

An Empirical Study on Database Usage in Microservices

Maxime André^{a,*}, Marco Raglianti^b, Souhaila Serbout^c, Anthony Cleve^a, Michele Lanza^b

^aUniversity of Namur, Belgium

^bREVEAL @ Software Institute — USI, Lugano, Switzerland

^cUniversity of Zurich, Switzerland

Abstract

Microservices architectures are an integral part of modern software development. Their adoption brings significant changes to database management. Instead of relying on a single database, a microservices architecture is typically composed of multiple, smaller, heterogeneous, and distributed DBs. In these data-intensive systems, the variety and combination of database categories and technologies play a crucial role in storing and managing data. While data management in microservices is a major challenge, research literature is scarce.

We present an empirical study on how databases are used in microservices. On the dataset we collected (and released as open data for future research), considering 15 years of microservices, we examine ca. 1,000 GitHub projects that use databases selected among 180 technologies from 14 categories. We perform a comprehensive analysis of current practices, providing researchers and practitioners with empirical evidence to better understand database usage in microservices. We report 18 findings and 9 recommendations. We show that microservices predominantly use Relational, Key-Value, Document, and Search databases. Notably, 52% of microservices combine multiple database categories. Complexity correlates with database count, with older systems favoring Relational databases and newer ones increasingly adopting Key-Value and Document technologies. *Niche* databases (e.g., EventStoreDB, PostGIS), while not widespread, are often combined with a *mainstream* one.

1. Introduction

Microservices architectures have significantly gained popularity, becoming an integral part of the software development landscape. This architectural style is now widely adopted by large and software-intensive companies like Amazon, Google, and Netflix [1, 2].

Their adoption brings significant changes to database (DB) management [3, 4, 5, 6]. According to the literature, the microservices architecture paradigm, which promotes the decomposition of a system into loosely coupled, independent, heterogeneous, and manageable services, is expected to naturally extend to DBs [6]. Specifically, each microservice is expected to have its own dedicated DB(s) following the *database server per microservice* pattern [3], ensuring data autonomy and minimizing dependencies. This is aligned with the *polyglot persistence* [34, 35]. In these data-intensive systems, the large variety and combination of DB categories and technologies play a crucial role in storing and managing data. Depending on the requirements, the main motivations concern, for instance, the need for independent schema evolution, data caching, data replication, data partitioning, decentralized data management, etc. These mechanisms aim to reduce coupling and ease maintenance and evolution [3].

Although a few existing studies recognize that data management in microservices is a major challenge [7, 3], it received little attention in the research literature. Current works lack concreteness, especially regarding the available datasets and in-depth empirical investigations that highlight the current status. In particular, the variety and numerous combinations of DB categories and the specific technologies and their implementations, across multiple heterogeneous microservices, often require precise justifications to understand the underlying reasons emerging from the community trends. Indeed, despite the growing adoption of microservices architectures, there is still a noticeable gap and a lack of benchmarks in the literature [3] regarding how microservices practitioners reason and handle data management *in vivo*. Existing studies [3, 8, 6, 9, 10] confirm a trend in the adoption of multiple DBs in modern software, such as the combination of relational and document DBs, the use of a cache layer, and the exploitation of search-based mechanisms. Some conclude that a poor understanding of data management practices, such as technology combinations, could lead to the introduction of a *technical data debt* [10]. To fill this gap, concrete observations on the state of the practice could be useful to practitioners, teachers, and students, helping them to make the right technical choices.

We present an empirical study on how DBs are used in microservices. Considering 15 years of history, from 2010 to 2025, we examine ca. 1,000 open-source projects mined from GitHub that use DBs selected among 180 technologies (e.g., PostgreSQL, Redis, MongoDB) from 14 categories (e.g., Relational, Key-Value, Document).

*Corresponding author

Email addresses: maxime.andre@unamur.be (Maxime André), marco.raglianti@usi.ch (Marco Raglianti), souhaila.serbout@uzh.ch (Souhaila Serbout), anthony.cleve@unamur.be (Anthony Cleve), michele.lanza@usi.ch (Michele Lanza)

We perform a comprehensive analysis providing insights into current practices, emphasizing the most prevalent DBs used in microservices. We also investigate the way they are combined in practice, observing recurrent patterns. We support our observations with objective, fine-grained metrics and highlight relationships between characteristics (*e.g.*, complexity vs. age).

Our study leads to 18 findings about DB usage in microservices, with a two-fold implication. First, on the industry and open-source side, our work helps practitioners to understand the latest trends and, thanks to the 9 recommendations we derive, to select the most appropriate data storage strategies in their projects. Second, on the research side, it guides researchers in shaping future directions based on empirical evidence and an open-source dataset.

2. Research Methods

We describe the research methods we employed to conduct our empirical study. We present our research question, the initial list of considered DBs and technologies, and the methodology we followed to collect and analyze the data from GitHub repositories.

2.1. Research Questions

We aim to answer the following research questions (noted **RQ***) to understand how DBs are used in microservices:

- **RQ1:** *What database categories and technologies are used in microservices, and how prevalent are they?* From the open-source microservices collected, we analyze the DB dependencies, establish their distribution, and assess the most popular DB categories (*e.g.*, Relational, Document, Key-Value, Column, Graph) and technologies (*e.g.*, PostgreSQL, MongoDB, Redis).
- **RQ2:** *How are databases combined in microservices, and what are the characteristics of those combinations?* We analyze the DB categories and technologies associations in microservices, their breakdown as stated in dependencies, and determine the most popular combinations, exploring further recurrent patterns and computing relevant metrics.
- **RQ3:** *What is the relationship between the complexity of microservices and their data management strategy?* We seek to understand whether the complexity (*e.g.*, number of services, size of the project) is correlated with the number of DB technologies, whether the complexity is linked to a certain degree of category associations, or whether some category associations are more suitable for projects of certain complexity.
- **RQ4:** *What is the relationship between the age of microservices and their database choices?* We consider the age of microservices and aim to find rationales in their DB choices, to recommend different strategies for older and more recent projects.

2.2. Database Categories and Technologies

Table 1 lists the DB categories considered in our study. We extracted them from DB-Engines’ March 2025 ranking [36], considering the top 250 DB management systems (DBMSs). The exhaustive list of DB technologies is available in our replication package (Section 7).

Table 1: DBMS categories considered in our study.

Category	Example DBMS
Relational	Oracle, MySQL, MS SQL Server, PostgreSQL
Document	MongoDB, Couchbase, CouchDB
Key-Value	Redis, Memcached, etcd
Column	Cassandra, HBase, ClickHouse
Graph	Neo4j, GraphDB
Time Series	InfluxDB, kdb+, TimescaleDB
Vector	Pinecone, Milvus, Qdrant, Chroma, Weaviate
Spatial	PostGIS
Hierarchical	IBM IMS
Network	IDMS
Object	Actian, db4o, ObjectDB
Event	EventStoreDB
Search	Elasticsearch, Splunk, Solr
Others	Amazon DynamoDB, Aerospike

Since the presence as a Docker container is a distinction criterion (see Section 2.3), we excluded DB technologies unavailable as Docker images (*e.g.*, Microsoft Access, FileMaker). Following the methodology of Paiva *et al.*, we also excluded warehouses (*e.g.*, Snowflake, Databricks, Apache Hive, Google BigQuery), frameworks (*e.g.*, Apache Flink), services (*e.g.*, Amazon Aurora, Prometheus), and platforms (*e.g.*, Google Firebase, Google Firestore, Microsoft Azure Table Store) [9]. In the end, we considered a total of 180 technologies.

DBs are categorized by type (Relational, Document, Key-Value, Column, Graph, Time Series, Vector, Spatial, Hierarchical, Network, Object, Event, Search Engine) and ranked by popularity according to DB-Engines. Information and classifications were cross-verified through the literature [11, 12, 13, 14, 15, 3, 8, 9] and online sources such as Database of Databases [37] and Wikipedia [38, 39, 40, 41, 42, 43].

For each DB, the first category is considered to be the main one. Less common, ambiguous, or multi-type DBs (*e.g.*, Native XML, RDF), are grouped under the “Others” category. Each DB is associated with a regular expression (RE) to identify its corresponding Docker image, if available. REs, verified on Docker Hub [44], are included in our replication package.

2.3. GitHub Repositories and Microservices

We mined GitHub using its REST (REpresentational State Transfer) API (Application Programming Interface) [45] to collect microservices project repositories that use DBs. Mining repositories is valuable for researchers seeking up-to-date real-world systems to evaluate their approaches and tools [16]. Benchmarks and datasets specifically studying microservices and DBs are limited in the literature [17, 18]. There are several challenges to be addressed to obtain a good benchmark.

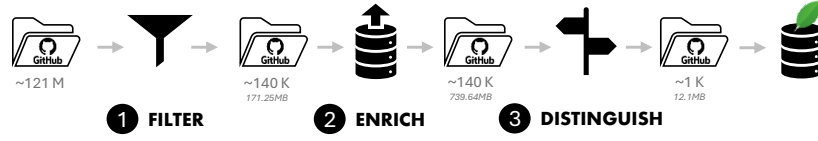


Figure 1: The mining process for extracting microservices with DBs from GitHub.

