# Development of a visual tool for the design of aggregate-oriented NoSQL databases.

Antonio Sarasa-Cabezuelo
Dpto. Sistemas Informáticos y Computación .*Facultad de Informática.*
*Universidad Complutense de Madrid*
Madrid, Spain
asarasa@ucm.es

2021 25th International Conference Information Visualisation (IV) | 978-1-6654-3827-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IV53921.2021.00017

***Abstract*—The aggregation model is the foundation of some of the main NoSQL databases. This model is characterized because the basic element of management is the concept of aggregate. Furthermore, another characteristic is that it is not necessary to create a definition schema in the database to store information as in the case of relational databases. It is for this reason that in general there are no sophisticated design tools for databases oriented towards aggregates since the schema is implicit in the stored information. However, a design tool allows better management of the information to be stored and especially the maintenance of the database. This article presents a visual tool that allows you to design a NoSQL database oriented towards aggregates, as well as manage different designs. On the other hand, the tool allows generating instances of the design containing data for a MongoDB-type document database.**

***Keywords—NoSQL database, visual design tool, aggregation model, web application.***

## I. INTRODUCTION

In relational databases, a common task is design. This basically consists of defining a conceptual schema using an Entity-Relationship diagram or a UML class diagram [10], and its subsequent transformation into an SQL definition schema [10]. All these schematics and diagrams are based on visual languages [10] that offer graphic elements to represent the semantics of the design. Based on these languages, visual relational database design tools have been developed that offer intuitive graphical interfaces [8] where the designer can represent the design, and perform other operations such as its export to different formats, or its connection to a relational database so that the design is implemented directly in the database in the form of sets of tables [5].

In the last decade, new information persistence models have emerged, generically called NoSQL databases [15], which present important differences with the relational model. Among the NoSQL databases, those that are based on the so-called aggregation model stand out. This model is characterized by using the concept of aggregate as a unit of persistence [16]. In this sense, it represents the minimum portion of information that can be managed and on which an atomic manipulation is guaranteed. Another general characteristic of NoSQL databases is that they do not require a definition schema to store information [17]. The schema is implicit in the information that is stored. This relaxes the rigidity of the relation databases where the information must verify what is specified in the schema in order to be stored. Thus, in a NoSQL database it is possible to store information with different structures that have not been previously defined. However, this flexibility introduces other maintenance problems [1] such as maintaining the consistency of the database that was previously carried out by the database management system and now must be carried out by the programmer, or the maintenance of the structure of the stored information [6] that was previously reflected in the definition schema and is now implicitly specified in the stored information. It is for this reason regarding the need to define a definition schema that the use or existence of design tools for NoSQL databases is not very popular. However, although it is not necessary to define a definition scheme to use this type of database, it is interesting to carry out conceptual designs [12] in the same style as that used in the relational model in order to maintain the information and from the database.

The most widespread aggregate-oriented databases are documentaries, and in particular the MongoDB database. That is why the main initiatives for visual design tools for NoSQL databases are those that are designed for MongoDB. These include Variety Schema Analyzer, MongoDB Schema Validation, and MongoDB Compass. The main functionality of the Variety Schema Analyzer tool [7] is given a collection, analyze it to define a data organization scheme. As a result, it shows the occurrences of all the keys other than the documents as well as summary information about the type of data that it has found associated with the key, the number of times it has been found, and the percentage of documents in which it occurs. The MongoDB Schema Validation tool [4] allows to create validation schemes in JSON format, so that the database administrator can decide on a standardized format for all the documents that are inserted into a certain collection. Validation schemes can be created at the same time as a new collection, or applied to an existing collection afterwards. In the first case, different degrees of tolerance for compliance with the schema can be specified: strict (if a document violates the collection schema, an error is thrown and the document is rejected) or moderate (if the document violates the schema, it is displayed a warning to inform that this document will not be "normalized" but it is allowed to insert it). In the second case, when an scheme is applied to an existing collection with already inserted documents, a degree of tolerance can also be defined. Thus, if it is moderate, the scheme will only be applied a posteriori (exclusively to documents inserted from that moment). And if it is configured in strict level, all the documents inserted so far must comply with the schema. Finally, MongoDB Compass [20], is a desktop application that acts as a MongoDB graphical interface that allows to visually perform the operations that the user would execute in a MongoDB terminal (create collections and indexes, insert and edit documents or make inquiries). Among the options is also document validation.

