POLITECNICO DI TORINO
Repository ISTITUZIONALE

NoER: A Tool for End-to-End Design of Document Databases

Original
NoER: A Tool for End-to-End Design of Document Databases / Avignone, A., Chiusano, S., Fiori, A., Giorza, S., Panduri, S., Torlone, R.. - ELETTRONICO. - 4279:(2026), pp. 41-45. (2026 ER: Doctoral Consortium, Tutorials, Posters and Demos, ER-Companion 2026 St. John's, NL (Canada) October 5-8, 2026).

Availability:
This version is available at: 11583/3017394 since: 2026-11-02T11:25:25Z

Publisher:
CEUR-WS

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

20 November 2026

NoER: A Tool for End-to-End Design of Document Databases

Andrea Avignone[1,†], Silvia Chiusano[1,*,†], Alessandro Fiori[1,†], Sara Giorza[1,†], Santa Panduri[1,†] and Riccardo Torlone[2,†]

[1]Dept. of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi, 24, Torino, 10129, Italy
[2]Dept. of Civil, Computer Science and Aeronautical Technologies Engineering, Università Roma Tre, Roma, Italy

Abstract

Database design is a crucial practice in relational databases, following well-established patterns. The rising popularity of document-oriented NoSQL databases has changed how data modeling is perceived due to their flexible schema. As a consequence, data modeling is often reduced to an application-driven approach, where conceptual design is not addressed and schema decisions are tightly coupled with implementation details. Recent work has shown that Entity-Relationship (ER) modeling principles can be adapted to design document schemas, but practical tool support remains limited. We present NoER, a web-based system that supports a conceptually grounded and guided design process for document-oriented databases using a revised version of the ER model. It guides users through an end-to-end workflow, from conceptual schema definition to the interactive exploration of translation alternatives and the automatic generation of a MongoDB-compatible JSON schema. The demonstration showcases how NoER helps users compare design options, apply document design patterns consistently, and derive document schemas that are aligned with application requirements and easier to maintain.

Keywords

NoSQL database design, conceptual modeling, database design tool, document-oriented databases, MongoDB

1. Introduction

Document-oriented NoSQL databases, like MongoDB, are widely adopted for their flexibility, scalability, and support for semi-structured data. However, their schema flexibility often leads to the misconception that conceptual modeling is unnecessary, resulting in a design process driven mainly by the application and by physical details. Unlike relational database design, which separates the conceptual level from the logical one, document-based database design often collapses these concerns, forcing simultaneous consideration of domain semantics, document structure, and design patterns. This limits the evaluation of alternatives and alignment with requirements, leading to experience-driven decisions whose consequences only emerge after implementation.

Several works have challenged the assumption that NoSQL systems do not require systematic design, reaffirming the separation between conceptual, logical, and physical levels [1, 2]. Entity–Relationship (ER) models have been shown to provide a system-independent conceptual formalism, which can be translated into document-oriented schemas through well-defined rules [3, 4].

Despite these advances, practical support for conceptually grounded document database design remains limited. Existing tools for document database design, such as Hackolade [5] and DbSchema [6], focus primarily on logical schema specification and visualization, with limited support for conceptual modeling or structured translation, while designers are still left to manually connect conceptual models

ER2026: Companion Proceedings of the 45th International Conference on Conceptual Modeling: Doctoral Consortium, Tutorials, Posters and Demos, October 5-8, 2026, St. John's, NL, Canada.

*Corresponding author.
†These authors contributed equally.
andrea.avignone@polito.it (A. Avignone); silvia.chiusano@polito.it (S. Chiusano); alessandro.fiori@polito.it (A. Fiori); sara.giorza@polito.it (S. Giorza); santa.panduri@polito.it (S. Panduri); riccardo.torlone@uniroma3.it (R. Torlone)
0000-0001-7003-129X (A. Avignone); 0000-0002-5740-5004 (S. Chiusano); 0000-0003-3819-2696 (A. Fiori); 0000-0003-1484-3693 (R. Torlone)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-10-01

CEUR-WS.org/Vol-4279/Paper-163.pdf

with document-oriented schemas, reason about alternative translation patterns, and assess their impact without systematic guidance.

Motivated by this gap, we propose NoER (*Not Only ER*), a design environment that brings conceptual modeling back into document database design by combining: (i) an extended ER modeling layer to capture design-relevant constraints; (ii) an interactive translation assistant that guides users through a catalog of document-design patterns; and (iii) an automatic generator of JSON Schema for immediate validation and deployment. NoER is built on ER_d, a revised ER model to capture the characteristics of document-oriented databases [7]. A key feature of NoER is its guided design workflow: at each modeling step, it proactively recommends design alternatives that are consistent with the current conceptual model and with document-design guidelines. NoER provides a design environment for MongoDB-oriented database design.

