# Teaching Query-Driven Design in Aggregate-Oriented NoSQL Systems: Resources, Methodology, and Tool Support

Barbara Catania*, Giovanna Guerrini and Amer Al Khoury

*DIBRIS - University of Genoa, Genoa - Italy*

**Abstract**

Teaching data design in aggregate-oriented NoSQL systems (ANoSQL) is a core topic in Advanced Data Management courses. A key challenge for students is understanding the main differences between SQL and ANoSQL data design approaches. SQL systems are typically used for integration-oriented databases with fully structured, normalized schemas, and a clear separation between logical and physical levels. In contrast, ANoSQL systems target application-oriented databases and favor flexible, denormalized schemas, optimized for workload-specific access patterns. In these systems, logical and physical levels are tightly integrated to minimize the number of nodes accessed during query execution. Unfortunately, existing ANoSQL design approaches are largely system-specific, often overlook this interplay, and offer limited reusability in educational contexts. Based on our experience teaching an Advanced Data Management course at the University of Genoa since 2016, we have therefore developed specific educational resources to address this gap and support students in understanding ANoSQL design principles and efficiency considerations. The resources include lesson plans, slides with methodological content and examples, and an optional formative tool. They emphasize query-driven design, clarify the logical–physical interplay, introduce a simple system-independent methodology for generating ANoSQL logical schemas enriched with physical metadata, and provide guidance for translating them into specific ANoSQL systems such as MongoDB and Cassandra.

**Keywords**

aggregate-oriented NoSQL data store, query-based design, logical-physical design methodology



## 1. Introduction

Teaching an Advanced Data Management (ADM) course in Master's programs in Computer Science, Engineering, and related disciplines has become increasingly challenging due to evolving market expectations and the rapid evolution of DBMS technologies. ADM learning outcomes span competencies ranging from large-scale data management principles and NoSQL system architectures to data design and management for AI-driven applications. Given their wide applicability and efficiency in large-scale distributed settings, aggregate-oriented NoSQL (ANoSQL) systems (key-value, document-based, and column-family data stores) have become a core topic in ADM courses. All ANoSQL systems rely on the unifying concept of *aggregate* [1], a self-contained unit of data grouping information accessed together and managed as a single logical and physical unit.

Due to their distributed nature, one of the main difficulties when learning ANoSQL systems is *data design*, since architectural choices and data management principles differ substantially from relational ones. From a teaching perspective, these differences require educational material that explicitly addresses the impact of distribution on data modeling, considering the following key factors [1, 2].

*K1. Integration and application databases.* Traditional relational systems rely on a single, integrated database with a shared, normalized schema serving multiple applications, with constraints centrally

*DataEd'26: 5th International Workshop on Data Systems Education*

Published in the Proceedings of the Workshops of the EDBT/ICDT 2026 Joint Conference (March 24-27, 2026), Tampere, Finland

*Corresponding author.

barbara.catania@unige.it (B. Catania); giovanna.guerrini@unige.it (G. Guerrini); amer.alkhoury89@gmail.com (A. A. Khoury)

0000-0002-6443-169X (B. Catania); 0000-0001-9125-9867 (G. Guerrini)

enforced by the DBMS. In contrast, large-scale systems often adopt application-specific databases, where each application owns its data and selects the schema and technology best suited to its workload, moving part of constraint management to the application [1]. This architectural change must be explicitly reflected in the resources employed in ANoSQL data modeling education.

*K2. Moving towards cluster computing.* Large data volumes typical of data-intensive applications are stored across clusters, where data are partitioned and replicated over multiple nodes. Partitioning enables intra-query parallelism and requests are executed concurrently on different data partitions and nodes. As a consequence, data design decisions directly affect query execution and performance and represent a key learning challenge in ANoSQL education [2, 3].

To effectively learn ANoSQL data design, students thus must rethink traditional database design principles in light of these characteristics. However, the lack of a reference standard for ANoSQL data modeling and querying makes this task quite challenging [2, 4]. Existing methodologies are often specific systems and thus not easily generalizable [5, 6, 7]; even approaches that abstract from system-dependent features typically defer decisions on data distribution and storage to later, system-specific steps [4]. This limits their effectiveness as reusable teaching artifacts.