This article describes a proposal for a visual aggregate-oriented NoSQL database design tool. For this, the same implementation ideas of the relational database design tools have been used. Thus, a visual metaphor is provided to design the aggregates that the database will manage, and a series of services will be provided to manage the created models, such

978-1-6654-3827-8/21/$31.00 ©2021 IEEE
DOI 10.1109/IV53921.2021.00017

as the association of data or the export of the models to specific instances of a aggregated-oriented database such as MongoDB . All of this has been implemented as a web application that offers a very intuitive and easy-to-use graphical interface.

The structure of the paper is as follows. In section II an introduction to aggregate-oriented databases is done. Next, section III describes in detail the functionalities of the developed application. Finally, in section IV the conclusions and a set of lines of future work are presented.

## II. Databases Aggregates-Oriented

The aggregate-oriented databases are made up of several families of NoSQL databases [9] (documentary, key-value and column-oriented) that share the model on which they are based, called the aggregation model [14]. This model defines the unit of persistence as the aggregate. It is a complex data structure that is managed as a unit in terms of its manipulation and consistency, that is, aggregates will be updated with atomic operations and it will interact with the database in terms of aggregates [11]. This model facilitates the execution of a database in a distributed system of clusters since the unit of distribution and replication will be the aggregates. It also solves the so-called impedance problem [2] (the difference that exists between data structures managed by programming languages and databases).

Aggregates are conceptualized differently according to the type of aggregate-oriented database family [18]. Thus, in documentary databases, an aggregate is a document, in column-oriented databases, an aggregate is a family of columns, and in key-value databases, an aggregate is the value that is stored (whose structure information cannot be accessed directly by the database).

The most widespread aggregate-oriented databases are documental. It is for this reason that the current version of the tool described in this article allows the design of generic aggregates, but it is only possible to define instances towards documentary databases [3]. As mentioned, in these databases, the aggregates are materialized in the form of documents. A document is a structure that stores related information of different types [19] (its conceptualization is similar to the data structure of the registry defined by most programming languages). Documents can modify their structure and the information they contain, and they can be consulted using a specific query language. The types of documents are grouped under the concept of collection, and a set of collections that works like a database.

From a design point of view, an aggregate can be conceptualized [13] as a set of hierarchical information where there are complex information fields that contain other information fields and simple information fields that contain basic data. This structure can be physically represented in various ways such as json, xml and others (For example in MongoDB a format called BSON based on json is used).

## III. A Modeling Visual Language

The objective of the project was to offer a visual modeling tool, and for this a language has been defined that allows to represent the design details of an aggregate. For this, a metaphor based on the inclusion of nodes within nodes is used that represents the composition relationships of the information fields of an aggregate. In this metaphor, an information field is represented by a node that has a name associated with it, and that can be formed by a basic data that cannot be decomposed further or by a set of one or more nodes drawn at the same level that represent one or more more compound or simple information fields.

