# Query Completion for Small-Scale Distributed Databases in PostgreSQL and MongoDB

Marin FOTACHE[1], Cătălina BADEA[2], Marius-Iulian CLUCI[3], Codrin-Ștefan EȘANU[4]
[1]Dept.of Accounting, Information Systems and Statistics
Alexandru Ioan Cuza University of Iaşi, Faculty of Economics and Business Administration
[2]Alexandru Ioan Cuza University of Iaşi
[3] Alexandru Ioan Cuza University of Iaşi, S.C. HVM SRL
[4] Alexandru Ioan Cuza University of Iaşi, Cegeka Romania
Iaşi, Romania
fotache@uaic.ro, catalina.badea@student.uaic.ro, marius.cluci@uaic.ro, codrin.stefan@feaa.uaic.ro

*Relational/SQL and document/JSON data stores are competing but also complementary technologies in the OLAP (On-Line Analytical Processing) systems. Whereas the traditional approaches for performance comparison use the duration of queries performing similar tasks, in this paper we compare the performance of two distributed setups deployed on PostgreSQL/Citus and MongoDB by focusing only on the query's successful completion within a 10-minute timeout. The TPC-H benchmark database was converted into a denormalized JSON schema in MongoDB. An initial set of 296 SQL queries was devised for execution in PostgreSQL/Citus and then mapped for execution in MongoDB using Aggregation Framework (AF). Query execution success within a 10-minute timeout was collected for both PostgreSQL and MongoDB in six scenarios defined by two small-scale data factors (0.01 and 0.1 GB) and three different node counts (3, 6, and 9) for data distribution and processing. The relationships between the query completion and the query parameters were assessed with statistical tests and a series of machine learning techniques.*

***Keywords:*** *PostgreSQL, Citus, MongoDB, SQL, Aggregation Framework, OLAP performance comparison.*

## 1 Introduction

Current data architectures incorporate various technologies for data storage and processing, such as relational/SQL and NoSQL data servers [1][2][3]. In this paper, we analyse and compare the OLAP [4] performance of PostgreSQL and MongoDB on six distributed configurations deployed on OpenStack [5], by varying the database size and the number of data distribution nodes.

For this purpose, we used the TPC-H Benchmark [6] database and tools, to populate the TPC-H database tables with a specified volume of data. We started with the original TPC-H schema, devising a module to populate a corresponding MongoDB schema by transforming (mainly by nesting) the data structure. This transformation allows a MongoDB collection to integrate data originally scattered across two or more tables.

We developed a set of 296 SQL queries (for the original TPC-H schema) that vary in terms of number of joins, filter clauses, or group clauses. Subsequently, this query set was translated into MongoDB's Aggregation Framework (MAF) and adapted to the new document database structure. In this new structure, some of the joins between collections become unnecessary, as the data is denormalized and stored as nested arrays inside documents.

The queries were run on both PostgreSQL and MongoDB setups across six scenarios. These scenarios were defined by the combination of two small-scale factors (of 0.01GB and 0.1GB) and three distribution architectures with 3, 6, and 9 nodes. A 10-

minute timeout was imposed for the execution of each query. Queries that did not complete within 600 seconds were canceled.

The preliminary results on the relationship between query completion (for both database servers) and the database size and the distribution setup were presented in [6]. In this paper, we examine the relationships between the query completion and some parameters describing the query complexity; we also developed, tuned, and interpreted Machine Learning (ML) classification models for predicting query completion on various predictors about database size, distribution setup and query complexity.

The remaining of this paper is organized as follows: section 2 examines the results of some previous studies approaching the performance comparison of SQL (PostgreSQL) and document (MongoDB) data stores. Section 3 provides a brief overview of the distributed architectures of Citus (a distributed PostgreSQL solution) and MongoDB. Section 4 describes the experimental setup, data analysis methods and tools used for obtaining the results. Results are analyzed in section 5. The paper concludes with the main findings, pointing out the study's main limitations, and some future directions for research.

## 2 Previous studies on OLAP performance comparison of SQL and NoSQL database servers

Both PostgreSQL and MongoDB are highly popular on the database market [7] and a considerable body of literature targeted their OLTP (On-Line Transaction Processing) and/or OLAP performance comparison. Results are far from convergent, as seen below.

In [8] PostgreSQL outperformed MongoDB. Güney and Ceylan [9] found that MongoDB had certain advantages in specific cases, particularly in data sorting operations, while PostgreSQL performed better in bulk data extraction and more complex query operations. Villalobos et al. [10] found that PostgreSQL performs more efficiently with complex queries involving intersection or combination functions, while MongoDB is better suited for simpler queries involving only filtering of descriptive, non-geographic data. In [11] MongoDB excelled in managing national election data, with consistent execution times regardless of operation size. PostgreSQL also performed well, with memory usage increasing linearly with the size of the operation. Also, in [12] MongoDB significantly outperformed SQL Server in processing e-government data.

Yedilkhan et al. [13] showed that both MongoDB and PostgreSQL are suitable for particular use cases and scenarios. The comparison was based on performance evaluations conducted across multiple clusters, including both cloud and on-premises environments. In [14] MongoDB outperformed PostgreSQL in terms of latency. Makris et al. [15] concluded that PostgreSQL performed better than MongoDB when managing spatio–temporal data. In [16] results showed that relational databases are more efficient than non-relational databases when executing the basic types of database operations (CRUD – Create, Read, Update, Delete).

Tracz and Plechawska-Wójcik [17] analyzed the performance of relational and non-relational databases using MS SQL Server, MongoDB, and CouchDB. A total of 30 test series were conducted for each database server across the scenarios of interest, using the following record counts: 500, 1000, 2000, 5000, and 10000; results revealed that SQL Server ranks first, followed by MongoDB in the second place, and CouchDB as the least efficient of the three. In [18] results indicated that NoSQL databases are a better alternative to their relational counterparts. Setyawan et al. [19] found that MongoDB is well-suited for handling large volumes of data, while MySQL recorded optimal performance for queries executed on a smaller scale.

Figueiredo et al. [20] showed that PostgreSQL manages better data that is well structured, due to its robust support for complex queries. In their study, a 10 GB dataset was processed by each database server, with three types of queries: 1st, 5th and 6th (from the TPC-H benchmark query-set) selected for performance evaluation. Two testing scenarios were employed, one involving approximately 750 MB of data inserted into PostgreSQL and MongoDB, and another with a 10 MB file inserted into HarperDB. On the response time, PostgreSQL recorded the best database performance.

Sals et al. (2023) [21] argued that MongoDB outperforms MySQL in data storage and processing capabilities. Moreover, Antas et al. [22] concluded that MongoDB has better results for almost all large data volume tests; similarly, in [23] MongoDB performed better for CRUD operations. Abukabar et al. [24] designed an evaluation benchmark and identified that MongoDB recorded superior performance in handling CRUD operations, while MySQL excels in executing stored procedures. Matallah et al. [25] found that MongoDB recorded smaller execution time than MySQL for all types of query operations. In [26] the results showed that MongoDB had significantly higher throughput than both MySQL and PostgreSQL, particularly for small transaction sizes, indicating that MongoDB is a better choice for applications which involve heavy writing of indexed spatial data.

