertise, to derive the most suitable schemas for their solutions.

To address this need and present our proposed solution, the remainder of this paper is organized as follows. Section 2 reviews the related work concerning data schema design tools in the NoSQL domain. Section 3 describes the overall architecture of the Qolqa platform, detailing both its frontend and backend components. Section 4 presents a demonstration of the tool through its main modules and evaluation features. Finally, Section 5 outlines the conclusions derived from this study along with proposed directions for future work.

2. Related Work

DbSchema [DbSchema 2025] is a desktop-based application without a web version. As its architecture was originally designed for the relational model, the tool does not adequately support the representation of multi-level nesting or complex attributes.

Moon Modeler [Datensen 2025] is a desktop tool with no web-based version, specialized in NoSQL databases such as MongoDB. It provides native support for multi-level nesting, complex attributes, and relationships through diagrammatic representations and "containment lines". Unlike tools grounded in the relational model, it is optimized for document-oriented databases.

Hackolade [Hackolade 2025] is a solution available in both web and desktop versions. Initially designed for document-oriented databases, it has evolved to support a broad range of NoSQL technologies. It stands out for its visual modeling capabilities, enabling the structured representation of multi-level nesting and complex attributes.

brModeloWeb (NoSQL module) [Luiz 2025] is a tool that implements the aggregate model for the visual representation of multi-level nesting and complex attributes. Although brModeloWeb is a browser-based platform, this specific functionality for NoSQL databases is not yet available in the public web version. Its academic-oriented approach facilitates logical design across document, column-family, and key-value models.

3. Architecture

The overall architecture of the Qolqa platform, illustrated in Figure 1, adopts the *Feature-Sliced Design* (FSD) [Feature-Sliced Design 2018] pattern on the frontend, while the backend is grounded in the principles of *Clean Architecture* [Martin 2017] and supported by a data persistence layer designed to support high scalability. The linguistic refinement for text correction and clarity purposes were supported by the ChatGPT generative AI tool.

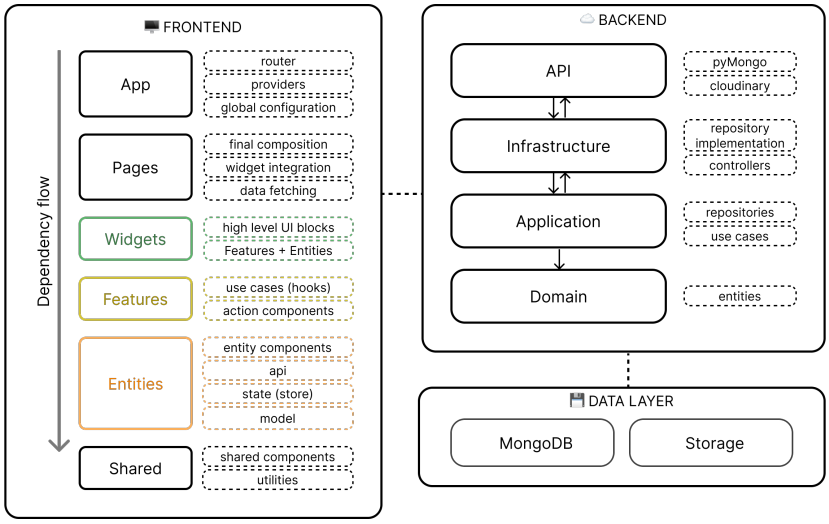

Figura 1. Overall architecture of the Qolqa platform

3.1. Frontend

The frontend adopts the Feature-Sliced Design framework to partition the application into specialized layers, enhancing modularity and maintenance. In this setup, global routing belongs to the `app` layer, application views to the `pages` layer, and reusable utilities to the `shared` layer. Furthermore, the operational core is split between the `Widgets` layer for rendering visual blocks, the `Features` layer for user use cases, and the `Entities` layer for domain models, state management, and backend communication. This organization guarantees a strict separation of concerns, successfully streamlining code reuse across the entire system.

3.1.1. System Query Storage and Management

Input is performed through an interactive interface where requirements are written as natural language phrases. After capturing the text, a prefix-matching algorithm analyzes the first three characters of each word to automatically link the corresponding collections in the diagram, allowing manual adjustments before final validation. Finally, data is secured through a hybrid strategy that combines remote server persistence with a local browser backup managed by Zustand, ensuring information availability in real time.