NoER can serve different types of users, such as ER designers moving to MongoDB, experienced MongoDB users seeking systematic documentation and faster schema generation, and students learning document database design in an intuitive environment. It is currently used in a Master's course in Computer Engineering at Politecnico di Torino (Turin, Italy), where students with a relational background are introduced to NoSQL modeling. NoER is available at https://nosql.designerdb.io; a demo video showing how it supports end-to-end document schema design from a conceptual specification is available at https://youtu.be/63VQ41TJLq4.

2. NoER Overview

Underlying Conceptual Data Model: ER_d. NoER is based on ER_d, our revised version of the ER model designed for document-oriented databases [7]. It builds on the original ER constructs (entity, relationship, attribute) to preserve consistency, but adapts them to fit document-native structures by relaxing some relational constraints. Following document database principles, ER_d replaces fixed entities with *partially homogeneous* entities to support polymorphism: optional attributes now represent a possible *missing field* in some documents. Generalizations, used to define subsets within an entity, also support polymorphism. Arrays and subdocuments are represented through *multivalued* simple and composite attributes. Recursively composite attributes create *hierarchical attribute trees*. To handle temporal data, ER_d introduces *time entities*. Relationships and identifiers keep the same meaning as in ER. For each entity E_i taking part in a relationship R, ER_d keeps minimum and maximum cardinalities, showing whether participation is mandatory or optional, and whether an entity occurrence can be linked to one or several relationship occurrences. ER_d defines translation rules that map conceptual constructs to JSON schema patterns, supporting different options (e.g., referencing vs embedding).

Architectural Overview. NoER turns the ER_d design principles into an application that produces ready-to-use JSON schemas. It offers a complete environment for designing MongoDB databases: users start by creating a conceptual schema with the ER_d model and are then guided toward a consistent JSON schema, with the possibility to refine design choices. NoER automates the whole methodology and validates each step, ensuring correctness and consistency even for complex schemas. Implemented as a web-based application, it requires no installation beyond a standard browser.

NoER relies on a *Catalog* of reference ER_d constructs and document design patterns following MongoDB documentation [8], along with a *Guidance* component that assists users throughout the design process. The design flow includes the following components. (i) The **Conceptual Modeling Engine** provides a graphical editor for defining entities, relationships, and attributes using the ER_d model, with document-specific constructs. (ii) The **Translation Engine** implements pattern-based translation rules to map conceptual constructs to document-oriented representations based on MongoDB design patterns. (iii) The **JSON Schema Generation Module** constructs the target document schema and generates a MongoDB-compatible JSON schema ready to be deployed in MongoDB Atlas. Before export, users can adjust field data types, which default to `string` but also support `integer`, `boolean`, and others.

Design Guidance in NoER. The tool supports an incremental, interactive design process, guiding users in correctly applying ER_d principles and MongoDB patterns, while keeping the *designer in the loop* to avoid over-automation. The system highlights modeling issues through warnings and error alerts, enables only valid options, offers contextual feedback for fixes, and recommends patterns based on entity participation and field access. Its user-centered interface minimizes complexity through progressive disclosure, showing features only when relevant. Users can inspect, validate, and refine intermediate outputs, so automated actions never override domain knowledge or design intent. The following guidance actions are provided throughout the design process:

Conceptual Design Guidance. Through a graphical ER_d editor, users define *entities*, *relationships*, and *attributes*, as well as hierarchies via *generalizations*. Temporal aspects can be modeled through *time entities*. Attributes may be atomic or composite, with support for recursive nesting. Atomic attributes can be designated as *identifiers* to distinguish entity occurrences. NoER only enables options valid for the selected construct, preventing and detecting rule violations in the conceptual schema: (i) actions that could introduce critical issues are disabled (e.g., identifiers are not allowed for child entities in generalizations or for relationships); (ii) violations to be fixed before proceeding are reported as errors (e.g., relationships linked to a single entity, missing entity identifier); (iii) naming conflicts that may cause ambiguity are detected (e.g., duplicate entity/relationship names, repeated attribute names within an entity/relationship); (iv) aspects affecting completeness, but not correctness, are reported (e.g., missing attributes beyond the identifier). Explanatory messages guide the user in fixing each issue.

Translation Process Guidance. The translation process is incremental and interactive. NoER enforces a clear sequence: entities are translated first, considering the types of their associated attributes and any possible involvement in generalization, followed by relationships. At each step, the tool enables only the options relevant to the selected construct. *Entities* map to collections, where each document represents one entity occurrence. For time-based or repeated events, the *bucket pattern* groups multiple occurrences over time. *Relationships* are translated using MongoDB patterns (embedding, referencing, extended referencing, or subset), based on cardinalities and participation constraints. During translation, the system provides a high-level graphical view of the schema, showing main and embedded collections and their references. Users can inspect the evolving document structure before generating the final JSON schema.