First, it is not straightforward to identify a GitHub repository belonging to a system that adheres to a microservices architecture, as they can follow various organizational structures, such as *mono-repositories* (a.k.a. *mono-repo*) [6] or *multi-repositories* (a.k.a. *multi-repo*). Additionally, documentation that lists and describes all the microservices within a given architecture is often unavailable, further complicating the mining process. When examining a specific repository, there is rarely a clear and explicit indication that it belongs to a microservices architecture. Sometimes, terms like “microservice” might help, but other associated terms like “REST API” can also be observed in titles, tags, descriptions, or README files, either at the top level of the architecture or at sub-levels representing parts of the architecture. Since a microservice is modular and distributed, it can be difficult to scope. Some components can be spread, isolated in other locations, without any clear links. The heterogeneity of implementations affects the automation capabilities of mining such repositories. These challenges often lead to noisy or incomplete results. Current benchmarks commonly require manual annotation, which slows down the process and limits the number of results included and analyzed. To address these challenges, we propose a mining process that combines several fine-grained filters and heuristics to reduce the difficulty of characterizing microservices repositories and provide a large benchmark dataset of microservices architectures.

The mining process is depicted in Figure 1. To ensure the reproducibility of our research, the source code and the complete benchmark dataset are available in our replication package.

1 Filter. Since GitHub may host private and inactive projects [16, 8, 9], we applied six filtering criteria to efficiently narrow down our search from 121 million repositories. The aim was to identify the relevant ones that are active real-world systems. After filtering we retained 140,000 repositories based on:

1. **Disk Size:** To eliminate outliers and retain repositories with meaningful content, we filtered them based on disk size [8]. We selected those with a size between 500 KB and 1 GB.
2. **Stars Count:** To retain relevant repositories by popularity, we applied a filtering criterion based on the number of stars [19]. We set the threshold to at least 100 stars.
3. **Commit History:** To target real and actively maintained systems, we retained only repositories with at least 100 commits, inspired by the work of d’Aragona *et al.* [18].
4. **Structural Completeness:** To remove placeholders and projects without the basic level of documentation encouraged on GitHub, we included only repositories with at least one README file and at least two directories [16].

5. **Recent Updates:** We focused on repositories that are likely to follow modern microservices practices by selecting only those updated after January 1, 2015. This date is often considered to mark the widespread adoption of microservices architectures [35].

6. **Programming Languages:** We targeted repositories written in popular programming languages, particularly those frequently used in microservices development [14, 15, 17, 18], [46]. The selected languages include C, C++, C#, Go, Java, JavaScript, PHP, Python, Ruby, Scala, and TypeScript.

2 Enrich. To better identify microservices repositories, we enrich the dataset with additional information. Using the GitHub API, we retrieve the content of README files, which often contain keywords and valuable project details, as well as *Docker Compose* files,¹ commonly used to define multi-container environments typical of microservices architectures [6, 18].

3 Distinguish. Based on the enriched dataset, we deepen the filtering process targeting repositories likely to be microservices. We define a number of heuristics to compute a score for each repository. The higher the score, the more likely the repository is a microservice. The heuristics are based on the presence of keywords in the repository, the number of specific files and directories, the presence and content of README and Docker Compose files, and the number of services and DBs declared in the Docker Compose files.

The keywords are the following variants: *microservice*, *micro-service*, *micro service*, *microservices*, *micro-services*, *micro services*. In addition, we add keyword variants about the different organizational structures: *monorepo*, *mono-repo*, *multirepo*, *multi-repo*. Finally, the *rest api* keyword aims to include the parts of microservices architectures that are outside the scope of a single repository and are intended to be served as external APIs for other services.

We compute the score based on the following heuristics:

- A keyword is featured in the title, the description, the repository topics, the contents of the README files.
- A keyword is featured in at least a directory, a file, a Docker Compose file.
- The repository Docker Compose files declare at least one service or one DB.
- The repository counts more services than DBs.

¹<https://docs.docker.com/compose/>

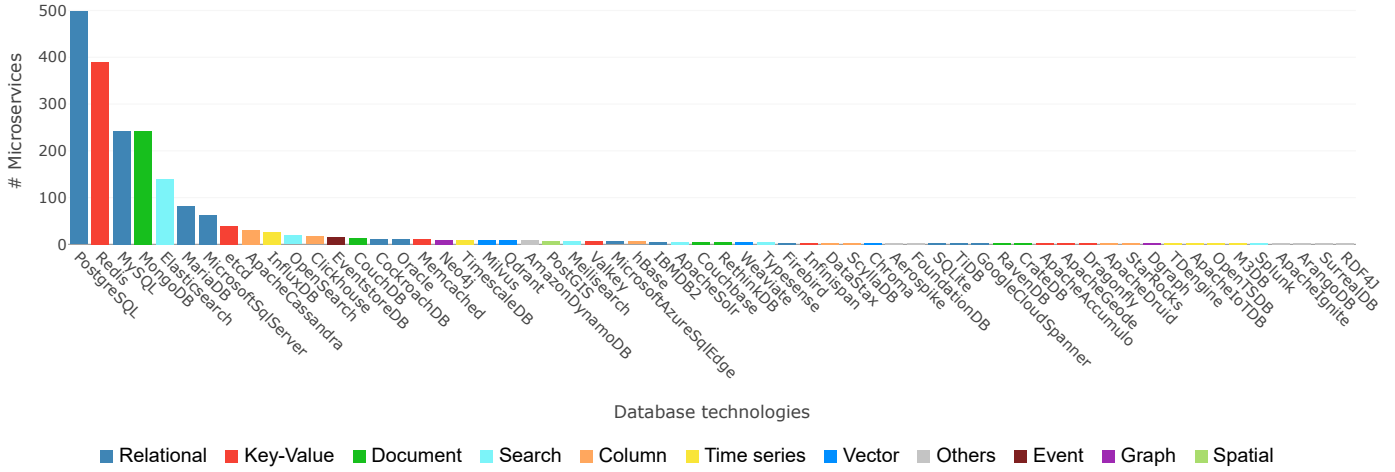


Figure 2: DB technologies utilized in microservices, colored by category.

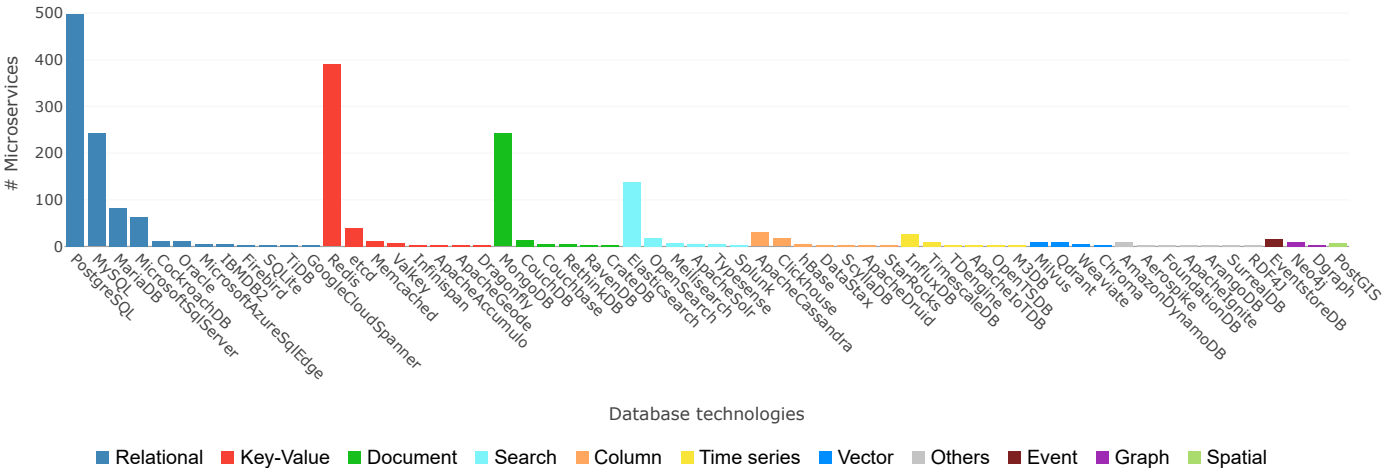


Figure 3: DB technologies utilized in microservices, colored and grouped by category.

These heuristics aim to compute a likelihood score for a repository to have a microservices architecture. All heuristics are assessed independently.

Once the score is computed, in order to flag the repository as a microservices architecture, it must satisfy all of the following conditions:

- The score is (strictly) greater than 0.
- The Docker Compose files declare at least one service.
- The repository contains at least one keyword in the title, description, topics, or README files.
- The Docker Compose files declare at least one DB.
- The repository counts more services than DBs.

We found 1,005 repositories that are likely to be microservices. We stored them in a DB containing, for each repository: GitHub ID, GitHub URL, git branch, owner username, repository title, repository description, GitHub associated topics, creation date, last updated date, disk size, star count, com-

mit count, contributor count, directory count, service directories list, service files list, README files and their content, Docker Compose files and their content, service count based on the Docker Compose files, DB list based on the ones declared in the Docker Compose files, and programming languages list. The DB dump is available in our replication package.

3. Results

3.1. RQ1: Database Usage in Microservices

We compute the distribution of DB technologies and categories across all repositories. For each technology, we count the number of repositories declaring it in their Docker Compose files. In Figure 2, we present the DB technologies (x-axis) we found in our dataset, sorted by popularity (*i.e.*, number of repositories that use them, y-axis). In Figure 3, we grouped them by DB category.

Among the 1,005 repositories in our dataset, a total of 60 distinct DB technologies are identified out of the 180 considered, highlighting significant heterogeneity with 11 different DB categories.