3.1.2. Real-Time Implementation of Evaluation Metrics

The proposed system automatically executes four essential metrics to validate the quality of the schemas designed within the graphical interface. The first of these is the Access Pattern, which evaluates navigation complexity by analyzing the hierarchical depth of the nested tables along with the connectivity level of the most central node, which is directly expressed through the equation $accessPattern = (maxDepth \times 0.4) + (maxRelations \times 0.6)$.

The second metric corresponds to the Recovery Cost, an indicator that estimates the effort required to extract information by recursively summing simple atomic attributes and complex elements together with the access pattern. Linear arrays are excluded because they represent structures already embedded within the container document,

formalizing the indicator through the relationship $recoveryCost = (totalAttributes \times 0.51) + (totalNestedTables \times 0.49) + accessPattern$.

For its part, Redundancy measures entity duplication by collecting all textual identifiers of the tables into a flat array. The calculation of the surplus of repetitions is performed directly using the formula $redundancy = \sum(count - 1)$ for each set of names whose frequency is strictly greater than one.

Completeness represents the capacity of the model to provide effective support for user requirements by verifying that each collection and its connection paths exist within the designed schema. By constructing a bidirectional adjacency graph and using the BFS search algorithm to validate connectivity, the system defines this percentage with the formula $completude = (handledQueries/totalQueries) \times 100$.

Schema evaluation is performed continuously within the interface through a strategy based on caching and result memoization. The system generates a dependency hash code for nodes and edges that excludes the spatial coordinates of the canvas, which prevents unnecessary recalculations when the user repositions elements. Consequently, the algorithms operate exclusively upon actual structural mutations of the model, achieving response times in the order of milliseconds.

3.1.3. Versioning and Comparison

Versions allow the exploration of different alternatives for the same model without affecting the main version. The system allows creating an empty version to start a new modeling process from scratch or generating a new version from the current model. In both cases, the information for each version is stored independently in the database, facilitating its management and subsequent analysis.

To perform the comparison, the system retrieves all stored versions from the database, calculates the corresponding metrics for each one, and presents the results comparatively using a bar chart and a table. This visualization allows identifying the version that best fits the problem being modeled.

3.2. Backend and Database

As shown in the right-hand block of Figure 1, the backend is implemented as a REST API using `FastAPI`, following the principles of *Clean Architecture*. The system is structured into isolated layers (*API*, *application*, *domain*, and *infrastructure*) to ensure strict functional decoupling. During the execution flow, controllers in the outer layer intercept client requests, validate data through DTOs supported by `Pydantic`, and delegate orchestration to use cases. These interact exclusively with pure domain entities and forward operations to repository implementations in the infrastructure layer, integrating `MongoDB` as the primary engine to natively persist solutions, versions, and queries. This architectural arrangement, reinforced with security mechanisms such as stateless authentication via `JSON Web Tokens (JWT)` and rate limiting (*rate limiting*), ensures that business logic remains immutable and independent of technical details, thereby optimizing both platform maintainability and resource protection.

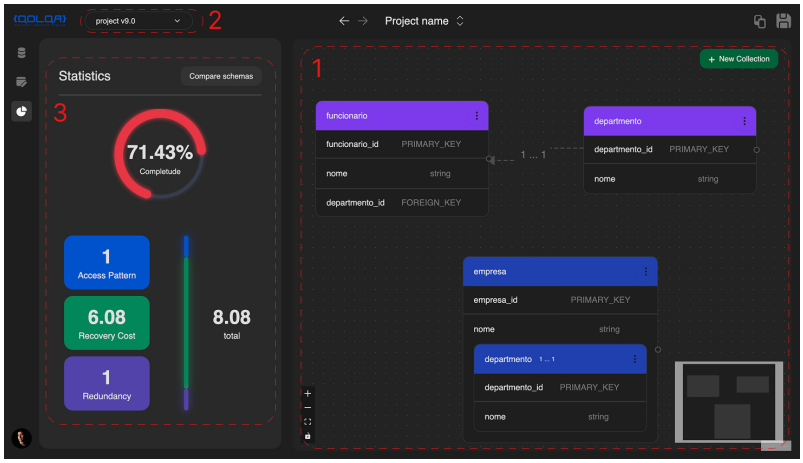

Figura 2. Example of the modeling solution page

4. Tool Demonstration

The tool provides a set of modeling and analysis functionalities, as shown in Figure 2 and Figure 3, including:

1. A **Design** module for defining queries, collections, and their respective relationships, allowing the configuration of complex attributes, including nested objects, as detailed in Figure 2.
2. A **Versioning** system that enables users to generate alternative versions from any point in the current design, facilitating the exploration of different architectural decisions and navigation across previous states.
3. A **Metrics** panel designed to evaluate the solution through the analysis of the *Access Pattern*, *Cost of Recovery*, *Redundancy*, and *Completeness* metrics proposed in [Vera-Olivera and Holanda 2024].
4. A **Comparison** feature that allows users to visualize and contrast the performance of the designed versions, analyzing the results of each metric to determine the optimal solution, as illustrated in Figure 3.