Reichardt et al. [27] compared the performance of NoSQL data stores, relative to their relational/SQL counterparts. They found that MongoDB is more suitable than MySQL for a wide range of dataset sizes. Naufal et al. [28] found MongoDB outperforms in the Update and Delete operations based on runtime differences.

## 3 Distributed architectures for PostgreSQL/Citus and MongoDB. Deployment on OpenStack

Citus is an open-source and easy-to-deploy PostgreSQL extension which enhances PostgreSQL by distributing the database using horizontal scaling and parallel query processing [29]. It utilizes multiple PostgreSQL instances to distribute both data and queries across the nodes (in this paper we deployed distributed setups with 3, 6 and 9 nodes). Thus, it provides efficient management of large datasets and high query volumes. In a Citus cluster (see Fig. 1), data is divided into shards and distributed across worker nodes. The master node handles the metadata for the distributed tables, coordinates query execution by delegating tasks to worker nodes, and aggregates the results. Worker nodes store the data shards and execute the queries [30].

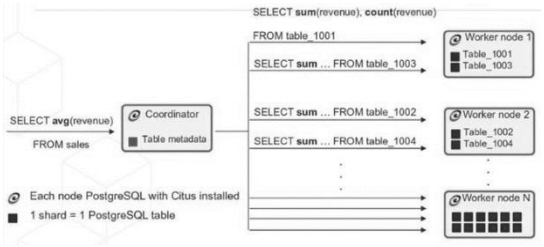


**Fig. 1.** Citus architecture ([29])

This architecture enables Citus to uphold the robustness and flexibility of PostgreSQL. At the same time, it delivers substantial performance improvements for OLAP workloads [31].

MongoDB is the most popular NoSQL product [7] which proved to be a reliable solution in providing database scalability, flexibility, and high performance. The database uses JSON-like documents with flexible schemas, which makes it easier to manage various data types and speeds up the development process.

Moreover, it employs a sharding architecture (Fig.2) to spread the processing load and storage across multiple servers. By horizontal scalability, MongoDB can handle large volumes of data with good overall performance in data processing. Data is split

into smaller, more manageable units known as shards, which can be hosted on separate servers or containers for better and easier management.

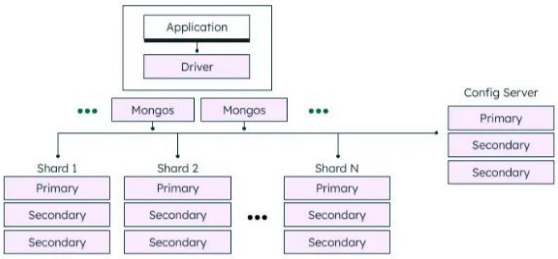


**Fig. 2.** MongoDB architecture ([32])

MongoDB distributes queries across these shards, allowing it to efficiently handle large data volumes and high traffic loads [32]. The data chunks are duplicated among shards to ensure high availability. Mongos instances route queries and manage data distribution, while config servers handle metadata and configuration settings.

MongoDB supports replication through replica sets, which further enhances data availability and fault tolerance in critical systems. Each replica set includes a primary node for writes and multiple secondary nodes that maintain copies of the data, these are in pairs of 3 machines in most cases. If failure occurs, one of the secondary nodes is automatically promoted to primary, ensuring minimal downtime and data consistency.

More details on Citus and MongoDB distributed setups are provided in section 4.3.

## 4 The experimental design. Research method and tools.

We created an initial set of 296 SQL queries (for the TPC-H benchmark database) in PostgreSQL and translate them into MongoDB Aggregation Framework queries on a JSON schema with nested arrays in collections (as described in section 4.1). The queries were executed multiple times on both servers, with a 600-second timeout, varying the database size (scale factor) and the data distribution (the number of nodes).

Whereas most of the previous studies presented in section 2 assessed the database performance with the query response times metric, our research focuses on the successful completion of queries within a timeout period of 10 minutes. In [6] the successful query completion was examined in relation to the database server and the nodes of data distribution using inferential statistics. In this paper we approached a series of research questions and sub-questions, also concerning the query successful completion, as follows:

• RQ1: Is the query completion/success associated with the parameters describing the query complexity?

  o RQ1a: Is query completion associated with the number of joins/lookups in the query?

  o RQ1b – Is query completion associated with the number of filters (predicates for record selection)?

  o RQ1c – Is query completion associated with the record grouping?

  o RQ1d – Is query completion associated with limiting the number of records in the result?

• RQ2: Can the query successful execution be predicted based on database size, physical setup and query complexity parameters?

  o RQ2a: Do Machine Learning classification models reliable predict the success of query execution based on the predictors related to the data volume, data distribution and query complexity?

  o RQ2b: Which are the most important variables for the prediction of query completion? How their distribution influences the query success?

### *4.1 Database structure in PostgreSQL and MongoDB*

Since our study is focused on the OLAP performance [33] of MongoDB and PostgreSQL, we relied on the TPC-H benchmark database. This benchmark evaluates database systems performance by

using a series of business-oriented queries (which we did not use in this paper).

In PostgreSQL the database followed the original TPC-H schema consisting of eight tables, as represented in Fig. 3. For details on the table attributes and constraints, see [34] and [6].

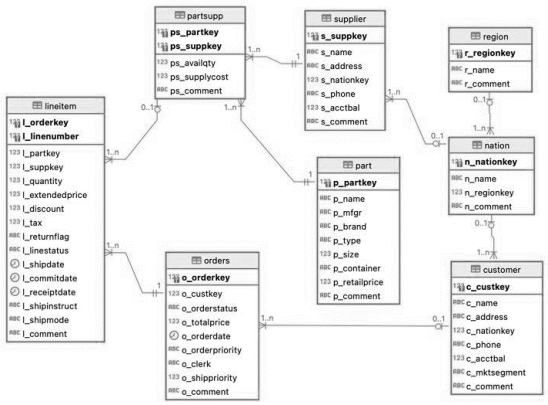


**Fig. 3.** TPC-H database schema ([34] [6])

When devising the corresponding MongoDB collections, we did not simply map the PostgreSQL tables into "flat" collections. Instead, we combined some tables through nesting and thus we incorporate some redundancy into the database. The resulting collections were:

- *customer_nation*
- *lineitem*
- *lineitem_orders*
- *lineitem_partsupp*
- *part*
- *partsupp*
- *region*
- *supplier_nation*.