The most popular DB categories are Relational, Key-Value, Document, and Search. As reported in Table 2, Relational DBs appear in 71.64% of repositories, followed by Key-Value DBs in 42.09%, Document DBs in 25.77%, and Search DBs in 16.32%.

Table 2: Distribution of DB categories in microservices.

Category	Count	%	Category	Count	%
Relational	720	71.64%	Time Series	38	3.78%
Key-Value	423	42.09%	Vector	19	1.89%
Document	259	25.77%	Event	15	1.49%
Search	164	16.32%	Graph	10	1.00%
Column	50	4.98%	Spatial	8	0.80%
			<i>Others</i>	14	1.39%

A repository is counted in a category if it has at least one technology from it. Relational DBs also dominate in terms of the number of distinct technologies (12), followed by Key-Value DBs (8), Document DBs (6), and Search DBs (6). At the other side of the spectrum we find the Spatial DBs category, with a single technology (PostGIS).

Among our collected data, some categories and technologies cannot be found. Notably, no Hierarchical, Network, or Object DBs are identified in microservices. Table 3 shows the distribution of the top 10 most popular DB technologies.

Table 3: Top 10 DB technologies used in microservices.

Technology	Category	Count	Percentage
PostgreSQL	Relational	498	49.55%
Redis	Key-Value	390	38.81%
MySQL	Relational	242	24.08%
MongoDB	Document	242	24.08%
Elasticsearch	Search	139	13.83%
MariaDB	Relational	82	8.16%
Microsoft Sql Server	Relational	63	6.27%
etcd	Key-Value	39	3.88%
Apache Cassandra	Column	30	2.99%
InfluxDB	Time Series	27	2.69%

For the Relational category, PostgreSQL and MySQL dominate, with 49.55% and 24.08% of repositories, respectively. Redis, for Key-Value DBs, is present in 38.81% of repositories. MongoDB brings the top Document DB technology on par with the second most popular in the Relational category, with 24.08%.

Table 4 summarizes the most popular DB technology of each category and its in-category percentage (*i.e.*, popularity within the category).

PostgreSQL is chosen 69.17% of the times when Relational DBs are needed. With MySQL, they almost have a monopoly on Relational DBs. In Key-Value and Document DBs, Redis and MongoDB cover 92.20% and 93.44% respectively, making them almost a *de facto* standard, save for a few exceptions. Elasticsearch occupies 84.76% of the Search landscape. Apache Cassandra concerns 60.00% of Column DBs and, for Time Series DBs, InfluxDB is present in 71.05% of cases. They

Table 4: Most popular DB technology for each category.

Category	Technology	Count	Percentage
Relational	PostgreSQL	498	69.17%
Key-Value	Redis	390	92.20%
Document	MongoDB	242	93.44%
Search	Elasticsearch	139	84.76%
Column	Apache Cassandra	30	60.00%
Time Series	InfluxDB	27	71.05%
Event	EventStoreDB	15	100.00%
Graph	Neo4j	10	100.00%
Vector	Milvus	9	47.37%
Spatial	PostGIS	8	100.00%

are popular alternatives but not the only option in their respective categories. On the contrary, while EventStoreDB, Neo4j, and PostGIS are not widespread, they have almost no alternatives in their respective categories. Finally, Milvus and Qdrant are similarly popular alternatives for Vector DBs.

RQ1: Findings

- F1.1 *Four different categories of technologies are widespread in microservices architectures (Relational, Key-Value, Document, and Search).*
- F1.2 *Relational DBs are still prevalent, by a large margin, with multiple popular alternative technologies.*
- F1.3 *Specific DB categories, despite their niche applications, should not be overlooked. For example, Graph and Spatial DBs, while addressing specialized needs, are utilized in 10 and 8 repositories respectively, indicating their relevance.*
- F1.4 *For most categories, there is a specific technology which is the most widespread (e.g., Redis for Key-Value, MongoDB for Document). Relational DBs have a more varied “footprint” of popular technologies.*
- F1.5 *Sixty unique DB technologies show that diversity is preferred to one-size-fits-all DBs.*

3.2. RQ2: Database Associations in Microservices

For each repository and each Docker Compose file, we inspect the declared DB technologies. Figure 4 presents the repositories (rows) and the DB technologies used (columns). The repositories are sorted by number of DB technologies and the DBs by popularity (see RQ1). The cells are filled if the repository declares the DB technology in one of its Docker Compose files. Only the top 25 repositories are shown. A full interactive version, allowing access to the DB declaration in GitHub, is provided in the replication package.

Figure 5 (top) proposes an overview of the 1,005 repositories (columns) and their DB categories (rows).

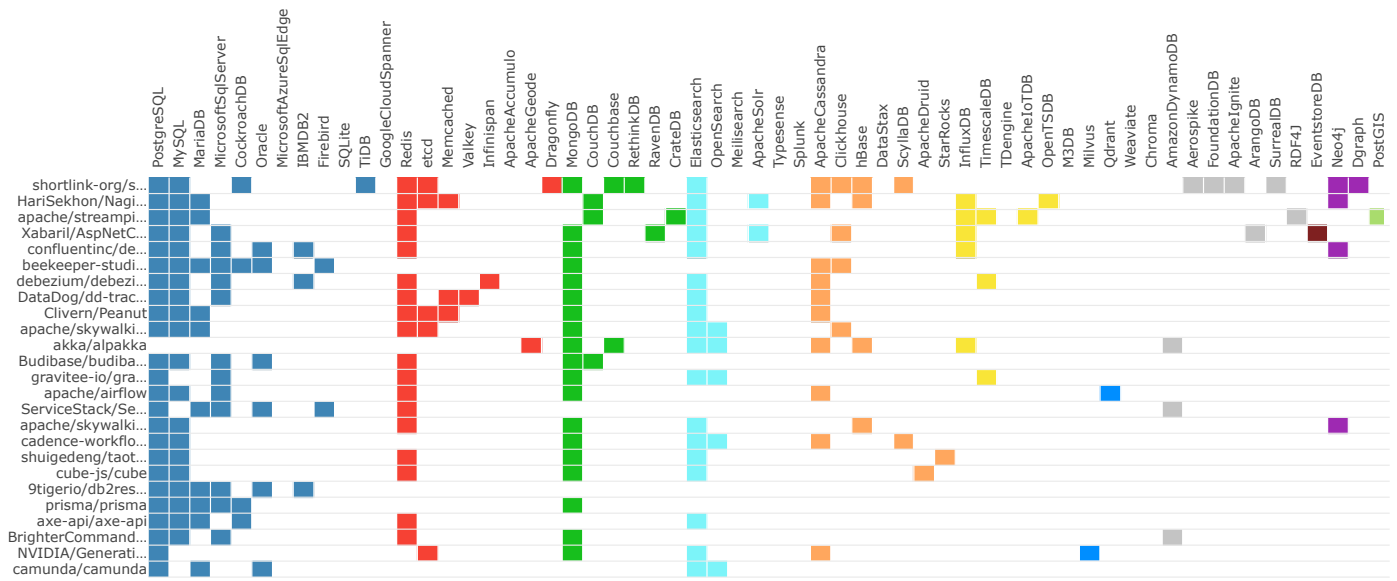


Figure 4: Top 25 repositories sorted by number of DB technologies, colored by categories.



Figure 5: Repositories and their DB categories overview (top). Zoom-in on the top 25 repositories sorted by number of DB technologies (bottom).

The sorting of repositories and DBs remain the same. The colored dots show the presence of the DB category in the repository. A zoom-in is done on the top 25 repositories in Figure 5, bottom. A full version is available in the replication package.

The aim of these figures is to demonstrate the heterogeneity of DBs in microservices and to highlight the most popular ones. On the 1,005 repositories collected, we compute what we call the *database heterogeneity rate* (DHR), which is the ratio between the number of microservices combining at least two DB technologies (or categories) and the total number of microservices. The DHR based on technologies is 0.52: Half of the repositories mix two different technologies. The DHR based on categories is 0.47, which highlights that some microservices also use different technologies within the same category.

In Figure 6, we present a cross-matrix showing, for each pair of possible DB categories, the percentage of combinations. The y-axis and x-axis represent DB categories. They are sorted by popularity according to the results obtained previously. Cells are filled with a color gradient from white to black depending on the percentage. Cells account also for the combinations of more than just the exclusive pair, as long as the two categories are present in the tuple. The diagonal indicates the percentage of repositories having the category.

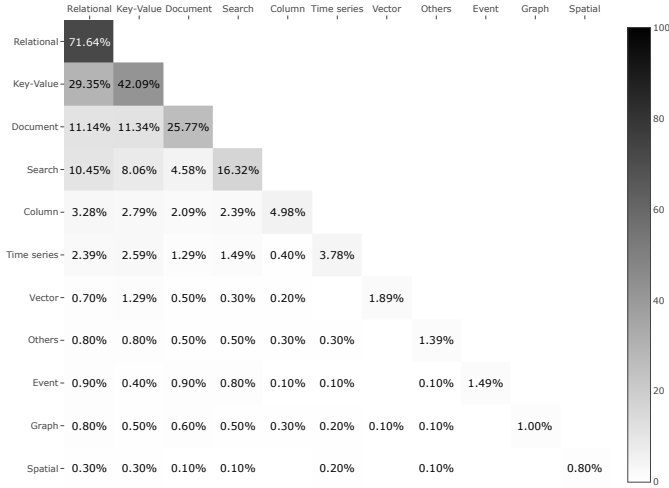


Figure 6: Pairwise combinations of DB categories.