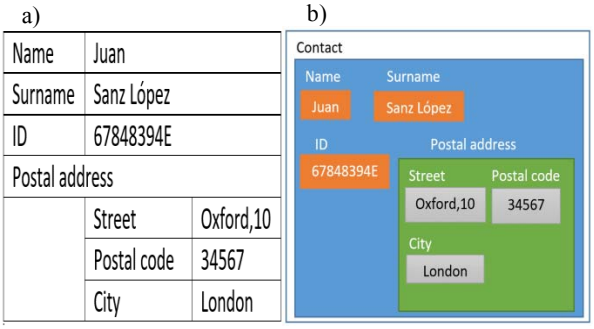


Fig. 1. a) Example of Aggregate, b) Visual representation of the aggregate

An example of an aggregate representing the information of a contact is shown in Figure 1.a. In the main aggregate, there are a set of simple information fields: Name, Surname, ID, Street, Postal code and City. Also, there is a compound information field: Postal addres. Its visual representation is shown in figure 1.b. This consists of a node that represents the entire aggregate "contact" which is made up of 4 nodes. On the one hand, there are the simple nodes Name, Surname and ID represented at the same level. And on the other hand, we have the Postal address node. It is a compound node that is made up of 3 simple nodes: Street, Postal code and City. Each simple node has a basic data associated with it that cannot be decomposed further.

## IV. Functionality

In this section the functionalities of the developed application will be shown.

### *A. Access to the application and registration.*

Access to the functionality is done from the application's home page (Fig. 2) by means of a username and password, for which it is necessary to have previously registered. In this sense, the home page shows a link to authenticate and a link to register.

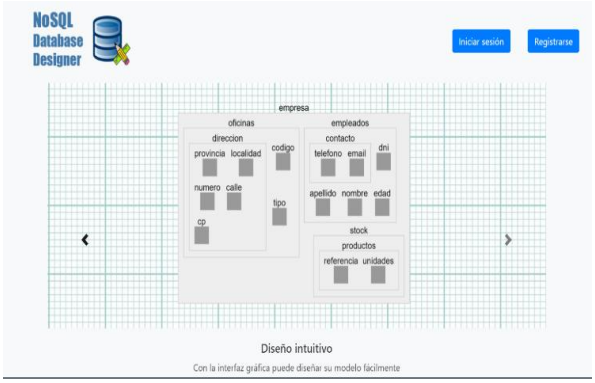


Fig. 2. Home page of the application

If you click on the registration link, a form is displayed (Fig. 3.a) where the user must enter the following personal information: name, surname, email and password.

Once registered, the user can access the application's functionality by clicking on the login link on the main page, showing a page (Fig. 3.b) where it must be entered the username and password.

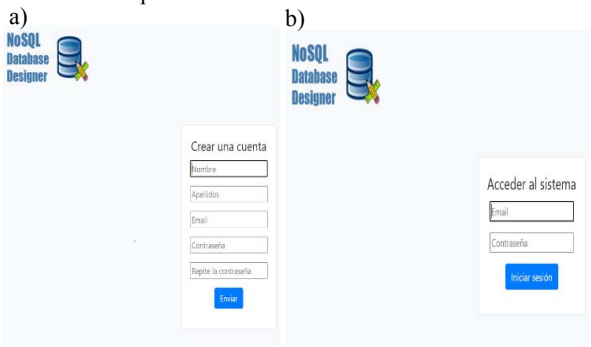


Fig. 3. a) Register page, b) Login page

## B. *Page "My models"("Mis modelos")*

When a user has been authenticated, it is shown a personal page called "My models" (Fig. 4) where the user can start the creation of a new model or query the models that it has been already created.


Fig. 4. "My models" page

If the option to create a new model is chosen, a window opens (Fig. 5) asking for a name for the model. Once created, the designer page is displayed, and the new model appears in the list of models with unsaved changes.

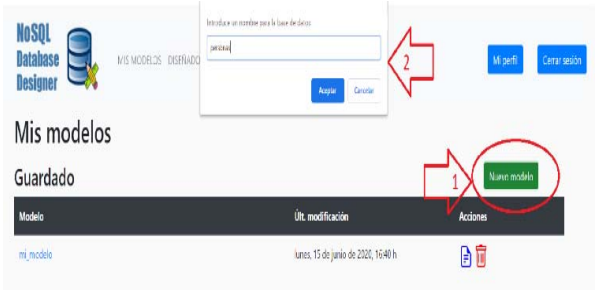


Fig. 5. Creation a new model.

The following options can be executed on a model:

- Discard changes (Fig. 6): This option closes the model without saving the changes, after confirmation. If the model was just created (not saved yet), the entire model will be deleted. On the contrary, if a model created at another time was being edited (it was already saved), only changes done in the current session are discarded.