User interaction begins on the modeling canvas, where the data structure is defined. Once an initial proposal is created, branching enables exploration without affecting the base design. After schema configuration, the evaluation engine generates metrics, and the comparison view highlights quantitative differences to guide the selection of the solution that best meets the technical requirements.

5. Conclusions

The development of *Qolqa* advances document-oriented NoSQL database design by reducing reliance on subjective expertise through evaluation metrics. Built on FSD and Clean Architecture, it enables efficient query integration and version management. By providing indicators such as redundancy, recovery cost, access patterns and completeness *Qolqa* supports informed decision-making, helping identify optimal, scalable data architectures aligned with system requirements, regardless of user expertise.

As future work, we propose incorporating a metaheuristic algorithm to automate design exploration and optimize configurations based on *Qolqa*'s metrics. Furthermore,

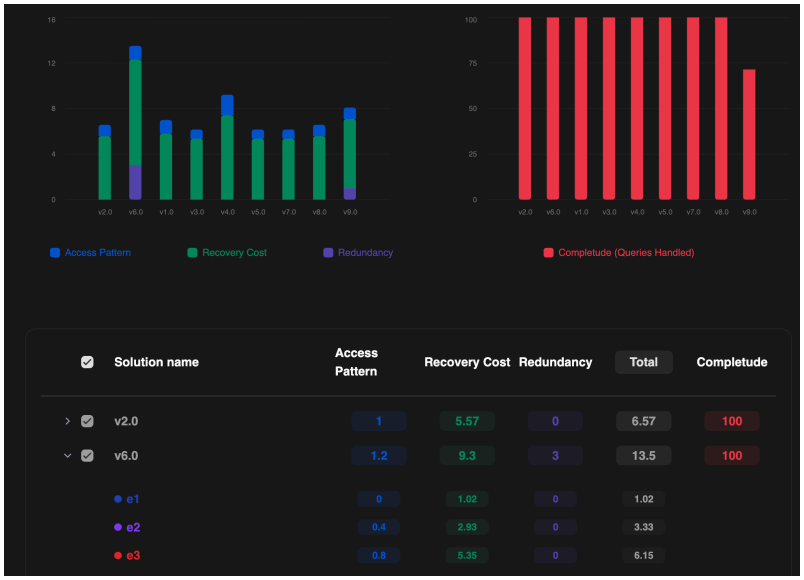

Figura 3. Example of the comparison page

we plan to conduct a practical field study to evaluate the tool's impact by comparing database schemas designed by an expert against those created by a beginner.

Referências

Datensen (2025). Moon modeler – nosql data modeling for mongodb & mongoose. https://www.datensen.com. Sitio web oficial.

DbSchema (2025). Dbschema: Diseñador de bases de datos con diagramas er. https://dbschema.com. Sitio web oficial.

Feature-Sliced Design (2018). Docs. `https://feature-sliced.design/docs/get-started/overview`. Accessed: 2026-04-16.

Hackolade (2025). Hackolade: Data modeling tool for sql and nosql. https://hackolade.com. Sitio web oficial.

Luiz, F. C. (2025). Incorporação do modelo de agregados na ferramenta brmodeloweb.

Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, Boston.

Mozaffari, M., Dignös, A., Gamper, J., and Störl, U. (2024). Self-tuning database systems: A systematic literature review of automatic database schema design and tuning. *ACM Comput. Surv.*, 56(11).

Vera, H., Boaventura, W., Holanda, M., Guimaraes, V., and Hondo, F. (2015). Data modeling for nosql document-oriented databases. In *CEUR Workshop Proceedings*, volume 1478, pages 129–135. sn.

Vera-Olivera, H. and Holanda, M. (2024). Métricas para análise de esquemas em banco de dados nosql orientado a documentos. In *Anais do XXXIX Simpósio Brasileiro de Bancos de Dados*, pages 381–393, Porto Alegre, RS, Brasil. SBC.