$\langle \text{Relational}, \text{Key-Value} \rangle$ is the pair of DB categories most frequently combined, in 29.35% of cases, followed by $\langle \text{Key-Value}, \text{Document} \rangle$ in 11.34% of cases, and closely by $\langle \text{Relational}, \text{Document} \rangle$ in 11.14% of cases. Combinations paired with Search are quite popular: 10.45% for $\langle \text{Relational}, \text{Search} \rangle$, 8.06% for $\langle \text{Key-Value}, \text{Search} \rangle$, and 4.58% for $\langle \text{Document}, \text{Search} \rangle$. The remaining associations appear in less than 3% of cases. Some are non-existent: No combinations appear for $\langle \text{Column}, \text{Spatial} \rangle$, $\langle \text{Time Series}, \text{Vector} \rangle$, $\langle \text{Vector}, \text{Event} \rangle$, $\langle \text{Vector}, \text{Spatial} \rangle$, $\langle \text{Event}, \text{Graph} \rangle$, $\langle \text{Event}, \text{Spatial} \rangle$, or $\langle \text{Graph}, \text{Spatial} \rangle$, suggesting that niche categories are generally not associated with each other. To overcome the limited overview provided by pairwise associations, we report all combination patterns observed for the top 5 DB categories with all the subsets in their power set (and exclusive use).

Table 5 presents them formally with the mathematical notation of sets (e.g., R , K), difference (\setminus), union (\cup), and intersection (\cap). Then, Figure 7 shows the corresponding Sankey diagram [20] highlighting the frequency of each pattern.

Table 5: DB categories associations in microservices.

Association	#	Association	#
$R \setminus \{K \cup D \cup S \cup C\}$	337	$R \cap K \cap D \cap S \cap C$	11
$R \cap K$	183	$K \cap S$	6
$R \cap K \cap D$	43	$K \cap D \cap S$	8
$R \cap D$	37	$R \cap S \cap C$	3
$K \setminus \{R \cup D \cup S \cup C\}$	70	$R \cap D \cap S \cap C$	1
$R \cap K \cap S$	39	$K \cap C$	2
$R \cap S$	32	$K \cap D \cap C$	2
$R \cap K \cap D \cap S$	10	$D \cap S$	8
$K \cap D$	36	$K \cap D \cap S \cap C$	2
$R \cap K \cap C$	4	$K \cap S \cap C$	2
$R \cap C$	7	$S \setminus \{R \cup K \cup D \cup C\}$	31
$R \cap K \cap D \cap C$	2	$D \cap C$	1
$R \cap D \cap S$	6	$S \cap C$	2
$D \setminus \{R \cup K \cup S \cup C\}$	90	$C \setminus \{R \cup K \cup D \cup S\}$	6
$R \cap D \cap C$	2	$D \cap S \cap C$	0
$R \cap K \cap S \cap C$	3		

Sets: (R)elational = 720, (K)ey-Value = 423, (D)ocument = 259, (S)earch = 164, (C)olumn = 50.

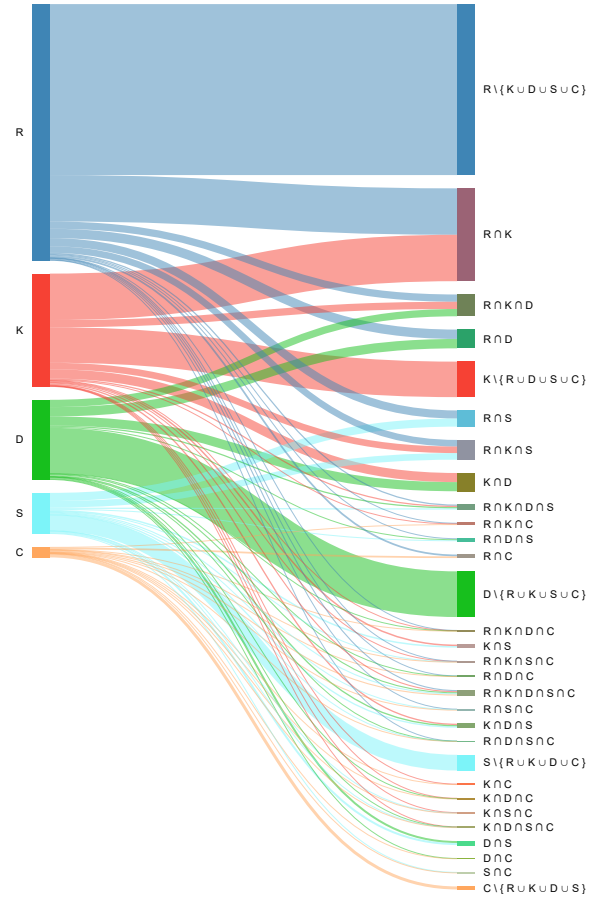


Figure 7: Sankey diagram of the 5 DB categories associations.

The most popular association patterns for two categories are: $\langle \text{Relational}, \text{Key-Value} \rangle$ (18,21%), $\langle \text{Relational}, \text{Document} \rangle$, and $\langle \text{Relational}, \text{Search} \rangle$. For triplets: $\langle \text{Relational}, \text{Key-Value}, \text{Document} \rangle$ (4,28%) and $\langle \text{Relational}, \text{Key-Value}, \text{Search} \rangle$. On the side of quartets, $\langle \text{Relational}, \text{Key-Value}, \text{Document}, \text{Search} \rangle$ is the most popular one with 10 occurrences, even though it represents only 1% of the dataset. For quintets, we reveal 11 microservices repositories opting for $\langle \text{Relational}, \text{Key-Value}, \text{Document}, \text{Search}, \text{Column} \rangle$. Once again, this happens only for a tiny fraction of the repositories in the collected dataset, but the fact that there are a few repositories mixing all the categories highlights the high variety *in the wild*.

The relationships between the associations are not transitive. For example, no repository follows the $\langle \text{Document}, \text{Search}, \text{Column} \rangle$ pattern, although $\langle \text{Document}, \text{Search} \rangle$, $\langle \text{Document}, \text{Column} \rangle$, and $\langle \text{Search}, \text{Column} \rangle$ associations exist. This last duo also shows that in the top 5, some repositories exist without any DB categories from the top 3. Furthermore, excluding only the top category (Relational DBs), we can observe that several associations exist that do not include the most popular category. For instance, we can notice 36 microservices repositories with only Key-Value and Document DBs. Another interesting observation is the number of microservices repositories containing one and only one category of DBs. For instance, 337 out of 1,005 repositories (33.53%) contain only Relational DBs, 90 (8.96%) contain only Document DBs, and 70 (6.97%) contain only Key-Value DBs. Those are in the complement of the 0.47 DHR.

To analyze how niche DBs are connected with mainstream ones, we propose a graph-based representation depicting links between DB categories in Figure 8.

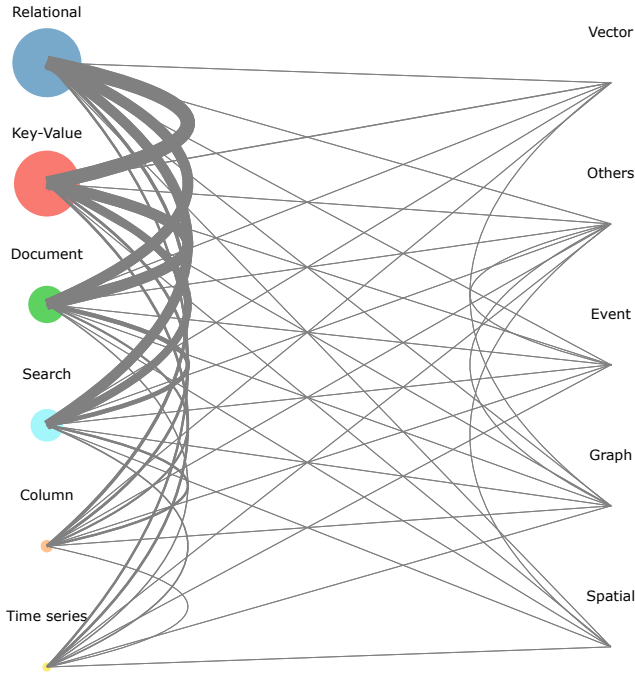


Figure 8: Pairwise associations between mainstream (left) and niche (right) DB categories.

Nodes represent DB categories, whose size depends on the popularity, and links represent DB associations, where the size of the link is proportional to the number of associations in that category. Nodes on the left represent mainstream DB categories, while nodes on the right represent niche ones.

In line with previous observations, this graph confirms that niche DBs are rarely associated with each other (0.37%). It also shows that, in most cases, niche DBs are commonly associated with a mainstream one (12.34%). Mainstream-mainstream DB associations are the most popular (87.29%).

Finally, considering specific DB technologies, the most popular associations are $\langle \text{PostgreSQL}, \text{Redis} \rangle$, $\langle \text{Redis}, \text{MongoDB} \rangle$, and $\langle \text{PostgreSQL}, \text{MongoDB} \rangle$ for duos, $\langle \text{PostgreSQL}, \text{Redis}, \text{Elasticsearch} \rangle$ and $\langle \text{PostgreSQL}, \text{Redis}, \text{MongoDB} \rangle$ for trios, and $\langle \text{PostgreSQL}, \text{Redis}, \text{MongoDB}, \text{Elasticsearch} \rangle$ for quartets.