Collections with simple names mirror the PostgreSQL tables directly, containing a straightforward JSON representation with the same attributes. The compounded collections, identified by names with underscores, merge data from two PostgreSQL tables. These collections are designed to reflect more complex relationships by nesting documents and arrays. As illustrated in Fig. 4, each document in the *lineitem_orders* collection represents an order from the *orders* table which includes (lines 7 to 29 in Fig. 4) an array of line items associated with that order, integrating data from both the *orders* and *lineitem* tables.

```
1  {   // this is the first document (describing the first order)
2      "o_orderkey": 1, "o_custkey": 370, "o_orderstatus": "O",
3      "o_totalprice": 172799.49,
4      "o_orderdate": {"$date": "1996-01-02T00:00:00.000-0700"},
5      "o_orderpriority": "5-LOW         ", "o_clerk": "Clerk#000000951",
6      "o_shippriority": 0, "o_comment": "nstructions sleep furiously among ",
7      "lineitem": [
8          { // this is the first line of the current order
9              "l_orderkey": 1, "l_partkey": 1552, "l_suppkey": 93,
10             "l_linenumber": 1, "l_quantity": 17.00, "l_extendedprice": 24710.35,
11             "l_discount": 0.04, "l_tax": 0.02, "l_returnflag": "N",
12             "l_linestatus": "O", "l_shipdate": {"$date": "1996-03-13T00:00:00.000-0700"},
13             "l_commitdate": {"$date": "1996-02-12T00:00:00.000-0700"},
14             "l_receiptdate": {"$date": "1996-03-22T00:00:00.000-0700"},
15             "l_shipinstruct": "DELIVER IN PERSON        ",
16             "l_shipmode": "TRUCK     ", "l_comment": "egular courts above the"
17         }
18         ,{  // this is the second line  of the current order
19             "l_orderkey": 1, "l_partkey": 674, "l_suppkey": 75,
20             "l_linenumber": 2, "l_quantity": 36.00, "l_extendedprice": 56688.12,
21             "l_discount": 0.09, "l_tax": 0.06, "l_returnflag": "N",
22             "l_linestatus": "O", "l_shipdate": {"$date": "1996-04-12T00:00:00.000-0700"},
23             "l_commitdate": {"$date": "1996-02-28T00:00:00.000-0700"},
24             "l_receiptdate": {"$date": "1996-04-20T00:00:00.000-0700"},
25             "l_shipinstruct": "TAKE BACK RETURN         ",
26             "l_shipmode": "MAIL      ", "l_comment": "ly final dependencies: slyly bold "
27         }
28         // ... the remaining lines of tge current order
29     ]
30 }
31 //, ... the remaining orders
```

**Fig. 4.** Structure of a document in collection *lineitem_orders* ([6])

In PostgreSQL the join is required when extracting data from both *lineitem* and *orders* tables; but in MongoDB's Aggregation Framework, this join (*$lookup* function) is not needed because all the relevant data is contained within a single collection.

In MongoDB the names of the collections which map two PostgreSQL tables are formatted with the child table's name on the left side of the underscore and the parent table's name on the right side. In collection *lineitem_orders*, *lineitem* is the child table and table *orders* is the parent. Each document in this collection corresponds to a record from the parent table (*orders*), and within each document, there is an array that stored all related records from the child table (*lineitem*).

### *4.2 SQL and Aggregation Framework Queries*

The initial 296-query set (written in the PostgreSQL dialect of SQL) was devised to ensure some variability on the parameters describing the query complexity, as contained in the SELECT, FROM, WHERE, GROUP BY, ORDER BY, LIMIT/FETCH and OFFSET/SKIP clauses. The query set

was converted into the syntax of the main declarative query language in MongoDB – the Aggregation Framework, considering the new (nested) JSON structure.

Table 1 illustrates two scenarios of mapping SQL queries to the Aggregation Framework language, considering the MongoDB's "nested" structure (using arrays). In SQL, the 187th query involves a join between *customer*, *orders*, *lineitem* and *partsupp* tables. For the same query in MongoDB, Aggregation Framework uses the *$lookup* method to achieve a similar effect by joining data from different collections. By contrast, the 277th query requires join only in SQL, because in Aggregation Framework there is no need for join since all query data are stored in a single collection, *customer_nation*.

**Table 1.** Two examples of query mapping

| Query/ Syntax | Query content |
|---|---|
| Q187 in SQL | select customer.c_name, orders.o_orderdate,<br>lineitem.l_quantity,partsupp.ps_availqty<br>from customer<br>join orders on customer.c_custkey = orders.o_custkey<br>join lineitem on lineitem.l_orderkey = orders.o_orderkey<br>join partsupp on lineitem.l_partkey = partsupp.ps_partkey<br>where l_quantity < 5 and l_tax > 0.05 and l_discount > 0.08 limit 800; |
| Q187 in AF | db.customer_nation.aggregate([{$unwind: ""$customer""},<br>{$lookup: {from:""lineitem_orders"",<br>localField:""customer.c_custkey"",<br>foreignField: ""o_custkey"", as: ""orders_1join""}},<br>{$unwind: ""$orders_1join""},<br>{$lookup: {from: ""lineitem"",localField: ""orders_1join.o_orderkey"",<br>foreignField: ""l_orderkey"", as: ""lineitem_2join""}},<br>{$match: {""lineitem_2join.l_quantity"": {$lt: 5},<br>""lineitem_2join.l_tax"": {$gt: 0.05},<br>""lineitem_2join.l_discount"": {$gt: 0.08}}},<br>{$lookup: {from: ""partsupp"",localField: ""lineitem_2join.ps_partkey"",<br>foreignField: ""l_partkey"",as: ""partsupp_3join""}},<br>{$project: { _id: 0, c_name:""$customer.c_name"",<br>o_orderdate:""$orders_1join.o_orderdate"", l_quantity:<br>""$lineitem_2join.l_quantity"",<br>ps_availqty: ""$partsupp_3join.ps_availqty""}}, {$limit: 800}]); |
| Q277 in SQL | select count(c_custkey), n_name<br>from customer<br>join nation on customer.c_nationkey = nation.n_nationkey<br>where c_acctbal > 0 and c_mktsegment = 'FURNITURE' and n_regionkey = 1<br>group by n_name; |
| Q277 in AF | db.customer_nation.aggregate([<br>{$unwind: "$customer"},<br>{$match: {"customer.c_acctbal": { $gt: 0 }}}, {$match:<br>{"customer.c_mktsegment": "FURNITURE "}}, {$match: {"n_regionkey":<br>1}},{$group: {_id:"$n_name",NumberOfCustomer: { $sum: 1 }}},<br>{$project: {_id: 0, n_name: "$_id",NumberOfCustomer:1}}<br>]); |

When translating the SQL queries into MongoDB Aggregation Framework queries for yielding the equivalent results, we strived to maintain simplicity in the syntax for both languages.

### *4.3 Physical Setup*

The experimental setup was deployed on the RaaS-IS platform, which operates as a private cloud managed by OpenStack, as described in [6]. The RaaS-IS datacentre

architecture includes 20 servers: 3 controller nodes to ensure high availability of OpenStack services, 16 compute nodes for Virtual Machine and container provisioning, and a management server responsible for MAAS, Juju, LDAP, and network management. The storage infrastructure uses an HPE 3PAR 8440 SAN with a total raw capacity of 760 TB, of which 550 TB is usable, and includes an 8 TB SSD cache for enhanced performance. The SAN's NL-SAS HDDs are set up in RAID 5, while the SSD cache uses RAID 6 to deliver higher performance. The compute nodes have dual Intel Xeon Gold 6240 CPUs (18 cores, 2.6 GHz each), 128 GB of RAM@2933MHz, and 300 GB SAS drives in RAID 1. Together, these provide 1152 virtual cores and 2 TB RAM for the RaaS-IS project and infrastructure.