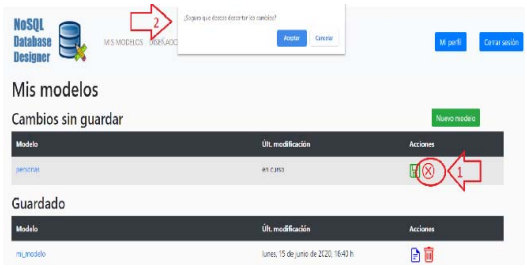


Fig. 6. Discard changes option.

- Save and close (Fig. 7): This option saves the last changes made to the model and closes it to be able to edit another one.


Fig. 7. Save and close option

On the other hand, the following actions can be executed on the saved models:

- Rename (Fig. 8): This option allows you to change the name assigned to a model.

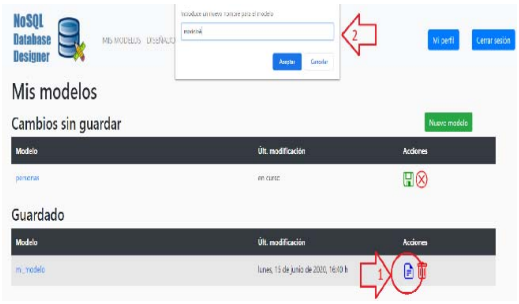


Fig. 8. Rename option

- Delete model (Fig. 9): This option deletes the model and all its associated information from the database, without the possibility of recovery. When the user executes this action, confirmation is first requested. If the answer is affirmative, the deletion is carried out. Otherwise, nothing is changed. Finally, it is redirected to the list of models to show it updated.


Fig. 9. Delete model option

*C. Designer*

The designer (Fig. 10) provides a graphical design interface so that the user can create models intuitively. The designer consists of a canvas where the graphical representation of the model is shown and a button panel that provides access to the various actions that can be executed on the model.

As explained in the previous section, the design of an aggregate-oriented database consists of defining the aggregates that it will manage. In this sense, the tool visually represents the aggregates as nodes and the composition relationships between aggregates as the composition of nodes. Thus, if an aggregate A is a subagregate of another aggregate B, the node that represents A will be drawn within the node that represents B. In addition, the direct subagregates of an aggregate will be drawn as a set of nodes that are at the same composition level.

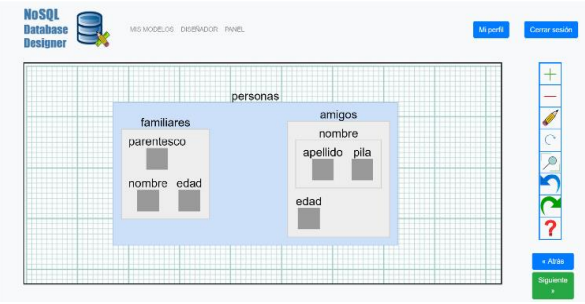


Fig. 10. Designer

For example, in the people ("Personas") aggregate in figure 10, it is made up of two simple sub-aggregates: family and friends. On the other hand, the family ("Familiares") aggregate is made up of 3 simple sub-aggregates: kinship ("parentesco"), name("nombre") and age ("edad"). While the friends ("amigos") aggregate is made up of 2 simple sub-aggregates: name ("nombre") and age ("edad"). Finally, the name ("nombre") aggregate is made up of two direct subagregates: surname ("apellidos") and first-name ("nombre-pila"). Aggregates that do not have subagregates are considered information fields that will have some basic value associated with it that cannot be decomposed.

When modeling begins, an initial node representing the model, with the name associated with it, always appears on the designer canvas. Within this aggregate the other subagregates will be added. To modify a node, it is necessary to select it and clicking on the desired action from the designer button panel.