RQ2: Findings

- F2.1 52% of the repositories declare two DB technologies and 47% contain DBs from two distinct categories. Half of the repositories are mono-technology/-category.
- F2.2 Some microservices repositories also declare several DBs belonging to the same DB category.
- F2.3 The DB categories practitioners combine the most (in pairs, trios, and all together) are Relational, Key-Value, Document, and Search, associated according to several different and non-transitive patterns.
- F2.4 There are empty association patterns (e.g., Document, Search, and Column) that should be investigated further.
- F2.5 In most cases, niche DBs are associated with a mainstream one (12.34%) and rarely with other niche DBs (0.37%).
- F2.6 The most popular DB technology associations are PostgreSQL with Redis for duos, PostgreSQL, Redis, Elasticsearch for trios, and PostgreSQL, Redis, MongoDB, and Elasticsearch for quartets.

3.3. RQ3: Microservices Complexity & Databases

To analyze the complexity of microservices and their DBs, we compute the number of services (excluding DBs) declared in the Docker Compose file(s). We create scatter plots comparing the number of services (x-axis) with the number of DB technologies, in Figure 9, and categories, in Figure 10, on the y-axis. We use the number of services within a microservices architecture as a proxy measure of complexity. We estimate the slope of the linear regression and plot the corresponding regression lines to identify trends and explore the potential relationship between microservice complexity (x-axis) and DB (y-axis).

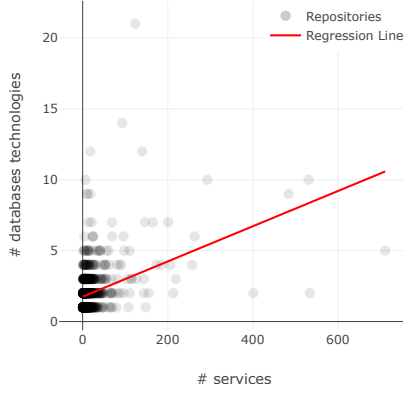


Figure 9: Comparison of the number of services and the number of DB technologies in microservices.

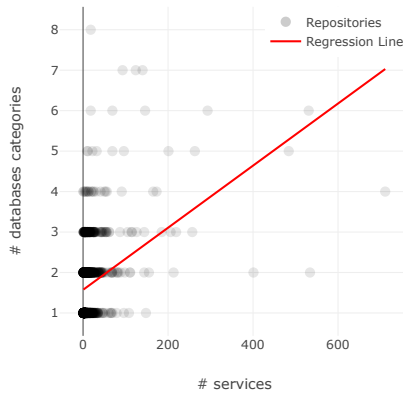


Figure 10: Comparison of the number of services and the number of DB categories in microservices.

We also perform a Student's t-test to assess whether the slope differs significantly from zero. The null hypothesis assumes no linear relationship (*i.e.*, a slope of zero). We consider the null hypothesis rejected if the resulting p -value is less than or equal to 0.05, indicating statistical significance. In our case, the linear regressions suggest that the more services there are, the more DBs there are. Results are confirmed as statistically significant (p -value ≤ 0.05).

Following the same approach, we analyze another perspective with a different complexity proxy. We compare the disk size of each repository (x-axis) with the number of DB technologies in Figure 11 and categories in Figure 12 (y-axis). Larger microservices architectures tend to have more DBs in a statistically significant way (p -value ≤ 0.05).

The two complexity proxies we used agree on indicating the complexity of a microservices architecture as linked to the number of DBs it contains.

To complement this observation with concrete examples, we analyze the popular DB technologies and categories in projects that the considered proxies identify as complex. We select 61 projects among the 1,005 that have at least 80 MB in size, 20 services, and 2 DBs.

The popular technologies differ slightly from those found across the entire dataset (see RQ1). The top 5 are Redis, PostgreSQL, MySQL, Elasticsearch, and MongoDB.

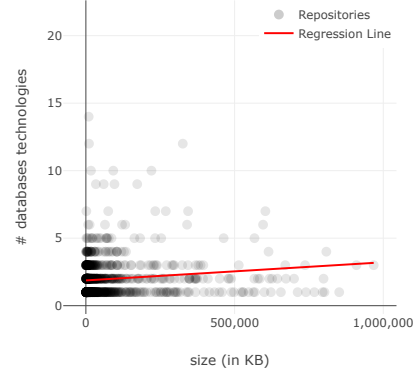


Figure 11: Comparison of the repository size on disk and the number of DB technologies in microservices.

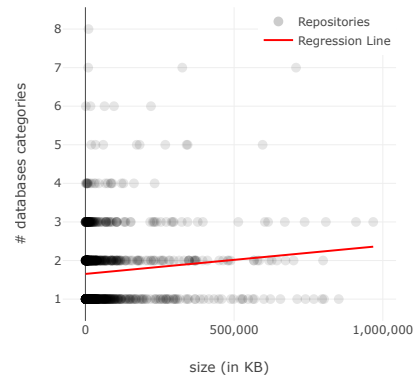


Figure 12: Comparison of the repository size on disk and the number of DBs categories in microservices

Redis overtakes PostgreSQL as the most used DB, indicating a shift from Relational to Key-Value in complex architectures, likely due to increased caching needs. Elasticsearch surpasses MongoDB, highlighting the growing role of Search DBs in complex scenarios.

At the category level, the distribution remains similar to the overall dataset, with Relational DBs still offering the most diverse technology options.

RQ3: Findings

- F3.1 *The complexity of a microservices architecture in terms of number of services is correlated both with the number of DB technologies and categories.*
- F3.2 *The number of DBs and DB categories is dependent from the size of microservices architectures.*
- F3.3 *Relational DBs are still prevalent, regardless of the complexity of microservices architectures.*
- F3.4 *In the most complex systems, DB technologies are in a different order of popularity than in the complete dataset. Redis is more popular than Postgres in complex systems and Elasticsearch is above MongoDB.*

3.4. RQ4: Microservices Age and Databases

The results presented in the following scatter plots show the relation between the age of a repository (x-axis) and its number of DB technologies in Figure 13 and categories in Figure 14 (y-axis).

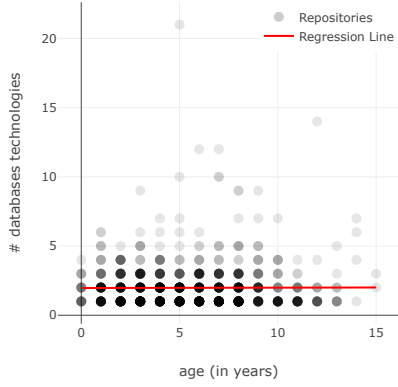


Figure 13: Age vs. number of DB technologies.

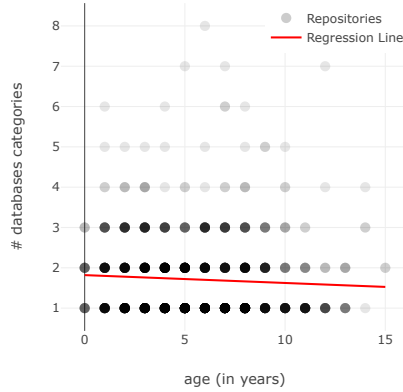


Figure 14: Age vs. number of DB categories.

We estimate the slope, draw the regression lines, and perform the Student’s t-test to derive the p -value. Results indicate that we cannot reject the null hypothesis and thus we cannot conclude statistically significant correlations (p -value > 0.05) between the age of the repositories and the number of DB technologies and categories used.

We conduct complementary investigations to identify the 5 most popular DB technologies and categories according to two age groups.

We analyze the oldest (13 years or more) and the most recent (2 years or less) microservices we collected. PostgreSQL, MySQL, Redis, and MongoDB are the most used in both age groups (although in a different order). Among the oldest, the top 3 are Relational. In contrast, we observe a more diverse distribution in newer projects, with the introduction of Key-Value and Document DBs. While MariaDB is more popular among older projects, Elasticsearch has taken its place in the top 5 for newer projects, confirming a clear shift.

RQ4: Findings

- F4.1 *No conclusion can be drawn regarding a correlation between the age of a microservices architecture and the heterogeneity of its DBs.*
- F4.2 *Microservices shifted from Relational DBs of older architectures (PostgreSQL, MySQL, and MariaDB as the top 3 technologies), to Key-value, Document and Search categories, with Redis, MongoDB, and Elasticsearch leading in popularity in younger projects (less than 2 years old) after PostgreSQL.*
- F4.3 *The relational model remains predominant although it is now 5 times less present in the most recent microservices.*

4. Implications and Recommendations

In this section we analyze the implications of the presented results for the state of practice. We compare our findings with previous literature to highlight up-to-date recommendations for practitioners.

4.1. Database Usage and Prevalence in Microservices

Relational and Document DB categories are expected to be prevalent in microservices due to their popularity in monolithic architectures and their ability to manage structured and semi-structured data [3, 8]. Document and Key-Value DBs are common due to their flexibility and adaptability to the dynamic nature of microservices [3]. Search DBs are also expected to be used, fulfilling a specific role in microservices that require search capabilities [3, 8]. We confirm the popularity of these four categories, noting that Key-Value DBs are way more popular (42.09%) than Document DBs (25.77%). This is a key difference with respect to what has been found for general software [8]. Our work also highlights how Column and Time Series DBs are important for microservices architectures, given the fact that these categories are present in tens of repositories (respectively, ranking 5th and 6th in the top 10).

In terms of technologies, PostgreSQL, MySQL, SQL Server, Redis, MongoDB, and Elasticsearch are expected to be among the most commonly used DB technologies [3, 8]. Our results confirm their popularity, now reporting them in a ranking dedicated to microservices.