*4.3.1 PostgreSQL Citus Cluster Setup*

The Citus cluster was configured and partially deployed using the OpenStack Heat orchestration module, which allows for Infrastructure as Code (IaC) using built-in OpenStack tools. The cluster was set up on virtual machines within the cluster nodes running a custom-configured Ubuntu Linux 18.04.6 LTS (kernel version 4.15.0-213-generic), along with TPC-H v3.0 and Citus v11.3.
To streamline cluster management, custom bash scripts were developed to enhance monitoring, result retrieval, and automated deployment.
The Citus cluster consisted of 10 machines, designed to provide distributed parallel query execution and scalability for analytical workloads. The cluster setup was as follows:

- Coordinator Node (1 machine): This node served as the primary entry point for queries, distributing them across worker nodes while maintaining metadata and query routing.
- Worker Nodes (9 machines): These nodes handled data storage and parallel execution of queries, improving performance for analytical workloads

*4.3.2 MongoDB Sharded Cluster setup*

The MongoDB cluster was configured and partially configured using the OpenStack Heat orchestration module. This module uses built-in OpenStack tools to allow Infrastructure as Code (IaC). The MongoDB cluster was set up on virtual machines within the compute nodes, using a custom-configured Ubuntu Linux 18.04.6 LTS (kernel version 4.15.0-213-generic), TPC-H v3.0, and MongoDB v6.0.14. A set of bash scripts was developed to improve monitoring and configuration for retrieving results and automating the cluster setup.
The MongoDB cluster comprised 13 machines configured to ensure optimal performance and reliability, mirroring a typical production environment.
The servers were organized as follows:

- Router Node (1 machine): This server directed client requests to the correct shard.
- Config Nodes (3 machines) – these nodes managed the cluster's metadata and configuration, ensuring smooth operation.
- Shards (9 machines): The data was spread across three shards, each with three replica sets. Each set included a primary node for write operations and two secondary nodes for redundancy and high availability. The number of shards could be adjusted depending on the test scenario.

Each machine was set up with 4 cores, 4GB of RAM, and a 200GB storage volume on the SAN.

***4.4 Variables (outcomes and predictors)***

Table 2 shows the variables used in the analysis. The first three variables define the physical setup of each scenario in the experiment. Variable *success* refers to the outcome of the query execution (whether it was completed within the 10-minute timeout).

**Table 2.** Variable description

| Variable | Description |
|---|---|
| db_server | name of the DBMS |
| *scale_factor* | size of the DBMS |
| *n_of_nodes* | number of nodes used for data distribution |
| *success* | binary variable whose value is TRUE when the query was successfully completed within the 10-minute timeout, and FALSE otherwise |
| *join* | number of JOIN clauses (PostgreSQL), equivalent to $lookup (MongoDB) |
| *where* | number of WHERE clauses (PostgreSQL), equivalent to $match (MongoDB) |
| *count* | number of COUNT clauses (PostgreSQL), equivalent to count: (MongoDB) |
| *group_by* | number of GROUP BY clauses (PostgreSQL), equivalent to $group (MongoDB) |
| *having* | number of HAVING clauses (PostgreSQL), equivalent to $match (MongoDB) |
| *limit* | number of LIMIT clauses (PostgreSQL), equivalent to $limit (MongoDB) |

The variable names related to the query complexity are inspired from the SQL syntax, and the second column points to the Aggregation Framework equivalent feature.

### *4.5 Method and tools for data analysis*

For RQ1a-RQ1c the analysis relied on classical inferential statisticss. For RQ2a and RQ2b we built, tuned and interpreted a series of ML classification models based on two of the most popular classification algorithms, Random Forest and Extreme Gradient Boosting (XGBoost) algorithms.
After the query results and parameters were collected and integrated, the dataset was examined using Exploratory Data Analysis. The Chi-square test of independence was applied to assess the association between nominal variables. We employed Random Forest and XGBoost to identify the most important predictors for the success of query execution.

The dataset was processed and analyzed using R programming language [35], employing tools from the *tidyverse* package collection [36] along with *ggstatsplot* [37]. The *tidyverse* suite is a set of R packages designed for data manipulation and visualization, while *ggstatsplot* and *rstatix* provide additional functionalities for statistical analysis and visualization of results. All models were built and tuned with the *tidymodels* ecosystem [38].
All the Interpretable ML techniques (Variable Importance, Partial Dependency Profiles, Individual Conditional Expectation, and Accumulated Local Effects Profiles) were deployed using the *DALEX* ecosystem ([39]).

## 5 Results and discussion

All 296 queries were run on both database servers across six different scenarios. The chart in Fig. 5 illustrates the distribution of query parameters, providing a glimpse into the queries complexity.

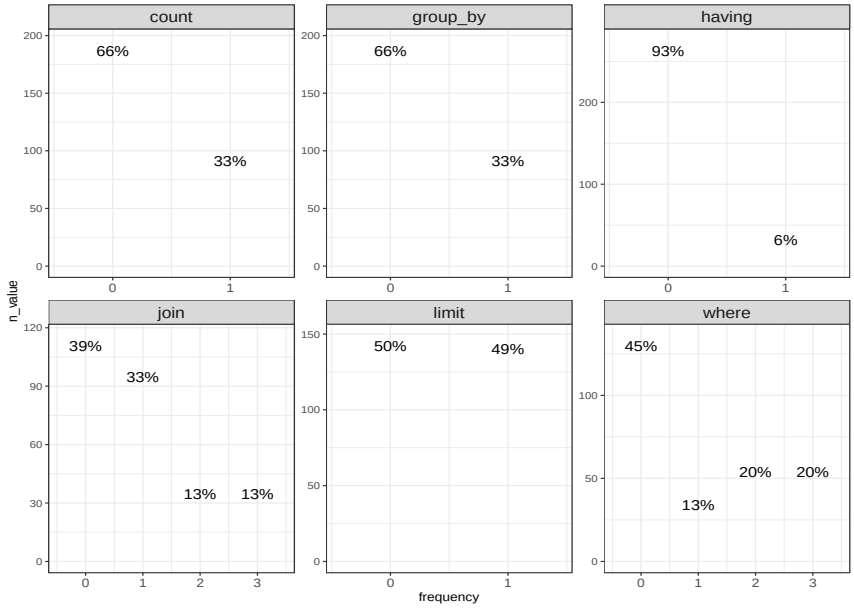


**Fig. 5.** Distribution of queryparameters

The chart reveals that 33% of the queries involved grouping records and performing aggregation by counting the records within each group, 45% had no filters applied, and roughly half of the queries imposed a limit on the result size.

As presented in [6], there is a correlation between query completion (within the 10-minute timeout) and the database server. Of the 3552 queries executed across both servers in all six scenarios, 315 failed to complete. Of these, 309 failed on MongoDB, while only six failed on PostgreSQL/Citus. Of the 3237 successful queries, 1770 (55%) were executed on Citus, with the remaining 1467 (45%) completed on MongoDB.

Also, [6] found that the relationship between query completion (within the 10-minute timeout) and the data distribution setup differs between database servers. In Citus, the proportion of successful queries remains consistent across the three data distribution scenarios. However, MongoDB shows more variability: the 6-node setup had the highest failure rate, while the 9-node setup performed the best.

The subsequent series of statistical tests concerned the associations between the successful query completion and some variables describing the query complexity. To start with *RQ1a: (Is query completion associated with the number of joins/lookups in the query?)*, Fig. 6 shows the results of the Chi-Square test of independence between variables *success* and *joins* for PostgreSQL/Citus (left) and MongoDB (right).