Based on these considerations, we propose educational resources supporting ANoSQL data design teaching through a simple system-independent methodology that integrates logical and physical requirements and emphasizes abstraction. The resources consist of modular teaching materials (lesson plans, slides, and worked examples) organized into four methodological slide blocks and one exercise block with explicit learning objectives, complemented by a lightweight formative tool supporting self-directed learning. The aim is to provide reusable materials that clarify core ANoSQL data design principles and support students in reasoning about design choices and their implications, rather than to empirically evaluate the corresponding learning outcomes.

The remainder of this paper is organized as follows. Section 2 presents the educational context and the proposed resources, with guidelines for their reuse. Sections 3 and 4 detail the system-independent ANoSQL data design methodology at the logical and physical levels, highlighting the related educational purposes. The translation of the system-independent schemas into system-specific schemas is then discussed in Section 5. Finally, Section 6 discusses related work while Section 7 concludes the paper.

## 2. Educational Context and Resources

**Educational Context** The proposed artifacts were developed based on our experience in teaching a graduate course on Advanced Data Management at the University of Genoa since 2016, within the Master's programs in Computer Science and Computer Engineering. The course enrolls about 50 students per year (43 in 2025/2026: 37% Computer Science, 44% Computer Engineering, 19% international programs). Students are expected to have prior knowledge of relational database design, SQL, and basic database system concepts.

The course covers advanced data management topics, focusing on large-scale processing for data-intensive applications and ANoSQL data stores. A central learning objective is ANoSQL data design, requiring students to rethink traditional database design principles in light of large-scale system characteristics. The main ANoSQL design principles that students are expected to master are the following [1, 2].

*P1. Query-based design and nested aggregates.* In distributed settings, join operations incur high communication costs. As a result, ANoSQL logical design relaxes normalization and groups attributes that are frequently accessed together into a single aggregate, enabling query execution without cross-collection joins and leading to nested data structures.

*P2. Many ANoSQL data models, one single unifying concept: the aggregate.* Despite differences among ANoSQL data models, all systems are built around the notion of aggregate, abstractly represented as a $(key, value)$ pair. Models differ in how much of the $value$ can be accessed and processed by the system. For example, in key-value stores, the $value$ is opaque and can only be processed by applications; in column-family stores, only first-level components are system-accessible while in document-based

systems the $value$ is fully accessible.

*P3. Interplay of the logical and physical levels.* Unlike relational systems, ANoSQL systems tightly couple logical and physical design. The workload drives both aggregate design and their physical placement across cluster nodes through the choice of a partition key, which should co-locate aggregates frequently accessed together to minimize cross-node communication and ensure predictable performance. Moreover, index selection remains challenging, as indexing strategies must align with both aggregate structures and data partitioning to efficiently support the workload.

**Resource Organization** Effective ANoSQL design education should build on these three key principles. Our review of existing ANoSQL design approaches (see Section 6) shows that all methodologies adopt a query-driven process based on nested aggregates [1, 2] (P1). However, most are tied to specific ANoSQL systems or models and are not easily generalizable [5, 6, 7, 8, 8, 9], while a few abstract from system-dependent features and define aggregate schemas at a more abstract level centered on the notion of aggregate [4, 10, 11] (P2), from which system-specific schemas are derived through a translation step. In most cases, physical design aspects, such as partition-key and index selection, are addressed in a system-specific manner [4, 7, 9]. We argue that postponing physical design aspects in an educational context may hinder students' early understanding of performance implications and of the tight coupling between logical and physical design choices (P3). At the same time, the notion of aggregate provides an effective abstraction for a system-independent methodology, capturing workload-driven principles while avoiding unnecessary system-specific complexity. Starting from these considerations, we developed ANoSQL design resources that integrate logical and physical design within a system-independent methodology to support coherent and effective learning. The proposed material is organized into four main blocks of slides:

*Block 1* serves as an introduction, motivating query-driven design and nested aggregates (P1) and justifying the need for a system-independent ANoSQL design methodology (P2).

*Block 2* introduces a simple yet effective model for specifying aggregate schemas in a system-independent way, grounded in existing standards like JSON Schema, and presents a lightweight methodology for their generation, according to (P1) and (P2).