The button panel is made up of a set of buttons that allow the following actions to be carried out on the model:

- Add aggregate (Fig. 11). To add a node, first it must select the container node, and then click on the "Add item" button. When this button is pressed, a dialog box appears requesting the name of the new aggregate, and then a new node is created with the name entered and whose parent is the previously selected node. The first level nodes (those that are children of the node that represents the model) must necessarily aggregates that contain other aggregates. They cannot be information fields since the first level nodes represent a collection of aggregates. The aggregates within the first-level aggregates can be both aggregates and information fields.

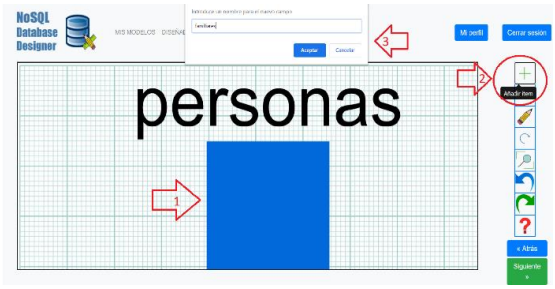


Fig. 11. Add aggregate option

- Remove aggregate (Fig. 12). To delete a node, it must be selected by clicking and pressing the delete button. If the node has associated information, the operation must be confirmed, as the data will be erased without the possibility of recovery. Otherwise, the user is not asked for confirmation, since the action can be reversed with the undo button or by recreating that node

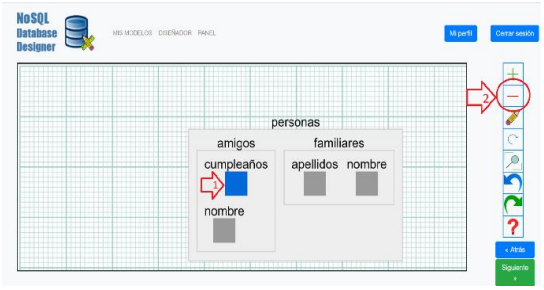


Fig. 12. Remove aggregate option

- Edit aggregate (Fig. 13). This option allows to change the name of a field or an aggregate. In order to do this, it is selected the node and click on the "edit item" button. A dialog box will open for entering the new name. Note that the name of the outermost node that represents the model cannot be edited. To do this, it must be done it from the list of models.


Fig. 13. Edit aggregate

- Restart canvas. (Fig. 14). In order to discard the model to start designing a new one, it is not neccesary to remove it or remove all nodes manually. To do this, it must be clicked on the «restart canvas» button, which allows the user to empty all the nodes added to the model, returning to the state it presented when it was created. It will be necessary to confirm the operation if

the model already contains data. This button does not require selecting any node to run.

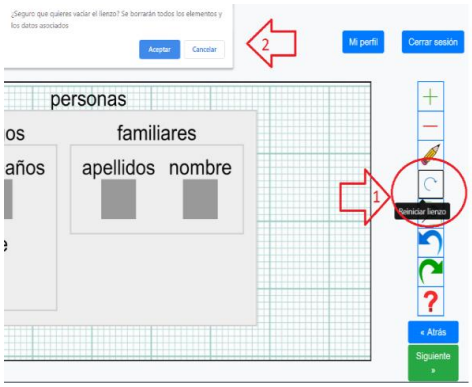


Fig. 14. Restart canvas option

- Adjust (Fig. 15). The view of the model can be enlarged and reduced by scrolling with the mouse. In order to return to the original size (100%), it must be clicked on the adjust button. This button also does not require a previous selection.


Fig. 15. Adjust option

- Redo (Fig. 16). Changes done to a model can be undone by clicking on the «undo» button. Similarly, undone changes can be redone with the "redo" button.

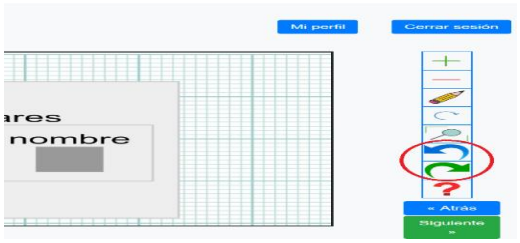


Fig. 16. Redo option

- Help. The help button allows to query a tutorial on how to use the tool.