Adding to this picture, MariaDB, etcd, Cassandra, and InfluxDB are also part of the top 10 most popular technologies, while previously they were flying under the radar. There are “safe” choices that, by general consensus, are preferable. We can refer to the top technology in each category including PostgreSQL for Relational DBs, Redis for Key-Value, MongoDB for Document, Elasticsearch for Search, Cassandra for Column, InfluxDB for Time Series, Milvus for Vector, EventStoreDB for Event, Neo4j for Graph, and PostGIS for Spatial DBs.

Regarding less common and specialized DB technologies, they are expected to address specific requirements within microservices, as encouraged by the architectural style [3]. Our results reveal the specific DB technologies selected for such niche goals. Examples include EventStoreDB, Neo4j, and PostGIS.

R1.1 Variety training. *Microservices practitioners should be aware of the variety of DB categories and technologies commonly used in such architectures. Training on Relational, Key-Value, Document, and Search DBs is still essential.*

R1.2 Niche awareness. *Microservices developers should be aware of specific DB categories addressing niche goals, such as Column, Time Series, Vector, Event, Graph, or Spatial DBs.*

R1.3 Popularity consideration. *Microservices developers' training should include popular technologies like PostgreSQL, MongoDB, Elasticsearch, and eventually Cassandra, InfluxDB, Milvus, EventStoreDB, Neo4j, and PostGIS.*

R1.4 Technology vs. principles. *In some categories (e.g., Event, Graph, Spatial), knowledge of the most popular technology is enough to guarantee direct applicability in most cases, while for other (e.g., Relational), knowledge of the underlying principles can improve "knowledge portability" with respect to the many alternative technologies.*

4.2. Database Associations in Microservices

Regarding DB associations in microservices, practitioners often combine multiple technologies and typically at least two distinct categories [3]. Our results confirm this degree of heterogeneity on technologies. 52% of the selected repositories declare two DB technologies, with 47% from two distinct categories. Nevertheless, the remaining half does not embrace polyglot persistence, declaring only a single technology and category. Surprisingly, some microservices repositories declare several DBs belonging to the same category. These are interesting candidates for future studies on the role and evolution of co-existing technologies that our dataset provides.

The association of Relational and Document DBs together has been previously found to be prevalent [3, 8]. Some repositories also integrate with them Key-Value and Search-based DBs, all simultaneously [3, 8].

We pointed out the various patterns in the associations among these categories. Other associations with niche DBs concern only a few microservices. While DB categories are more commonly combined in pairs or trios, there exist repositories without any Relational, Key-Value, and Document DBs (*i.e.*, the top 3 categories). There is an interesting and unexpectedly empty association pattern concerning the combination of Document, Search, and Column categories that deserves further attention.

In the literature, associations among PostgreSQL, MongoDB, Redis, and Elasticsearch, have been found to occur more frequently [3, 8]. Our analyses confirm this result. Instead, our analyses highlight how niche categories (*e.g.*, Vector, Event, Graph, Spatial) are less commonly combined with each other. Practitioners prefer to pair them with mainstream ones.

R2.1 Heterogeneity handling. *Microservices practitioners should handle the heterogeneity and association of DB technologies, especially among Relational, Key-Value, Document, and Search DBs, by being able to master two or three categories at the same time. Competence with a single specific DB is often insufficient.*

R2.2 Homogeneity acceptance. *However, in about half of the cases, homogeneity is observed. Microservices architectures developers should also be prepared for single-category and single-technology scenarios, regardless of the polyglot persistence.*

4.3. Microservices Complexity and Databases

Microservices tend to increase the technical complexity [6]. It is expected to positively correlate with the use of a greater number of DB technologies [21, 22]. For instance, microservices network centrality derived from inter-service calls is associated with the number of public methods and call frequency [23], reflecting the potential increase in data access and, consequently, in the number of DBs. Our results confirm that the number of services and the size of the project are correlated with the number of DB categories and technologies.

In terms of technologies, complex architectures should favor Document DBs for their flexibility in handling constraints and Key-Value DBs for caching and performance optimization [3]. We found that in the most complex systems in our dataset, Redis is over PostgreSQL and Elasticsearch is over MongoDB. Nevertheless, Relational DBs remain very popular with multiple alternatives. As a side effect, this gives to Redis the top spot among the technologies.

R3.1 Complexity management. *Practitioners contributing to complex systems should be able to manage multiple DB technologies across different categories.*

R3.2 Variety-induced gap. *Although Relational DBs are prevalent in general, the single most relevant technology for practitioners is Redis.*

4.4. Microservices Age and Databases

We analyzed the relationships between the age of the architecture and number of DB categories and technologies. Older microservices are expected to accumulate more DBs [21, 22].

Our analyses show there is no statistically significant correlation between the two metrics.

Regarding the popularity of specific categories over time, trends suggest a slightly decreased reliance on Relational DBs [8], [47], especially in older microservices architectures refactoring their codebase to leverage the newer technologies.

Our results confirm this scenario, with Relational DBs popular in older microservices (13 years or more) while Document, Column, and Search DBs, are preferred in recent systems [3, 8], [48, 49, 50, 51].

R4.1 Cautious replacement. *Microservices contributors should not discard, over time, Relational DBs in favor of other “modern replacement” DBs, given their deep-rooted popularity, even in microservices.*

5. Threats to Validity

In this section, we discuss limitations and threats that may affect the validity of our study and how we mitigated them.

5.1. Construct validity

Threats to construct validity concern the relation between theory and observation. Our identification of microservice repositories relied on heuristics such as keywords, programming languages, and structural indicators. While these heuristics were carefully designed, some false positives and negatives may persist, potentially including non-microservices and excluding relevant repositories that do not explicitly match some of the criteria.

To partially mitigate this threat, multiple conditions must be satisfied for the repository to be included in the dataset, with more emphasis on precision rather than recall, ensuring the quality of our dataset as future benchmark.

Another threat is the implicit exclusion of DB technologies that lack Docker images, as the presence of Docker files was used as a distinguishing criterion. Although these exclusions are limited and do not involve the most popular DBs, they may still slightly affect the comprehensiveness of our results. The historical perspective presented in RQ4 is based on a fixed snapshot and considers only the age of the repository computed from the creation date. While this offers initial insights, a more fine-grained analysis of the actual evolution history of repositories would provide a better understanding of trends.

Finally, our analyses considered all DBs declared in Docker Compose files, regardless of whether they were actively used in the codebase. This could lead to the inclusion of unused or “ghost” dependencies. A more in-depth approach, for example leveraging static program analysis, would be necessary to confirm the actual usage of declared DBs in the application code.

5.2. Internal validity

Internal validity concerns how one can be confident on claimed cause and effect relation. We do not claim any causation in our study. We analyze the (joint) usage of DB technologies in microservices applications, and assess possible correlations with system complexity and age. Hence, this study is not subjected to threats to internal validity.

5.3. External validity

External validity concerns the generalizability of findings beyond the study context. Our study considers different types of projects in terms of application domain, size, complexity, programming languages, and DB technologies. Although we ensured to collect an heterogeneous sample with respect to these criteria, we only considered *open-source* projects and DB technologies in public GitHub repositories. These choices affect the generalizability of our findings, yet our approach remains valid and the insights pertinent in the context of industrial software systems leveraging the same DBs.

5.4. Conclusion validity

Threats to conclusion validity concern the degree to which the statistical conclusions about the claimed relationships are reasonable. To reduce bias in results interpretation, we used standard statistical methods such as linear regression, Student’s t-test, and *p*-values.

5.5. Reliability validity

Reliability validity concerns factors that could cause an error in data collection and analysis. To minimize potential threats to reliability, we analyzed open-source projects publicly available on GitHub and provided a replication package that contains our dataset and all the scripts we used for our analyses.

6. Related Work

Most related studies focus on DBs *or* on microservices, but *separately*. In this section, we review the relevant literature to contextualize and position our work, emphasizing its contribution to bridging these two domains.

6.1. Databases

Curino *et al.* [24] propose an empirical study on DB schema changes in real-world data-intensive open-source projects, focusing on Wikipedia as a case study. They highlight the challenges of maintaining and evolving such relational DBs and provide observations based on 4.5 years of history and 171 revisions, culminating in 34 tables, 242 columns, and 700GB of data. Their goal is to illustrate typical evolution scenarios through types of changes, warn about common design errors, and offer recommendations to researchers. This work represents a step towards a unified benchmark for researchers built on a real, complex, and large case study.

In the same vein, Qiu *et al.* [25] empirically analyze the co-evolution of relational DB schemas and related source code in 10 popular and large projects composed of over 160K revisions collected from the Subversion version control system. They demonstrate the high frequency of DB changes in the software life cycle, their impact on the source code, and the types and patterns of these changes, proposing a list of DB schema change types.

Still focusing on relational DB schemas, Vassiliadis [26] studies the evolution profiles, *i.e.*, the recurring activity patterns in relational DB schema evolution. The author extracts and analyzes the schemas of 195 open-source projects from *Libraries.io*. This work provides practitioners and researchers with insights into evolution patterns to help them understand and predict such phenomena.

Goeminne and Mens [27] investigate the technical (co-)usage of DB access frameworks and object-relational mappers. They empirically analyze usage implications for 5 relational DB frameworks in 3.7K open-source projects in GitHub. The authors perform a survival analysis, observing combinations and complementarities across frameworks, especially those involving JDBC. They report that some technologies (*e.g.*, JPA, Spring) exhibit better survival rates.