*Block 3* concerns physical design and its interplay with logical issues (P3). It is used to introduce a reference notion of query efficiency for ANoSQL systems, upon which a system-independent methodology for partition-key selection is provided, together with educational material for understanding how unique constraints can be enforced in ANoSQL systems and why and when index creation matters. Physical design choices are then represented inside aggregate schemas, extending the model presented in Block 2.

*Block 4* shows how an aggregate schema, including logical and physical information, can be translated into concrete schemas for key-value, document-based, and column-family systems, assuming systems and data models have already been presented in the course.

All concepts are introduced through a single running example consistently used across the four slide blocks. A final *Block 5* provides alternative examples covering all introduced design concepts. Blocks 1–4 are intended for frontal lectures (approximately 2 hours for Block 1 and 4 hours for Blocks 2–4), optionally integrated with instant polling, or delivered in a flipped-classroom setting, particularly for Block 4. Block 5 is designed for exercise sessions (2 hours for each of Blocks 2–4), which may be conducted collaboratively. The blocks are designed to take approximately 16—20 hours in total.[1]

The slide blocks assume prior knowledge of basic concepts of distributed data management. Depending on course organization, Block 1 can be omitted; Block 3 must follow Block 2, while Block 4 is intended for courses covering specific ANoSQL systems. To complement Blocks 2 and 4, an optional lightweight self-study tool is also available.[2] Developed in JavaScript, it supports ER diagram and

[1]All slide blocks are available as PDF files in the github project available at https://github.com/gioguerrini/ANoSQL-Methodology.

[2]The tool is available at https://amazing-benz-6643f3.netlify.app/.

workload specification, generates logical aggregate schemas as JSON schemas, and translates them into Cassandra commands. As it is based on an earlier version of the methodology, it covers only logical design. For this reason, it is not further discussed in the next sections that focus instead on Blocks 2–4, representing the main contribution of this work.

## 3. Block 2- Logical Design

Block 2 introduces the core artefact for logical ANoSQL design. It guides students in deriving aggregate-oriented logical schemas from a conceptual diagram and a query workload, emphasizing query-driven design while remaining system-independent. The proposed methodology simplifies existing approaches (e.g., [4]), builds on students' prior database design knowledge, and introduces only a minimal set of new concepts. It is heuristic-based, leaving room for alternative design choices to be refined when moving from system-independent schemas to system-specific ones (Block 4), encouraging students to reason about design trade-offs.

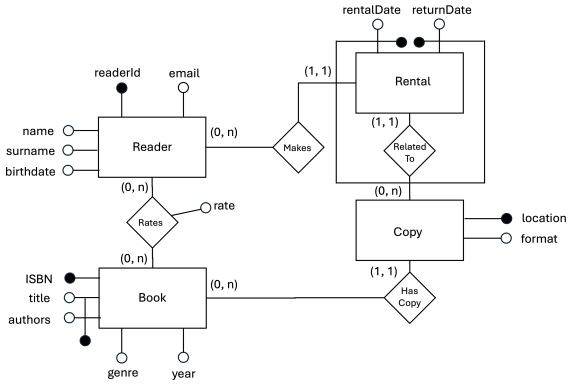
Given an Entity-Relationship (ER) diagram and a workload in natural language, the methodology produces a set of aggregate-oriented logical schemas, represented in a JSON-inspired schema meta-notation. To reduce cognitive load, the focus is exclusively on queries with equality-based selection conditions. The process consists of three steps and three main heuristics, described below.

**Step 1: Query modeling** Each query is represented in a formal and unambiguous way, helping students clarify which entities and attributes are accessed. More precisely, each workload query Q is represented as a triple Q(E, LS, LP), where: (i) E is the *aggregation entity* used to group all selection and projection attributes accessed by Q; (ii) LS (LP) is the list of entities containing selection (projection) attributes. For each entity $E_i$ in LS or LP, the representation specifies the $E_i$ attributes used in the query and the path linking $E_i$ to E.[3] Paths are expressed as sequences of association names (or initials, when unambiguous). When $E_i$ coincides with E, the empty path is denoted by !.
*Heuristics.* To ensure a general, system-independent design (P2), we introduce *heuristic H1* for selecting the aggregation entity E. H1 favors aggregate schemas in which selection attributes appear as simple (non-nested) properties, reducing selections on nested attributes that may be inefficient or unsupported in some ANoSQL systems. Accordingly, E is chosen among the entities in LS, or, if none exist, among those in LP; when multiple candidates are available, preference is given to the entity on the one-side of the highest number of associations linking selection entities.