NoER supports the following three scenarios for selecting relationship translation patterns, ranging from minimal to increasingly guided system intervention:

Translation Pattern Display. For each relationship, the user selects a MongoDB pattern from those shown by the system, and translation is applied automatically.

Guided Pattern Selection. For each relationship, NoER suggests a design pattern based on the user's input on data access, update frequency, and relationship characteristics, along with its pros and cons. The user selects one, and translation is applied automatically.

AI-Driven Translation. NoER generates a global translation plan, suggesting a pattern for every relationship, with rationale and confidence level. The user can apply these patterns one at a time or all at once, or refine them by revising guidance and rerunning the AI. This approach relies on a structured prompt given to the LLM, automatically including the conceptual schema summary (entities, attributes, relationships, and cardinalities), the output format constraints for valid JSON, and the reference MongoDB translation patterns. In addition, the prompt includes global workload assumptions and per-relationship workload details (e.g., joint reads, growth, independent updates, navigation direction), which the user specifies through a graphical interface. NoER currently integrates three LLMs that users can choose from: Cohere North Mini Code (free), OpenAI GPT-5.1-Codex-Mini, and Anthropic Claude Haiku 4.5.

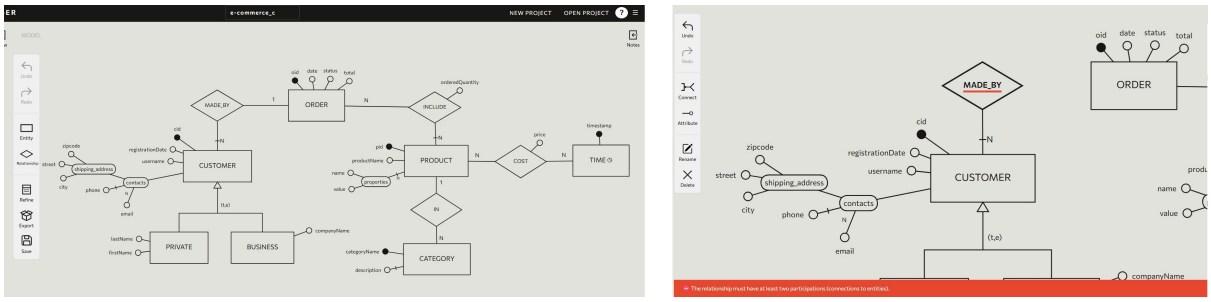


Figure 1: ER$_d$ conceptual schema of the running "e-commerce" use case created. The figure shows an example of detected error during conceptual modeling when a relationship is missing a connection.

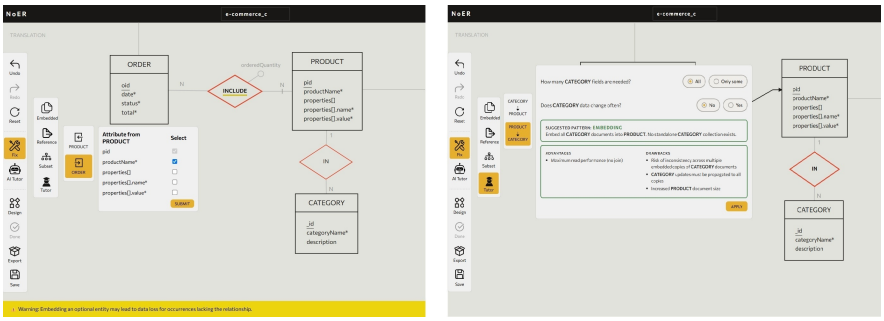


Figure 2: Examples of document-oriented relationship translation patterns supported by NoER, including extended referencing and embedding when using the *Guided Pattern Selection* functionality.

3. Illustrative Example

This section presents an illustrative example designed with the implemented tool. The complete example is available in the demo video at https://youtu.be/63VQ41TJLq4.

In the example conceptual model in Figure 1, ORDER, PRODUCT, and CATEGORY are modeled as simple entities, while CUSTOMER is a generalization of PRIVATE and BUSINESS. Entities include identifiers and attributes. For example, CUSTOMER has `userName`, a mandatory atomic attribute, and `contacts`, a mandatory composite attribute including the attribute `phone`, the multi-valued attribute `email` and the composite attribute `shipping_address`. The time entity TIME is used to model the product price evolution over time. Relationships such as MADE_BY connect entities. Figure 1 shows that the system identifies the MADE_BY relationship as incorrect since it connects only one entity.