Fig. 17. Help option

### *D. Data entry and export*

Once the model has been designed, data can be inserted into it by pressing the Next (“Siguiente”) button, located in the lower right part of the designer page. Then a page (Fig. 18) is shown with a form that has the same structure as the designed model. To insert data into the model, it must be filled in the form fields and clicked on insert(“Insertar”) button. If the form is filled in with wrong data and it is finally decided not to insert, all the fields of the form can be emptied with the reset (“Restablecer”) button.

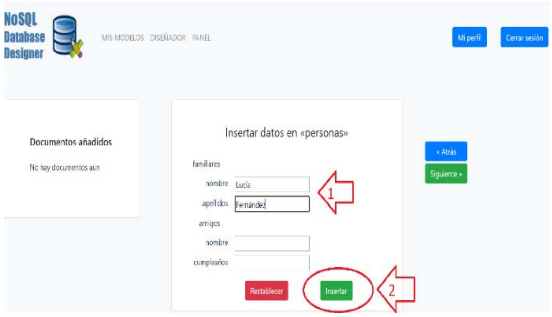


Fig. 18. Data entry page

The inserted documents can be consulted by clicking on the corresponding link in the left panel of the page. Once a document has been opened, it can be modified by modifying the necessary fields and pressing save (“Guardar”). From this view (Fig. 19), also it is possible to delete the document by pressing delete (“Eliminar”).

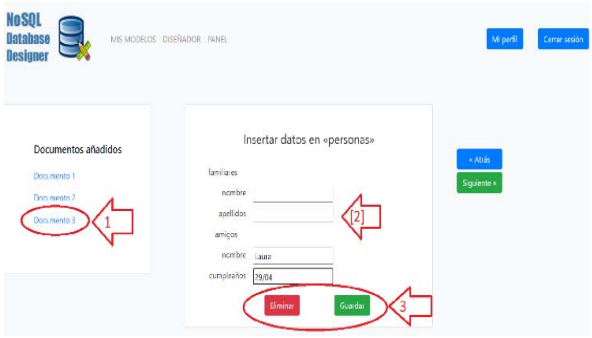


Fig. 19. Modify document option

When no further changes are to be done to the model or new information added, then the results can be saved and exported. From the data insertion page, click on the Next (“Siguiente”) link and the Save and export (“Guardar y exporter”) page (Fig. 20) is displayed. From this page it is possible to save the changes to the model (by clicking on the save (“Guardar”) button), as well as export the results in the form of code for MongoDB, which can be easily copied using a button enabled for this purpose.


Fig. 20. Save and export option

## V. Conclusions and future work

This article has presented a visual tool for aggregate-oriented NoSQL database design and modeling. For this, a very simple visual language has been implemented that is based on a graphic metaphor where the aggregates are represented as circular nodes and the information aggregation relationships are represented as the composition of circular nodes. Thus, a node within another node represents an information field that is part of another, more external information field. In this sense, in the representations there are container nodes for nodes and container nodes for values of basic types that cannot be decomposed into other information fields. The application allows the management of more than one aggregation model. It also offers the possibility of using the created model to create instances of the aggregates with data entered by the user. These instances can be exported to queries of a MongoDB-type document database. One of the advantages of the tool created is that the model created is neutral to the type of database oriented towards aggregates, for this reason, it is possible to export the instances created to queries to create aggregates of other types of databases as column-oriented or key-value. The main disadvantage is that the editor is not connected to any database so that it is not possible to store the instances created directly in any database.

This development consists of a first version of the tool, however some future lines of work are open:

- Export of aggregate instances to other aggregate-oriented NoSQL databases such as key-value type (for example Redis) or column-oriented (for example Cassandra).
- Create an option to connect the tool to a particular database where the aggregate instances created can be stored.
- Implement an option to reverse engineer so that, given a set of aggregate instances, an abstract model can be created appropriate to the structure of the instances.
- Add a visual query designer that allows the user to create queries on the developed models without having knowledge of the specific query language.
- Improve the graphic interface of the designer with other elements such as templates already created from aggregates.
- Implement an edition page where the user can model an entity-relationship diagram of the persistence problem to solve, and establish mechanisms for transforming the entity-relationship model to a model based on aggregates.
- Develop a data export format that allows representing the created model in a json or xml data format, so that it can be stored in a file outside the tool environment.