**Step 2: ER schema annotation** The second step annotates the ER diagram with workload information, starting from the result of Step 1. For each query Q(E, LS, LP): (i) entity E is marked with label Q and represented using a double rectangle; Q is referred to as the *query associated with* entity E; (ii) for each entity $E_i$ in LS or LP, the path linking $E_i$ to E is annotated with arrows directed toward E and labeled with Q; (iii) attributes involved in selections and projections are annotated with labels Q-S and Q-P, respectively.

**Step 3: Aggregate schema generation** During the last step, students learn how to transform the annotated ER diagram into a set of system-independent aggregate schemas. For each aggregation entity E, according to (P1), one logical schema is derived by considering all selection and projection attributes used in the queries associated with E. The resulting schemas are represented in a JSON-inspired Schema Meta-notation (JSON-SM), extended to capture ANoSQL-specific design information. JSON-SM represents the high-level aggregate structure by retaining property names and nesting relationships, and supports the explicit representation of identifiers. Complex attributes are expressed using curly brackets {} for nested objects and square brackets [] for collections. The notation intentionally departs from JSON and JSON Schema standard: JSON-SM is not meant to be parsed or validated, but serves as a

[3] Listed only if they do not correspond to the whole entity attribute set.




(a) ER diagram for the library book rental domain. Entities are shown as rectangles, associations as diamonds, attributes as labeled circles. Cardinality constraints indicate minimum and maximum participation as pairs of values (min,max). Black circles denote identifiers, which may combine entity attributes with identifiers of related entities (e.g., `Rental`).

| | | |
|---|---|---|
| $Q_1$ | Determine the average age of readers | `Q1(Reader,[], [Reader(birthdate)_!])` |
| $Q_2$ | For each reader, determine the name, surname, and related rated books | `Q2(Reader,[], [Reader(name,surname)_!, Book(title,authors)_R])` |
| $Q_3$ | Determine the name and the surname of readers who rated the book "1984" by "George Orwell" | `Q3(Book, [Book(title,authors)_!], [Reader(name,surname)_R])` |
| $Q_4$ | Given a book, determine all information of copies that contain it | `Q4(Book, [Book(title,authors)_!], [Copy_H])` |
| $Q_5$ | Determine all the copies with format "hardcover", rented from a certain date | `Q5(Copy,[Copy(format)_!, rental(rentalDate)_Rt], [Copy(location)_!])` |
| $Q_6$ | Determine the copies with format "hardcover" containing the book "1984" by "George Orwell", together with the readers that rented them | `Q6(Copy, [Copy(format)_!, Book(title,authors)_H], [Copy(location)_!, Reader(readerId)_MRt])` |

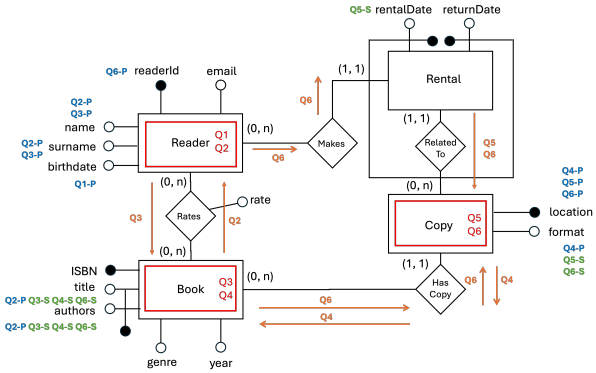
(b) Workload and query representation. When a query specifies "all the information of entity E," it is assumed to refer to all attributes of E. When a query refers simply to entity E, it is assumed to refer to the identifier of E.