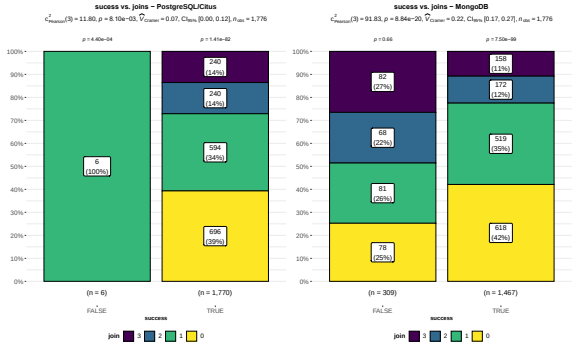


**Fig. 6.** Association query success vs. number of joins

In Citus, successful and unsuccessful queries recorded a similar average number of joins (1). By contrast, in MongoDB, the joins/lookups appear more costly since for the completed queries (1467), the average number of joins was 0.91, whereas, for the canceled queries (309), the average number of joins was 1.50.

For Citus, the Chi-square test of independence indicated a p-value below 0.05 and an effect size (*Cramer's V*) of 0.07,

providing evidence to state that the number of joins is significantly linked to the successful completion of queries within the 10-minute timeout. Meanwhile, for MongoDB, the Chi-square test computed a p-value less than 0.05 and a *Cramer's V* effect size of 0.22, which offer support to state that the number of joins is significantly related with the successful completion of queries within the 10-minute timeout.

Regarding *RQ1b (Is query completion associated with the number of filters /predicates for record selection?),* Fig.7 shows the results of the Chi-Square test of independence between variables *success* and *filters* for Citus (left) and MongoDB (right).

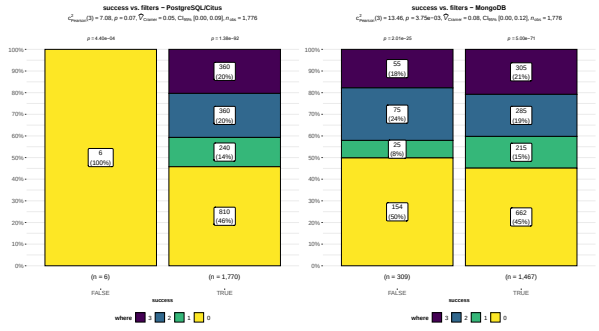


**Fig. 7.** Association success vs. number of filters

In Citus, for the successful queries (1770), the average number of filters was 1.14 and for the unsuccessful queries (6) the average number of filters was 0. In MongoDB, successful and unsuccessful queries recorded a similar average number of filters, ~ 1.

For Citus, the Chi-square test recorded a p-value greater than 0.05 and an effect size (*Cramer's V*) of 0.05, suggesting that the number of filters is not associated with the successful completion of queries. Similarly, for MongoDB, the Chi-square computed a p-value smaller than 0.05 and an effect size (Cramér's V) of 0.08, indicating that the number of predicates is not significantly associated with the successful completion of queries, just as observed in Citus.

Addressing the next research question, *RQ1c - Is query completion associated with grouping (aggregation) the records?*, Fig. 8 shows the findings of the Chi-Square test of independence between the variables *success* and *group_by* for both Citus (left) and MongoDB (right).

In Citus, the unsuccessful queries (6) included those that were executed without any record grouping. In the case of successful queries (1770), 66% were executed without record grouping, whereas 34% included it. In contrast, MongoDB experienced 309 unsuccessful queries, the majority of which did not feature record grouping, while out of 1467 successful queries, only 37% included record grouping.

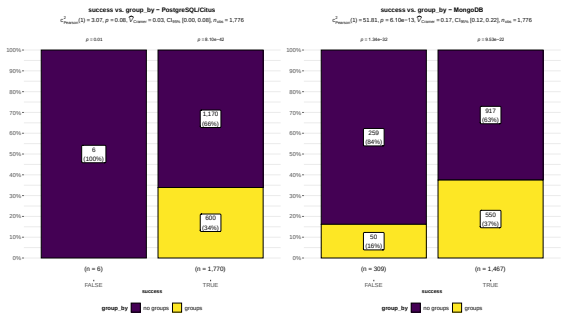


**Fig. 8.** Association success vs. record grouping

For Citus, the Chi-square test computed a p-value greater than 0.05 and an effect size (*Cramer's V)* of 0.03, which fail to support a significant association between record grouping and query completion. Conversely, for MongoDB, the Chi-square test of independence showed a p-value less than 0.05 and a higher effect size (0.17), which points to a strong association between record grouping and query completion.

Regarding the subsequent research question, *RQ1d – Is query completion associated with fixing a limit of records in the result?*, Fig. 9 presents the outcomes of the Chi-Square test of independence between variables *success* and *limit* for Citus (left) and MongoDB (right).

In Citus, the unsuccessful queries (6) were

those executed without limiting the result size, whereas for the successful queries (1770), more than a half of queries were executed without result size limitation. In contrast, MongoDB recorded 309 unsuccessful queries, the majority of which did not impose a limit on the result size. Among the 1467 successful queries, less than a half of the total applied a size limitation.

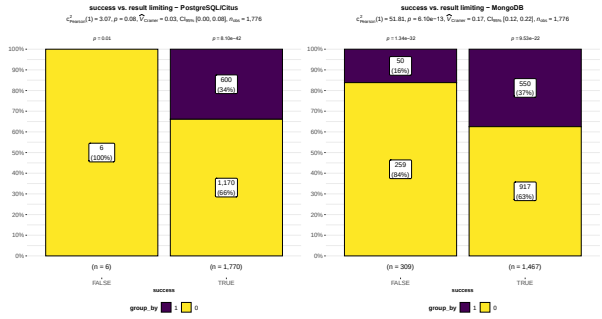


**Fig. 9.** Association of success and limiting the result size

For Citus, the Chi-square test of independence revealed a p-value greater than 0.05 and an effect size (*Cramer's V)* of 0.03, so limiting the result size seems not to be associated with the successful completion of queries. Alternatively, for MongoDB, the p-value of the test was below 0.05 and the effect size (0.17) was higher; for this database server, limiting the result size seems associated with the query completion.

Next, we examined whether the predictors in Table 2 can reliably predict the query completion. We built and tuned a series of ML classification models, with *query_completion* (variable that recorded the query success) as the outcome (target) and the remaining variables in Table 2 as the predictors. As described in Section 4.5, only two classification algorithms were employed for this paper, Random Forest (RF) and XGBoost. The metric performance for selecting the best model was *ROC_AUC*. Best RF performance was recorded for the hyper-parameter combination of *mtry* = 9 and *min_n* = 13, whereas for the best hyper-parameter combination for XGB was *mtry*=7, *min_n*=5, *tree_depth*=9, *learn_rate*=0.0618, *loss_reduction*=0.0000000189, and *sample_size* =0.509.

Fig.10 displays the performance of both algorithms on the new data (the test test). RF slightly overperformed XGB, with a *ROC_AUC* of 0.968 (relative to 0.965 for XGB). Recorded accuracy was higher for XGB - 0.950 (relative to 0.945 for RF). Consequently, for further estimation of predictor's importance, the RF best model was preferred.