Decan *et al.* [28] propose another empirical study on the use of relational DB access technologies like JDBC, Hibernate, and JPA in about 2.5K open-source Java GitHub projects. They perform a fine-grained analysis at file level, assessing technologies breakdown and the impact of their replacements on source code.

Small to medium scale case studies fail to capture general trends in a large, *in-vivo* population of real world applications. Large studies, on the contrary, focus on a limited amount of technologies, often in a single category, often relational, lacking qualities of a broad spectrum overview like in our study.

On the NoSQL side, Gessert *et al.* [12] compare several technologies, particularly for Key-Value, Document, and Column store categories. They provide practitioners with a decision tree to support choices based on functional and non-functional requirements.

Davoudian *et al.* [13] present a comprehensive survey on various NoSQL technologies, including Key-Value, Column, Document, and Graph DBs. They analyze these technologies from the perspectives of data models, consistency models, data partitioning strategies, and the CAP theorem (Consistency, Availability, and Partition tolerance). Their work includes both academic and industrial examples, providing valuable insights to assist practitioners in making informed decisions on which technology to use.

Scherzinger and Sidortschuk [29] present an empirical study on NoSQL DBs, analyzing 1.2K open-source Java projects and their GitHub history. They confirm common practices in schema-free data modeling and evolution scenarios.

These work focus on the new type of NoSQL DB categories but lack the context of the preceding and still surviving technologies. As confirmed by our findings on the collected dataset, including both SQL and NoSQL categories and technologies, the former are still far from being overthrown or obsolete.

A first step in the direction of a comparison between old and new is the work by Benats *et al.* [8], unifying relational and NoSQL data models in an empirical study investigating the use of hybrid multi-DBs over time in different languages. They consider 4 years of history across over 40K open-source projects from *Libraries.io*. We compare their empirical study with our findings (Section 4).

More recently, Paiva *et al.* [9] proposed an empirical study encompassing all data models in 362 Java open-source projects from GitHub. Studying the popularity and combinations of DB technologies, their stability, migration patterns, and the role of object-relational mappers, they provide insights to researchers and practitioners for selecting appropriate DB technologies.

In the mobile domain, Lyu *et al.* [30] investigate local DBs for Android by conducting an empirical study on 1,000 popular apps from the Google Play Store. They provide an overview of available technologies and their usage, identifying major problems and deriving recommendations for developers.

Our work compares the findings in microservices architectures with the ones in generic software systems at large, highlighting similarities and differences specific to the microservices domain.

6.2. Microservices

Brogi *et al.* [31] argue that previous works lack a reference dataset of open-source microservices projects, like a standardized benchmark. Thus, they propose μ SET, 5 microservices projects easy to set up for repeatable experiments. These projects are developed *ad hoc*, according to 5 common requirements in microservices, as identified by the authors.

Rahman *et al.* [17] propose the first dataset including real-world microservices open-source projects from GitHub. This list of 62 projects contains monoliths migrated to microservices or projects developed from scratch following the microservices architecture.

In a follow-up work, d’Aragona *et al.* [18] enlarge this dataset with 378 open-source projects from World of Code, developed in several languages. They document each project with additional data and insights helping researchers to select the most appropriate items for their work. In comparison to our work, the authors deliberately exclude the DBs from their study, focusing only on the modular nature of the microservices architectures of those systems.

Finally, Wang *et al.* [32] present a dataset of microservices applications utilizing Spring Cloud. Their contribution aims to complete previous works by suggesting complex fine-grained metrics in order to understand bad code smells in microservices.

6.3. Microservices and Databases

Only works by Gan and Delimitrou [14], Gan *et al.* [15], and Laigner *et al.* [33] propose, as benchmarks for researchers, microservices architectures repositories written in various popular languages and with different DBs. They aim to provide standard baselines for their studies and subsequent research. In all these cases, the intended benchmarks are comprised of a single to a maximum of six “*synthetic and prototypical*” end-to-end applications.

To the best of our knowledge, our empirical study is the first to bring together over 1,000 open-source microservices projects, in 11 different languages, to investigate their use of DBs across a wide variety of technologies and categories, surpassing the previous works discussed above in either size of the dataset or number and variety of considered DB technologies and categories.

7. Conclusion

We presented an empirical study on DB usage in microservices, analyzing one thousand open-source projects from GitHub developed in the last 15 years.

Our work addresses questions regarding the prevalence of DB categories and technologies used in microservices. We investigated, from several perspectives, the way DBs are combined in practice, observing recurrent patterns. We deepened our observations with objective fine-grained metrics and highlight relationships between different characteristics (*e.g.*, complexity vs. age).

Besides the “usual suspects”, we shed light on less common DBs like Time Series, Vector, Event, Graph, and Spatial DBs addressing niche goals. We highlighted a variety of Relational technologies and, overall, a variety of DBs with up to 60 unique technologies identified.

We show how microservices rely on heterogeneous DBs. Half of them use multiple DB technologies across different DB categories. Consequently, the other half uses a single technology and a single category, with unclear implications on the best strategy for database practitioners to prepare for a career path involving microservices architectures. From our analyses it emerges the large spectrum of combination patterns. Nevertheless, we try to pinpoint findings leading to practical observations and recommendations for practitioners. The 18 findings and 9 recommendations we derive are the simplest yet faceted representation of such a complex and heterogeneous reality.

Finally, we emphasize that larger microservices architectures tend to leverage more and diverse DBs. We analyze how still leading Relational DBs, shifting across the years, are now five times less prevalent in microservices, to the profit of emerging technologies such as Document, Key-Value, and Search DBs.

As a concluding remark, fundamental contributions of this work are also the dataset and the systematic approach with which we automatically built it. The published dataset is the factual basis for the reflections presented in this work and constitutes a sound starting point for future large scale research endeavors. Our work supports researchers and practitioners in understanding, evolving, and optimizing DB usage in microservices architectures.

Replication package and dataset

To ensure transparency, verifiability, and reproducibility of our work, all the artifacts resulting from our study are available at: <https://github.com/DatabaseEvolutionNudgeIMicroservices/daim>

This repository includes the MongoDB database with our complete dataset (*i.e.*, the list of GitHub repositories considered) and the scripts used to perform the analyses and to generate charts, tables, matrices, and metrics in the present work.

Acknowledgments

This work was supported by the SofinaBoël Fund for Education and Talent; the Federation Wallonie-Bruxelles (FWB), as part of the ARC project RAINDROP; and the Swiss National Science Foundation (SNSF) through the project “FORCE” (SNF Project No. 232141).

References

- [1] C. Richardson, *Microservices Patterns: with Examples in Java*, Simon and Schuster, 2018.
- [2] S. Newman, *Building Microservices*, O’Reilly Media, Inc., 2021.
- [3] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, M. Kalinowski, Data management in microservices: State of the practice, challenges, and research directions, *VLDB Endowment* 14 (13) (2021) 3348–3361. doi:10.14778/3484224.3484232.
- [4] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. A. Bernstein, P. Boncz, S. Chaudhuri, A. Cheung, A. Doan, L. Dong, M. J. Franklin, J. Freire, A. Halevy, J. M. Hellerstein, S. Idreos, D. Kossmann, T. Kraska, S. Krishnamurthy, V. Markl, S. Melnik, T. Milo, C. Mohan, T. Neumann, B. C. Ooi, F. Özcan, J. Patel, A. Pavlo, R. Popa, R. Ramakrishnan, C. Re, M. Stonebraker, D. Suciu, The Seattle report on database research, *Communications of the ACM* 65 (8) (2022) 72–79. doi:10.1145/3524284.
- [5] M. André, Automated database schema evolution in microservices, in: *Proceedings of the International Conference on Very Large Data Bases (VLDB) PhD Workshop*, Vol. 3452, CEUR-WS, 2023, pp. 37–40.
- [6] W. K. Assunção, J. Krüger, S. Mosser, S. Selaoui, How do microservices evolve? An empirical analysis of changes in open-source microservice repositories, *Journal of Systems and Software* 204 (2023) 111788. doi:10.1016/j.jss.2023.111788.
- [7] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, A. Barros, Migrating enterprise legacy source code to microservices: On multitenancy, statefulness, and data consistency, *IEEE Software* 35 (3) (2018) 63–72. doi:10.1109/MS.2017.440134612.
- [8] P. Benats, M. Gobert, L. Meurice, C. Nagy, A. Cleve, An empirical study of (multi-) database models in open-source projects, in: *Proceedings of the International Conference on Conceptual Modeling (ER)*, Springer, 2021, pp. 87–101. doi:10.1007/978-3-030-89022-3_8.