**Figure 1:** ER diagram and workload/query representation.

lightweight educational notation that remains close to JSON syntax, reflecting the fact that aggregates are often implemented as JSON documents in ANoSQL systems. At the same time, it extends JSON-based representations with design-specific information not supported by JSON Schema, such as identifiers and physical design elements (e.g., partition keys; see Section 4). This enables reasoning about both logical and physical design within a single, system-independent representation.

*Heuristics.* Depending on the cardinality constraints of the associations linking E to other entities, the aggregate may include complex attributes. Semantically equivalent but syntactically different representations can be generated at this stage. To ensure a general system-independent schema, students are encouraged to limit the depth of property nesting (*heuristic H2*). An additional heuristic concerns identifiers (*H3*). Although constraint checking in ANoSQL systems is typically delegated to the application, document-based and column-family systems still support uniqueness constraints, though with limitations. It is therefore important that students learn that one identifier for E should be selected at this stage. To avoid introducing irrelevant attributes, the identifier that shares the greatest number of attributes with the selection and projection attributes should be selected.

**Example 3.1.** *Figure 1b shows a reference workload and the corresponding query representation for the ER diagram in Figure 1a. Query* $Q_6$ *includes selection conditions on attribute* `format` *of entity* `Copy` *and on attributes* `title` *and* `authors` *of* `Book`*, projections over* `Copy` *and* `Reader` *entities. According to H1, the aggregation entity should be* `Copy` *since it appears in* `LS` *and on the one side of the association connecting the two selection entities. All entities except* `Reader` *are connected to the aggregation entity* `Copy` *through a single association;* `Reader` *instead is linked through a path composed of two associations, denoted as* `MRt` *in the query representation (*`Makes` + `RelatedTo`*). An alternative would be to select* `Book` *as the aggregation entity; however, since one book may correspond to multiple copies, this would introduce nested selection attributes (e.g.,* `location` *and* `format`*) in the schema, in contrast with H1.*




(a) Annotated ER diagram.

```
Book:   {  title, authors,
           rated_by: [{name, surname}],
           has_copies: [{location, format}] }

Reader: {  readerId, name, surname, birthdate,
           rates: [{title, authors}] }

Copy:   {  location, format,
           rentals: [{rentalDate, readerId}],
           title, authors }
```

(b) JSON-SM aggregate schemas.

**Figure 2:** Annotated ER diagram and JSON-SM aggregate schemas.

*Figure 2a shows the ER diagram annotated with the query workload, which serves as input to the final step producing the aggregate schemas in Figure 2b.*[4] *As an example, in the* Book *schema,* Book *attributes annotated with queries* $Q_3$ *and* $Q_4$ *are included as simple properties in the JSON-SM representation. Since the* Book *identifier is used in query selections, no further attributes should be added, according to H3. The identifier attributes are underlined. To complete the schema, attributes of entities connected to* Book *through annotated associations (*Reader *and* Copy*) are included as collection properties since they appear on the many side of the relationships.* ◊

## 4. Block 3 - Physical Design

To introduce physical design in a system-independent way, Block 3 focuses on (P3) and explains how logical design choices affect query execution efficiency in aggregate-oriented ANoSQL systems. Students learn to distinguish between *batch queries*, which scan the entire dataset, and *random-access queries*, which target specific data subsets and for which partition-key selection is crucial. We further show that query efficiency depends on (i) limiting execution to one or few nodes and (ii) avoiding full partition scans, through data ordering or indexing. Based on these principles, Block 3 guides students in understanding three complementary physical design aspects and one heuristic.

**Partition key selection** We show that a query is efficiently executed only when its selection predicates include equality conditions on all partition-key attributes, ensuring query routing to a single node. Additional selection conditions may restrict results to a subset of a partition but still require scanning the entire partition, increasing disk accesses. When some equality conditions on the partition key are missing, relevant aggregates may be distributed across multiple nodes, preventing efficient execution. *Heuristics.* Based on these observations, we introduce a heuristic for partition-key selection (*H4*) to support efficient execution for as many associated random-access queries as possible. Given an aggregate schema $A$ and the set of associated random-access queries $Q_A$, the partition key is derived from the intersection $int_{Q_A}$ of simple selection attributes across the queries in $Q_A$, with different choices depending on whether this intersection is empty or coincide/does not coincide with each set of query selection attributes.