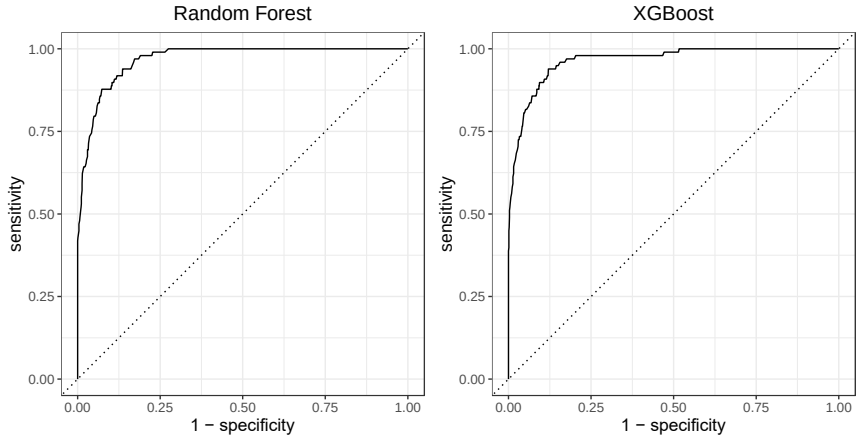


**Fig. 10.** Performance of the tuned classification models on the test set

Fig. 11 displays the permutation-based importance of variables as estimated by the RF algorithm. The graphical representation shows that the variable *db_server_pg.citus* has the highest importance, as permuting this variable results in the most significant performance loss, followed by the variables *join*,

*scale_factor* and *n_of_nodes*. Variables *group_by*, *having*, and *count* have the lowest importance with a minimal impact on the model's performance.

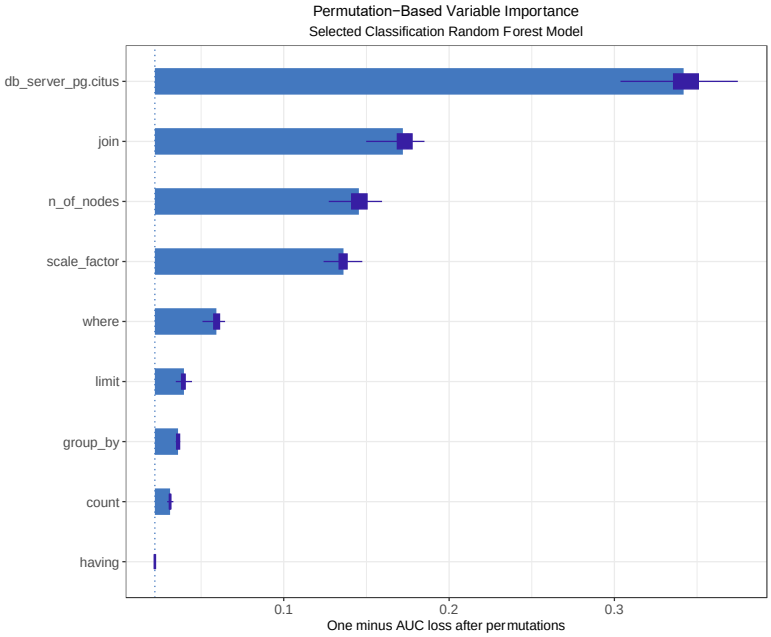


**Fig. 11.** Permutation-based variable importance on RF classification model

The six most influential variables in Fig.11 were selected for computing and displaying their effects on the average prediction of the outcome probability (probability of query to be successfully completed), using some techniques of interpretable ML. Fig. 12 shows three metrics describing the predictor's effect, Partial Dependency Profiles (PDP), Conditional Dependency Profiles (CDP) and Accumulated Local Effects Profiles (ALE).

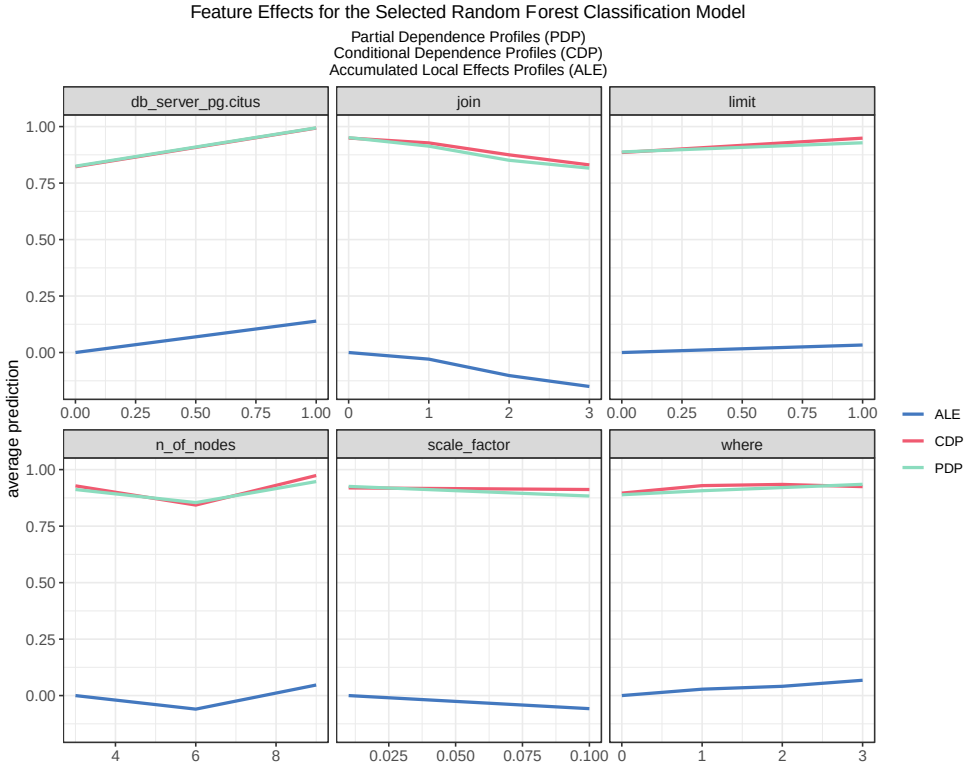


**Fig. 12.** Model explanations for the selected RF classification model.

Regarding the *db_server_pg.citus* variable, the average prediction of query completion probability appears to be positively associated with the use of the Citus database server, as indicated by the upward trend of the curve. For the *join* variable, the average prediction of query completion probability decreases as more joins are added in constructing the query. Regarding variable *limit*, setting a limited size of records contributes to achieving a higher average prediction. As for the *n_of_nodes* variable, employing either 3 or 9 nodes increase the odds of success for query completion. Unexpectedly, the influence of the database size (*scale_factor*) does not seem to differ between the two levels relatively, which can be explained by the small scales of 0.01 and 0.1 GB used. As for variable *where*, the average prediction of query completion increases when additional where clauses are added into the query.

## 6 Conclusion

Consistent with some of the earlier research, our findings indicate that distributed PostgreSQL/Citus outperforms MongoDB regarding query performance for smaller databases, even when both systems are subjects of the same requirements and physical resources. Also, we identified that the most critical factor influencing query execution is the type of database server, followed by the complexity of joins used in query construction and the data distribution setup.