- [9] C. A. Paiva, R. Maximino, F. Paiva, R. A. Vieira, N. Espanha, J. F. Pimentel, I. Wiese, M. A. Gerosa, I. Steinmacher, L. Murta, V. Braganholo, Analyzing the adoption of database management systems throughout the history of open source projects, *Empirical Software Engineering* 30 (3) (2025) 71. doi:10.1007/s10664-025-10627-z.
- [10] U. M. Graetsch, R. Hoda, H. Khalajzadeh, M. Shahin, J. Grundy, Managing technical debt in a multidisciplinary data intensive software team: An observational case study, *Journal of Systems and Software* 230 (2025) 112546. doi:10.1016/j.jss.2025.112546.
- [11] N. Viennot, M. Lécuyer, J. Bell, R. Geambasu, J. Nieh, Synapse: A microservices architecture for heterogeneous-database web applications, in: *Proceedings of the European Conference on Computer Systems (EuroSys)*, ACM, 2015, pp. 21:1–21:16. doi:10.1145/2741948.2741975.
- [12] F. Gessert, W. Wingerath, S. Friedrich, N. Ritter, NoSQL database systems: A survey and decision guidance, *Computer Science-Research and Development* 32 (2017) 353–365. doi:10.1007/s00450-016-0334-3.
- [13] A. Davoudian, L. Chen, M. Liu, A survey on NoSQL stores, *ACM Computing Surveys (CSUR)* 51 (2) (2018) 1–43. doi:10.1145/3158661.
- [14] Y. Gan, C. Delimitrou, The architectural implications of cloud microservices, *IEEE Computer Architecture Letters* 17 (2) (2018) 155–158. doi:10.1109/LCA.2018.2839189.
- [15] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinisky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, C. Delimitrou, An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems, in: *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ACM, 2019, pp. 3–18. doi:10.1145/3297858.3304013.
- [16] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian, The promises and perils of mining GitHub, in: *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, ACM, 2014, pp. 92–101. doi:10.1145/2597073.2597074.
- [17] M. I. Rahman, S. Panichella, D. Taibi, A curated dataset of microservices-based systems, in: *Joint Proceedings of the Summer School on Software Maintenance and Evolution (SSSME)*, CEUR-WS, 2019, pp. 1–9.
- [18] D. A. d’Aragona, A. Bakhtin, X. Li, R. Su, L. Adams, E. Aponte, F. Boyle, P. Boyle, R. Koerner, J. Lee, F. Tian, Y. Wang, J. Nyssölä, E. Quevedo, M. S. Rahaman, A. S. Abdelfattah, M. Mäntylä, T. Cerný, D. Taibi, A dataset of microservices-based open-source projects, in: *Proceedings of the International Conference on Mining Software Repositories (MSR)*, ACM, 2024, pp. 504–509. doi:10.1145/3643991.3644890.
- [19] H. Borges, M. T. Valente, What’s in a GitHub star? Understanding repository starring practices in a social coding platform, *Journal of Systems and Software* 146 (2018) 112–129. doi:10.1016/j.jss.2018.09.016.
- [20] M. Schmidt, The Sankey diagram in energy and material flow management. Part II: Methodology and current applications, *Journal of Industrial Ecology* 12 (2) (2008) 173–185. doi:10.1111/j.1530-9290.2008.00015.x.
- [21] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, C. Delimitrou, Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices, in: *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ACM, 2019, pp. 19–33. doi:10.1145/3297858.3304004.
- [22] J. Soldani, D. A. Tamburri, W.-J. Van Den Heuvel, The pains and gains of microservices: A systematic grey literature review, *Journal of Systems and Software* 146 (2018) 215–232. doi:10.1016/j.jss.2018.09.082.
- [23] A. Bakhtin, M. Esposito, V. Lenarduzzi, D. Taibi, Network centrality as a new perspective on microservice architecture, in: *Proceedings of the International Conference on Software Architecture (ICSA)*, IEEE, 2025, pp. 72–83. doi:10.1109/ICSA65012.2025.00017.
- [24] C. A. Curino, H. J. Moon, L. Tanca, C. Zaniolo, Schema evolution in Wikipedia: Toward a web information system benchmark, in: *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*, INSTICC, 2008, pp. 323–332. doi:10.5220/0001713003230332.
- [25] D. Qiu, B. Li, Z. Su, An empirical analysis of the co-evolution of schema and code in database applications, in: *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE)*, ACM, 2013, pp. 125–135. doi:10.1145/2491411.2491431.
- [26] P. Vassiliadis, Profiles of schema evolution in free open source software projects, in: *Proceedings of the International Conference on Data Engineering (ICDE)*, IEEE, 2021, pp. 1–12. doi:10.1109/ICDE51399.2021.00008.
- [27] M. Goeminne, T. Mens, Towards a survival analysis of database framework usage in Java projects, in: *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2015, pp. 551–555. doi:10.1109/ICSM.2015.7332512.

- [28] A. Decan, M. Goeminne, T. Mens, On the interaction of relational database access technologies in open source Java projects, arXiv:1701.00416 (2017). doi:10.48550/arXiv.1701.00416.
- [29] S. Scherzinger, S. Sidortschuck, An empirical study on the design and evolution of NoSQL database schemas, in: Proceedings of the International Conference on Conceptual Modeling (ER), Springer, 2020, pp. 441–455. doi:10.1007/978-3-030-62522-1_33.
- [30] Y. Lyu, J. Gui, M. Wan, W. G. Halfond, An empirical study of local database usage in Android applications, in: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2017, pp. 444–455. doi:10.1109/ICSME.2017.75.
- [31] A. Brogi, A. Canciani, D. Neri, L. Rinaldi, J. Soldani, Towards a reference dataset of microservice-based applications, in: Proceedings of the International Conference on Software Engineering and Formal Methods (SEFM), Springer, 2017, pp. 219–229. doi:10.1007/978-3-319-74781-1_16.
- [32] W. Yang, B. Song, Y. Xing, Y. Lyu, H. Cui, Z. Liang, Z. Tu, A feature dataset of microservices-based systems, in: Proceedings of the International Conference on Service Science (ICSS), Springer, 2024, pp. 73–87. doi:10.1007/978-981-97-5760-2_6.
- [33] R. Laigner, Z. Zhang, Y. Liu, L. F. Gomes, Y. Zhou, Online Marketplace: A benchmark for data management in microservices, Proceedings of the ACM on Management of Data 3 (1) (2025) 3:1–3:26. doi:10.1145/3709653.
- [34] Wikipedia contributors, Comparison of object database management systems — Wikipedia, the free encyclopedia, [Online; accessed 2025-10-15] (2024).
URL https://en.wikipedia.org/w/index.php?title=List_of_relational_database_management_systems&oldid=1277025704
- [39] Wikipedia contributors, Comparison of object database management systems — Wikipedia, the free encyclopedia, [Online; accessed 2025-10-15] (2024).
URL https://en.wikipedia.org/w/index.php?title=Comparison_of_object_database_management_systems&oldid=1238176727
- [40] Wikipedia contributors, Document-oriented database — Wikipedia, the free encyclopedia, [Online; accessed 2025-10-15] (2025).
URL https://en.wikipedia.org/w/index.php?title=Document-oriented_database&oldid=1278338966
- [41] Wikipedia contributors, List of column-oriented DBMSes — Wikipedia, the free encyclopedia, [Online; accessed 2025-10-15] (2024).
URL https://en.wikipedia.org/w/index.php?title=List_of_column-oriented_DBMSes&oldid=1241929445
- [42] Wikipedia contributors, List of in-memory databases — Wikipedia, the free encyclopedia, [Online; accessed 2025-10-15] (2025).
URL https://en.wikipedia.org/w/index.php?title=List_of_in-memory_databases&oldid=1269217579
- [43] Wikipedia contributors, NoSQL — Wikipedia, the free encyclopedia, [Online; accessed 2025-10-15] (2025).
URL <https://en.wikipedia.org/w/index.php?title=NoSQL&oldid=1277556972>

Web references

- [34] M. Fowler, Polyglot persistence, [Online; accessed 2025-10-15] (2011).
URL <https://martinfowler.com/bliki/PolyglotPersistence.html>
- [35] J. Lewis, M. Fowler, Microservices, [Online; accessed 2025-10-15] (2014).
URL <https://martinfowler.com/articles/microservices.html>
- [36] Red Gate Software Ltd, DB-Engines ranking — Popularity ranking of DBMS, [Online; accessed 2025-03-06] (March 2025).
URL <https://db-engines.com/en/ranking>
- [37] Carnegie Mellon Database Group, Browse — Database of Databases, [Online; accessed 2025-03-06] (March 2025).
URL <https://dbdb.io/browse>
- [38] Wikipedia contributors, List of relational database management systems — Wikipedia, the free encyclopedia, [Online; accessed 2025-10-15] (2025).
- [44] Docker, Docker Hub container image library — App containerization, [Online; accessed 2025-03-06] (March 2025).
URL <https://hub.docker.com/>
- [45] GitHub, GitHub REST API documentation — GitHub docs, [Online; accessed 2025-10-15] (November 2022).
URL <https://docs.github.com/en/rest?apiVersion=2022-11-28>
- [46] GitHub, Octoverse: AI leads Python to top language as the number of global developers surges — The GitHub Blog, [Online; accessed 2025-10-15] (October 2024).
URL <https://github.blog/news-insights/octoverse/octoverse-2024/>
- [47] Red Gate Software Ltd, Historical trend of relational DBMS popularity, [Online; accessed 2025-03-06] (May 2025).
URL https://db-engines.com/en/ranking_trend/relational+dbms

- [48] Red Gate Software Ltd, Historical trend of document DBMS popularity, [Online; accessed 2025-03-06] (May 2025).
URL https://db-engines.com/en/ranking_trend/document+store
- [49] Red Gate Software Ltd, Historical trend of key-value DBMS popularity, [Online; accessed 2025-03-06] (May 2025).
URL https://db-engines.com/en/ranking_trend/key-value+store
- [50] Red Gate Software Ltd, Historical trend of wide column DBMS popularity, [Online; accessed 2025-03-06] (May 2025).
URL https://db-engines.com/en/ranking_trend/wide+column+store
- [51] Red Gate Software Ltd, Historical trend of search DBMS popularity, [Online; accessed 2025-03-06] (May 2025).
URL https://db-engines.com/en/ranking_trend/search+engine