**Enforcing uniqueness constraints** Unlike relational systems, enforcing a UNIQUE constraint in a distributed setting is costly, since conflicting aggregates may reside on different nodes. To avoid global

[4]Notice that nested properties could also be represented with an additional nesting level (e.g., `has_copies` : [{`copy` : {`location`, `format`}}]); however, this increases schema depth and, according to H2, should be avoided.

checks, ANoSQL guarantee uniqueness only locally within each node. As a consequence, identifiers must include the partition key, and minimality, typical of conceptual and relational identifiers, is no longer preserved. To clarify this distinction and simplify the translation into system-related schemas, we introduce the notion of *super-identifier*, defined as an identifier from the ER schema extended with the partition-key attributes.

**Index creation** We first note that in ANoSQL systems user-defined indexes are *local*, indexing the aggregates stored on a single node, typically through ordered tree-based structures (e.g., B-trees). Local indexes serve two main purposes: (i) an index on a super-identifier (with partition-key attributes first) enables efficient uniqueness checking and may be created by the system (e.g., in Cassandra) or by the user (e.g., in MongoDB); (ii) secondary local indexes can be defined on selection attributes of queries that are not efficiently supported by design, improving aggregate retrieval on each node.

To help student understand the strict relationship between logical and physical schemas, we extend JSON-SM to represent partition keys and super-identifiers.

**Example 4.1.** *Consider the* Book *schema. Queries* $Q_3$ *and* $Q_4$ *share the same selection attributes (*`title` *and* `authors`*); selecting them as the partition key ensures efficient execution, since both queries are routed to a single node. Now consider the* Copy *schema. Queries* $Q_5$ *and* $Q_6$ *do not share the same selection attributes, but their intersection is non-empty and corresponds to* `format`*. Selecting* `format` *as the partition key routes both queries to a single node; however, additional selection attributes may require scanning the partition. Secondary indexes on these attributes (e.g.,* `(title, authors)` *for* $Q_6$*) improve query efficiency. For* Book*, the super-identifier coincides with the partition key, while for* Copy *it is (*`format, location`*). In both cases, a unique local index can enforce the unique constraint. We use superscripts and overlined text to represent partition keys and super-identifiers in JSON-SM (e.g.,* $\overline{format^p}$, $\underline{location}$ *for the* Copy *entity).* ◊

## 5. Block 4 - System-oriented Design

The fourth slide block shows how a JSON-SM schema representing both logical and physical information can be systematically translated into schemas for key–value, document-based, and column-family systems, exposing students to the power of abstraction across data models while clarifying the expressive power and limitations of each ANoSQL system.

To perform the translation, the block first guides students in understanding how each system interprets a $(key, value)$ pair and the degree of system-level processing allowed on the value. For each category of ANoSQL system, the block relies on general rules and worked examples to explain: (a) whether the JSON-SM schema can be fully retained in the system (as in document-oriented systems such as MongoDB) or needs to be adapted (as in column-family systems such as Cassandra), while highlighting which queries remain efficiently supported at the system level and which do not; (b) how super-identifiers map to system-specific mechanisms (e.g., unique indexes in MongoDB and `primary key` declarations in Cassandra); (c) which system-specific commands implement the logical/physical schema. This block is complemented by hands-on activities on MongoDB and Cassandra deployed on a small departmental cluster.

## 6. Related Work

The workload-driven nature of ANoSQL data modeling has been recognized since the early stages of NoSQL systems [1] and alternative schemas are known to affect performance [3]. Over the last decade, numerous approaches have supported DBAs and data engineers in the complex task of workload-based data design (see [2] for a survey). Most of them separate conceptual and logical design, transforming ER- or UML-based models into ANoSQL schemas. Existing works differ in (i) focusing on a single system (i.e., system dependent) [7, 9, 12] or on multiple systems (i.e., system independent) [13, 10, 11, 4, 14, 15],

(ii) considering only conceptual models [11] or also workload queries, with [7, 14, 15, 9] or without query frequencies [10, 6, 12], and (iii) generating one [10, 11, 15, 12] or multiple target schemas [6], or evaluating alternatives based on a single [4, 6, 12] or multiple criteria [7, 14, 9]. Other approaches address schema design from different perspectives, such as object-NoSQL mappers [16] and automated sharding approaches [17] from application-level classes, focusing on complementary issues.