The primary limitation of the experimental setup is the small database size, which, at 0.01 GB and 0.1 GB, is modest by OLAP standards. Additional limitations include the relatively small query set (296 queries), the complexity of the queries, and the brief 10-minute timeout imposed for query execution.

Future experiments on the setup should involve processing larger datasets and employing different data nesting structures in MongoDB collections. Moreover, increasing the query set size, expanding the variability of query parameters, and exploring various data distribution scenarios could provide more comprehensive insights.

## 7 Acknowledgment

Data processing and analysis for this study were supported by Competitiveness Operational Program Romania under project SMIS 124759 - RaaS-IS (Research-as-a-Service Iasi).

## References

[1] C. Madera and A. Laurent, “The next information architecture evolution: the data lake wave,” in *Proc. 8th Int. Conf. on Management of Digital EcoSystems (MEDES)*, 2016, pp. 174-180. doi: 10.1145/3012071.3012077.

[2] F. Naregsian, E. Zhu, R. Miller, K. Pu, and P. Arocena, “Data Lake Management: Challenges and Opportunities,” *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 1986-1989, 2019. doi: 10.14778/3352063.3352116.

[3] P. Pedreira, O. Erling, K. Karanasos, S. Schneider, and W. McKinney, “The Composable Data Management System Manifesto,” *Proc. VLDB Endow.*, vol. 16, no. 10, pp. 2679-2685, 2023. doi: 10.14778/3603581.3603604.

[4] D. Martinez-Mosquera, “Integrating OLAP with NoSQL Databases in Big Data Environments: Systematic Mapping,” *Big Data Cogn. Comput.*, vol. 8, no. 6, 2024.

[5] K. Prabhakar, J. Kurunandan, A. Amjad, P. Prabu, and B. Rajkumar, “OpenStackDP: A scalable network security framework for SDN-based OpenStack cloud infrastructure,” *J. Cloud Comput.: Adv. Syst. Appl.*, vol. 12, no. 1, 2023. doi: 10.1186/s13677-023-00406-w.

[6] M. Fotache, C. Badea, M. I. Cluci, C. Pinzaru, C. S. Esanu, and O. Rusu, “OLAP Performance of Distributed PostgreSQL and MongoDB on

OpenStack. Preliminary Results on Smaller Scale Factors," in *Proc. 23rd RoEduNet Conf.: Networking in Education and Research*,2024, doi: 10. 1109/RoEduNet64292.2024.10722556

**[7]** DB Engines Ranking [Online]. https://db-engines.com/en/ranking.

[8] A. Makris, K. Tserpes, G. Siliopoulos, D. Zissis, and D. Anagnostopoulos, "MongoDB vs PostgreSQL: A comparative study on performance aspects," *Geoinformatica*, vol. 25, pp. 243–268, 2021. doi: 10.1007/s10707-020-00407-w.

[9] E. Güney and N. Ceylan, "Response Times Comparison of MongoDB and PostgreSQL Databases in Specific Test Scenarios," in *ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 2022, pp. 178–188.

[10] M. T. Villalobos, L. V. Acuna, and R. Q. Oviedo, "Comparison of the Response Times of MongoDB and PostgreSQL According to Type of Query in Geographical Databases," *Computación y Sistemas*, vol. 24, no. 4, pp. 1461–1469, 2020.

[11] L. G. Wiseso, M. Imrona, and A. Alamsyah, "Performance Analysis of Neo4j, MongoDB, and PostgreSQL on 2019 National Election Big Data Management Database," in *International Conference on Science in Information Technology*, Palu, Indonesia, 2020.

[12] A. Flores, S. Ramírez, R. Toasa, J. Vargas, R. U. Barrionuevo, and J. M. Lavin, "Performance Evaluation of NoSQL and SQL Queries in Response Time for the E-government," in *2018 International Conference on eDemocracy & eGovernment (ICEDEG)*, 2018, pp. 257–262. doi: 10.1109/ICEDEG.2018.8372362.

[13] Y. D. Mukasheva, A. Bissengaliyeva, D. Suynullayev, "Performance Analysis of Scaling NoSQL vs SQL: A Comparative Study of MongoDB, Cassandra, and PostgreSQL," in *2023 IEEE International Conference on Smart Information Systems and Technologies (SIST)*, Astana, Kazakhstan, 2023, pp. 479–483. doi: 10.1109/SIST58284.2023.10223568.

[14] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments," *IEEE Access*, vol. 8, pp. 110656–110668, 2020. doi: 10.1109/ACCESS.2020.3002164.

[16] W. Ali, M. U. Shafique, M. A. Majeed, and A. Raza, "Comparison between SQL and NoSQL databases and their relationship with Big Data analytics," *Asian Journal of Research in Computer Science*, vol. 4, no. 2, pp. 1–10, 2019. Article no. AJRCOS.51946, ISSN: 2581-8260.

[15] A. Makris, K. Tserpes, G. Spiliopoulos, and D. Anagnostopoulos, "Performance evaluation of MongoDB and PostgreSQL for spatio-temporal data," in *EDBT: 22nd International Conference on Extending Database Technology*, Lisbon, Portugal, 2019, pp. 1–9.

[16] R. Čerešňák and M. Kvet, "Comparison of query performance in relational and non-relational databases," in *Proc. of the 13th Scientific Conference on Sustainable, Modern and Safe Transport (TRANSCOM 2019)*, Novy Smokovec, vol. 40, pp. 170–177, May 29–31, 2019. doi: 10.1016/j.trpro.2019.07.027.

[17] P. M. Tracz and M. Plechawska-Wójcik, "Comparative analysis of the performance of selected database management system," *Journal of Computer Sciences Institute*, vol. 31, pp. 89–96, 2024. doi: 10.35784/jcsi.5927.

[18] A. M. Kausar and M. Nasar, "SQL Versus NoSQL Databases to Assess Their Appropriateness for Big Data

Application," *Recent Advances in Computer Science and Communications*, vol. 14, no. 4, 2021, doi: 10.2174/2213275912666191028111632

[19] A. B. Setyawan, I. A. Kautsar, and N. L. Azizah, "Query Response Time Comparison SQL and No SQL for Contact Tracing Application," in *Proceedings of the 4th Seminar Nasional Sains. Procedia of Engineering and Life Science*, vol. 2, no. 2, 2022, doi: https://doi.org/10.21070/pels.v2i2.1296.

[20] D. Figueiredo, G. Saraiva, J. Rebelo, R. Rodrigues, F. Cardoso, C. Wanzeller, P. Martins, and M. Abbasi, "Performance Evaluation Between HarperDB, Mongo DB and PostgreSQL," in *Marketing and Smart Technologies. ICMarkTech 2022. Smart Innovation, Systems and Technologies*, vol. 344. Springer, 2024. doi: https://doi.org/10.1007/978-981-99-0333-7_7.

[21] M. Sals, A. Sghir, N. Rafalia, and J. Abouchabaka, "Analysis and comparison of NoSQL databases with relational databases: MongoDB and HBase versus MySQL," *International Journal on Technical and Physical Problems of Engineering (IJTPE)*, vol. 55, no. 15, pp. 155–161, June 2023, ISSN 2077-3528.