## Acknowledgment

I would like to thank Rubén Mendez Alarcón for implementing the system described in the article.

## References

[1] Abdelhédi, F., Brahim, A. A., Ferhat, R. T., & Zurfluh, G. (2020). Discovering of a Conceptual Model from a NoSQL Database. In ICEIS (1) (pp. 61-72).

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

[3] Baazizi, M. A., Colazzo, D., Ghelli, G., & Sartiani, C. (2019, June). Schemas and types for JSON data: from theory to practice. In Proceedings of the 2019 International Conference on Management of Data (pp. 2060-2063).

[4] Chellappan, S., & Ganesan, D. (2020). MongoDB Features and Installation. In MongoDB Recipes (pp. 1-24). Apress, Berkeley, CA.

[5] Daniel, G., Sunyé, G., Benelallam, A., Tisi, M., Vernageau, Y., Gómez, A., & Cabot, J. (2017). NeoEMF: A multi-database model persistence framework for very large models. Science of Computer Programming, 149, 9-14.

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

[7] Hernández, A., Feliciano, S., Sevilla, D., & García Molina, J. (2017). Exploring the visualization of schemas for aggregate-oriented NoSQL databases. In 36th International Conference on Conceptual Modelling (ER) ER Forum (pp. 72-85).

[8] Kalogirou, V., & Boehm, J. (2017, January). Interfacing NoSQL with open-source GIS. In Geographical Information Science Research UK Conference, Manchester, UK, April (pp. 18-21).

[9] Kaur, K., & Rani, R. (2013, October). Modeling and querying data in NoSQL databases. In 2013 IEEE International Conference on Big Data (pp. 1-7). IEEE.

[10] Khasawneh, T. N., AL-Sahlee, M. H., & Safia, A. A. (2020, April). SQL, NewSQL, and NOSQL Databases: A Comparative Survey. In 2020 11th International Conference on Information and Communication Systems (ICICS) (pp. 013-021). IEEE.

[11] Matallah, H., Belalem, G., & Bouamrane, K. (2020). Evaluation of NoSQL databases: MongoDB, Cassandra, HBase, Redis, Couchbase, OrientDB. International Journal of Software Science and Computational Intelligence (IJSSCI), 12(4), 71-91.

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

[13] Ruiz, D. S., Morales, S. F., & Molina, J. G. (2015, October). Inferring versioned schemas from NoSQL databases and its applications. In International Conference on Conceptual Modeling (pp. 467-480). Springer, Cham.

[14] Sadalage, P. J., & Fowler, M. (2013). NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education.

[15] Sarasa-Cabezuelo, A. (2016). Introduccion a las Bases de Datos NoSQL usando MongoDB. Editorial UOC.

[16] Sarasa-Cabezuelo, A. (2019). The Role of NonSQL Databases in Big Data. Smart Data: State-of-the-Art Perspectives in Computing and Applications. CRC Press

[17] Sarasa-Cabezuelo, A. (2021). New Trends in Databases to NonSQL Databases. In Encyclopedia of Information Science and Technology, Fifth Edition (pp. 791-799). IGI Global.

[18] Störl, U., Klettke, M., & Scherzinger, S. (2020). NoSQL Schema Evolution and Data Migration: State-of-the-Art and Opportunities. In EDBT (pp. 655-658).

[19] Vera, H., Boaventura, W., Holanda, M., Guimaraes, V., & Hondo, F. (2015, September). Data modeling for NoSQL document-oriented databases. In CEUR Workshop Proceedings (Vol. 1478, pp. 129-135).

[20] Vokorokos, L., Uchnár, M., & Baláž, A. (2017, October). MongoDB scheme analysis. In 2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES) (pp. 000067-000070). IEEE.