Despite the wide range of complexity of the mentioned approaches, they are often system-specific or highly detailed, rather than focusing on general principles, and primarily targeted at software practitioners rather than students. An exception is [8] which proposes a conceptually grounded ER-to-JSON mapping methodology, although limited to fully query-capable document-based systems. In designing a more general methodology with educational goals, intermediate abstract models that ensure technology independence become essential (see, e.g., [4, 13]), in line with the principles outlined in Section 2. Our methodology can be viewed as a pedagogically oriented simplification of such system-independent approaches (e.g., [4]), integrating both logical and physical design.

A unified methodology that captures common principles across different ANoSQL models is very valuable for data system education. Overall, data systems education remains a challenging component of many computer science curricula. Most research in this area focuses on query languages (primarily relational languages and SQL [18, 19] and, to a lesser extent, NoSQL query languages [20]). For what concerns modeling, some approaches target conceptual modeling [21] and relational logical modeling focusing on normalization and schema design [22, 23]. In contrast, NoSQL data modeling is less explored [24]. Exceptions include: (i) reports on experiences of introducing NoSQL systems in curricula [25, 26], emphasising query-driven modeling, data aggregation and denormalization; (ii) proposals offering insights into comparative modeling strategies and the flexibility of schema-less and graph-based systems [27]; (iii) works collecting learning analytics on the use of SQL and NoSQL systems through a React-based web application [28, 29].

## 7. Concluding Remarks

In this paper, we presented a set of educational resources to support the teaching of ANoSQL data design in graduate-level Advanced Data Management courses. The approach grounds this design in key properties of large-scale distributed environments, building on foundational data management knowledge acquired in undergraduate curricula. Compared to existing approaches, the resources are system-independent, emphasize abstraction, and highlight the interplay between logical and physical design.

The resources were deployed during the first term of the 2025/26 academic year and used in lectures, exercise sessions, and a two-task autonomous assignment. Formative feedback focused on aggregate schema correctness, partition-key selection, and consistency between logical and physical design choices. The optional tool supported self-checking and exploration of alternative designs. While further activities are needed to demonstrate measurable learning impact, preliminary observations are encouraging: more than 50% of the submissions applied the methodology correctly and, in the first two exam sessions, 76% of students achieved a passing grade in the modeling exercise, improving on the 67% recorded in the 2024/25 academic year. This suggests that the resources support an effective and structured approach to ANoSQL design learning.

Future work includes a comparison with traditional domain-focused design and an experimental exploration of the performance impact of alternative schemas, to be addressed through an additional block and hands-on experiments on MongoDB and Cassandra.

## Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-5.2 for: paraphrasing and rewording; improving writing style; grammar and spell checking. After using this tool, the authors reviewed and edited the content and take full responsibility for the publication's content.

## References


[1] P. J. Sadalage, M. Fowler, NoSQL distilled: A brief guide to the emerging world of polyglot persistence, Pearson Education, 2013.

[2] N. Roy-Hubara, A. Sturm, Design methods for the new database era: A systematic literature review, Software and Systems Modeling 19 (2020) 297–312.

[3] N. Bansal, L. K. Awasthi, Are NoSQL databases affected by schema?, IETE Journal of Research 70 (2024) 4770–4791.

[4] L. Chen, A. Davoudian, M. Liu, A workload-driven method for designing aggregate-oriented NoSQL databases, Data & Knowledge Engineering 142 (2022) 102089.

[5] N. Bansal, S. Sachdeva, L. K. Awasthi, Schema generation for document stores using a workload-driven approach, The Journal of Supercomputing 80 (2024) 4000–4048.

[6] W. Y. Mok, A conceptual-model-based design methodology for MongoDB databases, in: Proc. ICICT, 2024, pp. 151–159.

[7] V. Reniers, D. V. Landuyt, A. Rafique, W. Joosen, A workload-driven document database schema recommender (DBSR), in: Proc. ER 2020, volume 12400 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 471–484.