[22] J. Antas, R. R. Silva, and J. Bernardino, "Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data," *Computers*, vol. 11, no. 2, pp. 29, 2022. doi: https://doi.org/10.3390/computers11020029.

[23] B. Alyasiri, B. Sahi, and N. AL-Khafaji, "NoSQL: Will it be an alternative to a relational database? MySQL vs MongoDB comparison," in *Proceedings of 2nd International Multi-Disciplinary Conference Theme: Integrated Sciences and Technologies, IMDC-IST 2021*, Sakarya, EAI, 7-9 September 2021. doi: 10.4108/eai.7-9-2021.2314925.

[24] M. Abukabar, S. Abukabar, and U. M. Bello, "Analyzing and Designing an Evaluation Benchmark for SQL and NoSQL Database Systems for Some Selected Higher Institution in Zamfara State," *International Journal of Science for Global Sustainability (IJSGS)*, vol. 10, no. 2, p. 63, 2024, doi: 10.57233/ijsgs.v10i2.644.

[25] H. Matallah, G. Belalem, and K. Bouamrane, "Comparative study between the MySQL relational database and the MongoDB NoSQL database," *International Journal of Software Science and Computational Intelligence (IJSSCI)*, vol. 13, no. 3, pp. 38–63, 2021. doi: 10.4018/IJSSCI.2021070104.

[26] C. Axell, E. Schøien, I. L. Thon, L. O. Vågene, and L. Tveiten, "Insertion speed of indexed spatial data: comparing MySQL, PostgreSQL and MongoDB," 2022. *[Online]*: https://folk.idi.ntnu.no/baf/eremcis/2022/Group02.pdf.

[27] M. Reichardt, M. Gundall, and H. D. Schotten, "Benchmarking the operation times of NoSQL and MySQL databases for Python clients," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, Toronto, Canada, 2021, pp. 1-8, doi: 10.1109/IECON48115.2021.9589382.

[28] N. Naufal, S. Nurkhodijah, G. B. Anugrah, A. Pratama, M. I. Rabbani, F. A. Dilla, T. N. Anggraeni, and T. Firmansyah, "Comparisonal analysis of MySQL and MongoDB response time query performance," *Jurnal Informatika Dan Tekonologi Komputer (JITEK)*, vol. 2, no. 2, pp. 158-166, 2022. https://doi.org/10.55606/jitek.v2i2.245

[29] Citus Data Documentation, *[Online]*: https://docs.citusdata.com

[30] L. F. Silva and J. V. F. Lima, "An evaluation of relational and NoSQL distributed databases on a low-power

cluster," *J. Supercomput.*, vol. 79, pp. 13402–13420, 2023. https://doi.org/10.1007/s11227-023-05166-7

[31] U. Cubukcu, O. Erdogan, S. Pathak, S. Sannakkayala, and M. Slot, "Citus: Distributed PostgreSQL for data-intensive applications," in *Proc. of the 2021 International Conference on Management of Data*, ACM, 2021, doi: 10.1145/3448016.3457551.

[32] MongoDB Documentation, *[Online]*: https://www.mongodb.com/docs/manual/

[33] T. Taipalus, "Database management system performance comparisons: A systematic literature review," *J. Syst. Softw.*, vol. 208, no. 111872, 2024, doi: 10.1016/j.jss.2023.111872.

[34] Transaction Processing Council Benchmark H (Decision Support) Standard Specification, *[Online]*: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf.

[35] R, "R: A Language and Environment for Statistical Computing," R. C. Team, Producer, & R Foundation for Statistical Computing, R version 4.4.0, 2024. Available: https://www.R-project.org.

[36] H. Wickham et al., "Welcome to the Tidyverse," *J. Open-Source Softw.*, vol. 4, no. 43, pp. 1–6, 2019, doi: 10.21105/joss.01686.

[37] I. Patil, "Visualizations with statistical details: The 'ggstatsplot' approach," *J. Open Source Softw.*, vol. 6, no. 61, 2021, doi: 10.21105/joss.03167.

[38] M. Kuhn and J. Silge, *Tidy Modeling with R*, Sebastopol, California, USA: O'Reilly, 2022.

[39] P. Biecek, "Dalex: Explainers for complex predictive models in R," *J. Mach. Learn. Res.*, vol. 19, no. 84, pp. 1–5, 2018.

**Marin FOTACHE** graduated from the Faculty of Economics at Alexandru Ioan Cuza University of Iasi, Romania in 1989. He holds a PhD diploma in Business Information Systems (Business Informatics) from 2000 and he had gone through all didactic positions since 1990 when he joined the staff of Al. I. Cuza University, from teaching assistant in 1990, to full professor in 2002. Currently he is professor within the Department of Accounting, Business Informatics and Statistics in the Faculty of Economics and Business Administration at Alexandru Ioan Cuza University. He is the (co)author of books and journal/conference articles in areas such as SQL, database design, NoSQL, Big Data, Data Engineering and Machine Learning.

**Cătălina BADEA** completed her Master's degree in Data Mining at the Faculty of Economics and Business Administration, Alexandru Ioan Cuza University of Iași. Since 2024, she has been enrolled as a PhD student at the Doctoral School of Economics and Business Administration, focusing on Business Informatics. Her doctoral research approaches performance and architectural problems in Big Data platforms, exploring the transition from Data Lake to Data Lakehouse. Her conference participation includes RoEduNet: Networking in Education and Research 2024, Globalization and Higher Education in Economics and Business Administration 2024, and International Conference on Informatics in Economy (IE 2025). She serves as an associate teaching staff

member at Alexandru Ioan Cuza University of Iași, where she conducts laboratory sessions in database systems. Her research interests focus on Big Data system architectures and the performance of data processing in business applications.

**Marius-Iulian CLUCI** graduated from the Faculty of Economics and Business Administration at Alexandru Ioan Cuza University of Iași, Romania. He holds a master's degree in Software Development and Business Information Systems and currently he is a Ph.D. candidate at the Doctoral School of Economics and Business Administration. His research focuses on Big Data systems, the integration of Machine Learning in Apache Spark, and performance benchmarking of modern data architectures. He has taught on topics related to Big Data, Machine Learning, OpenStack, and distributed computing. He works as a Cloud Engineer on Microsoft Azure, designing scalable data integration and processing solutions. His academic contributions include papers on TPC-H benchmarking, Spark optimizations, and schema evolution. His areas of expertise include cloud computing, Apache Spark, Machine Learning, and data integration.

**Codrin-Stefan Esanu** graduated from the Faculty of Economics and Business Administration Iasi, specializing in Business Informatics, and obtained his Master's degree in Software Development and Business Information Systems. Currently, he is pursuing a Ph.D. at the Doctoral School of Economics and Business Administration. Started his career in the IT Service Management industry at Capgemini, he advanced quickly towards management roles. Leveraging the solid foundation in managing critical IT services, he successfully transitioned to a technical DevOps role at Cegeka Romania. Additionally, he serves as a university associate lecturer at Alexandru Ioan Cuza University in Iași, teaching courses in databases, Big Data and distributed computing. His professional expertise includes Linux administration, Scripting, Infrastructure as Code (Puppet, Ansible), Containers orchestration (OpenShift), GitOps (ArgoCD) and database systems like PostgreSQL, Citus, and Neo4j.