Figures 2 and 3 illustrate how NoER supports the interactive application of document-oriented translation patterns. As shown in Figure 2, through the toolbar users can select among the available alternatives for each relationship. The *extended reference* pattern is selected to represent PRODUCT in ORDER, allowing users to choose which fields to include in addition to the identifier, such as the *productName* field. For relationship IN the user exploits the *Guided Pattern Selection* functionality, which suggests embedding CATEGORY into PRODUCT based on the input provided by user. In Figure 3 the user uses the *AI-Driven Translation* functionality. The user selects an LLM model, sets general guidelines, and provides workload details for each relationship. The LLM then returns a summary of its choices, including the suggested pattern, description, and confidence level for each relationship.

4. Conclusions

NoER is a complete environment for designing MongoDB-based document databases. Users first create a conceptual schema with an ER-based model, then are guided to a consistent JSON schema, refining design choices along the way. Future work will address current limitations by extending the supported design-pattern catalog (e.g., time-series patterns beyond bucketing) and enhancing AI-Driven Translation mode to reduce required user input.

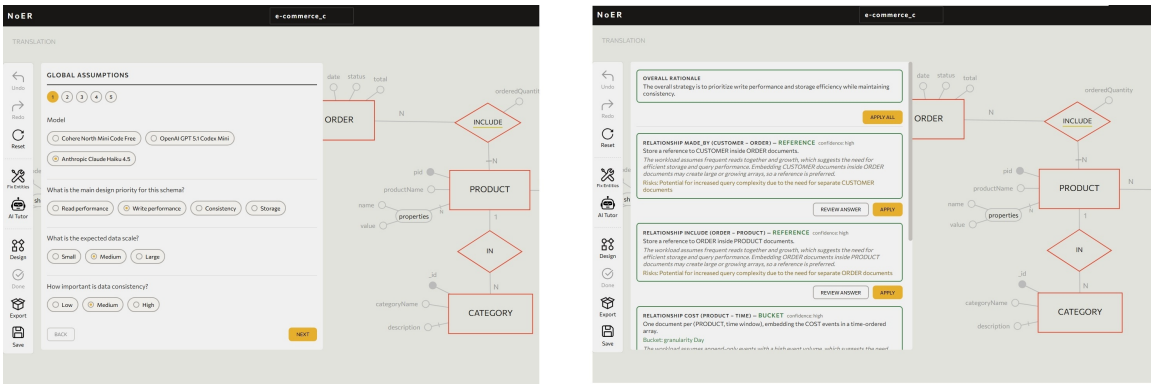


Figure 3: Examples of document-oriented translation patterns for relationships. The figure shows an example use of the *AI-Driven Translation* functionality.

Acknowledgments

This study was partially carried out within the FAIR (PNRR - Missione 4 Componente 2, Investimento 1.3 D.D. 1555 11/10/2022, PE00000013) and the PRIN-MUR project 2022XERWK9-S-PIC4CHU. This manuscript reflects only the authors' views and opinions, neither the EU nor the EC can be considered responsible for them.

Declaration on Generative AI

During the preparation of this work, the authors used Claude Sonnet 5 in order to: Grammar and spelling check. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] A. Badia, D. Lemire, A call to arms: revisiting database design, SIGMOD Rec. 40 (2011) 61–69. doi:10.1145/2070736.2070750.

[2] M. Carey, W. Alkowaileet, N. DiGeronimo, P. Gupta, S. Smotra, T. Westmann, Towards principled, practical document database design, Proc. VLDB Endow. 18 (2025) 4804–4816. doi:10.14778/3750601.3750606.

[3] A. Belussi, S. Migliorini, From er conceptual models to document-based nosql logical models, in: P. K. Chrysanthis, K. Nørvåg, K. Stefanidis, Z. Zhang, E. Quintarelli, E. Zumpano (Eds.), New Trends in Database and Information Systems, volume 2676 of *Communications in Computer and Information Science*, Springer Nature Switzerland, Cham, 2026, pp. 56–66. doi:10.1007/978-3-032-05727-3_6.

[4] S. Banerjee, A. Sarkar, Logical level design of NoSQL databases, in: 2016 IEEE Region 10 Conference (TENCON), IEEE, 2016, pp. 2360–2365. doi:10.1109/TENCON.2016.7848452.

[5] Hackolade, Nosql data modeling and visualization, https://hackolade.com/, 2026. Accessed: Jan. 2026.

[6] DbSchema, Mongodb schema design tool, https://dbschema.com/mongodb-tool.html, 2026. Accessed: Jan. 2026.

[7] A. Avignone, S. Chiusano, A. Fiori, R. Torlone, Boosting the entity-relationship model for document-oriented databases, in: Cooperative Information Systems, volume 15535 of *Lecture Notes in Computer Science*, Springer Nature Switzerland, Cham, 2026, pp. 550–561. doi:10.1007/978-3-032-15538-2_33.

[8] MongoDB, Data modeling with mongodb, https://www.mongodb.com/docs/manual/data-modeling/, 2026. Accessed: Jan. 2026.