[8] M. J. Carey, W. Y. Alkowaileet, N. Digeronimo, P. Gupta, S. Smotra, T. Westmann, Towards principled, practical document database design, Proc. VLDB Endowment 18 (2025) 4804–4816.

[9] M. Hewasinghage, S. Nadal, A. Abelló, E. Zimányi, Automated database design for document stores with multicriteria optimization, Knowledge and Information Systems 65 (2023) 3045–3078.

[10] A. de la Vega, D. García-Saiz, C. Blanco, M. E. Zorrilla, P. Sánchez, Mortadelo: Automatic generation of NoSQL stores from platform-independent data models, Future Generation Computer Systems 105 (2020) 455–474.

[11] J. Mali, F. Atigui, A. Azough, N. Travers, ModelDrivenGuide: An approach for implementing NoSQL schemas, in: Proc. DEXA 2020, volume 12391 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 141–151.

[12] M. Mozaffari, E. Nazemi, A. Eftekhari-Moghadam, CONST: Continuous online NoSQL schema tuning, Software: Practice and Experience 51 (2021) 1147–1169.

[13] P. Atzeni, F. Bugiotti, L. Cabibbo, R. Torlone, Data modeling in the NoSQL world, Computer Standards & Interfaces 67 (2020) 103149.

[14] E. M. Kuszera, L. M. Peres, M. D. D. Fabro, Exploring data structure alternatives in the RDB-to-NoSQL document store conversion process, Information Systems 105 (2022) 101941.

[15] N. Roy-Hubara, A. Sturm, P. Shoval, Designing NoSQL databases based on multiple requirement views, Data & Knowledge Engineering 145 (2023) 102149.

[16] U. Störl, T. Hauf, M. Klettke, S. Scherzinger, Schemaless NoSQL data stores – object-NoSQL mappers to the rescue?, in: Proc. BTW, 2015, pp. 579–599.

[17] S. Scherzinger, A. Thor, AutoShard – declaratively managing hot spot data objects in NoSQL document stores, CoRR abs/2111.01086 (2021).

[18] T. Taipalus, SQL: A trojan horse hiding a decathlon of complexities, in: Proc. DataEd, 2023, pp. 9–13.

[19] J. Yang, A. Gilad, Y. Hu, H. Meng, Z. Miao, S. Roy, K. Stephens-Martinez, What teaching databases taught us about researching databases: Extended talk abstract, in: Proc. DataEd, 2024, pp. 1–6.

[20] R. Alkhabaz, Z. Li, S. Yang, A. Alawini, Student's learning challenges with relational, document, and graph query languages, in: Proc. DataEd, 2023, pp. 30–36.

[21] M. Lopez, S. Ferrada, A. Hogan, ERDoc: A web interface for entity-relation modelling, in: Proc. DataEd, 2024, pp. 7–12.

[22] T. M. Connolly, C. E. Begg, A constructivist-based approach to teaching database analysis and design, Journal of Information Systems Education (2005) 43–53.

[23] C. Köhnen, U. Heuer, J. Zumbrägel, S. Scherzinger, Making a case for visual feedback in teaching database schema normalization, in: Proc. DataEd, 2025, pp. 11–16.

[24] N. Tripathi, NoSQL database education: A review of models, tools, and teaching methods, Journal

of Systems and Software 226 (2025) 112391.
[25] S. Mohan, Teaching NoSQL databases to undergraduate students: A novel approach, in: Proc. SIGCSE, 2018, pp. 314–319.
[26] S. Kim, Seamless integration of NoSQL class into the database curriculum, in: Proc. ITiCSE, 2020, pp. 314–320.
[27] A. Alawini, P. Rao, L. Zhou, L. Kang, P.-C. Ho, Teaching data models with TRIQL, in: Proc. DataEd, 2022, pp. 16–21.
[28] V. Meyer, L. Wiese, A. I. A. Al-Ghezi, Analyzing student feedback to assess NoSQL education, in: Proc. IDEAS, Springer, 2025, pp. 211–223.
[29] V. Meyer, L. Wiese, A. I. A. Al-Ghezi, A unified teaching platform for (No)SQL databases, in: Proc. ICEIS, 2024, pp. 374–381.