Impact Study of NoSQL Refactoring in SkyServer Database*

Enrico Gallinucci[1,*], Matteo Golfarelli[1], Wafaa Radwan[2], Gabriel Zarate[3] and Alberto Abelló[3]

[1] *University of Bologna, Cesena, Italy*
[2] *Jawwal Telecommunications, Ramallah, Palestine*
[3] *Universitat Politècnica de Catalunya, Barcelona, Spain*

Abstract

Data modeling in NoSQL databases is notoriously complex and driven by multiple and possibly conflicting requirements. Researchers have proposed methodologies to optimize schema design of a given domain for a given workload; however, due to the agile environment in which NoSQL databases are usually employed, both domain and workload are frequently subject to changes and evolution - possibly neutralizing the benefits of optimization. When this happens, the benefits of a new optimal schema design must be weighed against the costs of migrating the data. In this work, we empirically show the benefits of schema redesign in a real publicly available database. In particular, we identify multiple snapshots (in terms of domain extension and querying workload) in the 20+ years evolution of SkyServer, demonstrate how NoSQL schema optimization at a given time can later backfire, and evaluate the conditions under which data migration becomes beneficial. This takes us to define the foundations and challenges of a framework for continuous NoSQL database refactoring, with the goal of helping DBAs and data engineers decide if, when, and how a NoSQL database should be reconsidered to restore schema design optimality.

Keywords

NoSQL database, Database refactoring, Data modeling, Data migration

1. Introduction

Database design has been studied for many years in relational databases, but its automation has not been achieved yet. Moreover, the advent of NoSQL databases since the early 2010s has just added complexity to the problem by offering alternative data models: key-value, wide-column, document-based, and graph [1]. Among these, the first three are also known as *aggregate-oriented* data models, as they encourage the modeling of tuples as complex objects, embedding all the data required to answer a query and minimizing the need to compute joins (thus avoiding the costly operation of transferring data between nodes) [1]. For this reason, the traditional domain-driven data modeling strategies typically used in relational databases [2] are abandoned in favor of workload-driven strategies, where tuples are modeled (i.e., their schema is designed) depending on the queries that the database is bound to answer.[1] Notice that we do not use the term NoSQL to name a family of tools, but a family of models, as a synonim of "co-relational" in [3], which can then be implemented in any tool, including an object-relational one like PostgreSQL.

Several research papers have proposed methodologies to obtain the optimal schema design, especially on the wide-column and document-based data models (as the key-value does not leave much room for alternative modeling strategies). As fully discussed in Section 2, these methodologies typically rely on a conceptual model (CM) of the domain (e.g., a UML class diagram) and a set of queries to be answered (a.k.a. workload). Their goal is to find a *target* database schema design that minimizes query answering times. This is often achieved by indexing, partitioning and replicating data in multiple tables (or collections) with different contents to accommodate different queries.

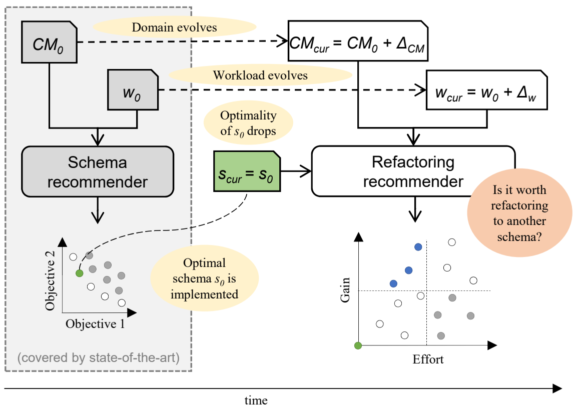

CM_0
Domain evolves
$CM_{cur} = CM_0 + \Delta_{CM}$
w_0
Workload evolves
$w_{cur} = w_0 + \Delta_w$
Optimality of s_0 drops
Schema recommender
$s_{cur} = s_0$
Refactoring recommender
Is it worth refactoring to another schema?
Objective 2
Objective 1
Optimal schema s_0 is implemented
(covered by state-of-the-art)
Gain
Effort
time

Figure 1: Intuition of the research problem

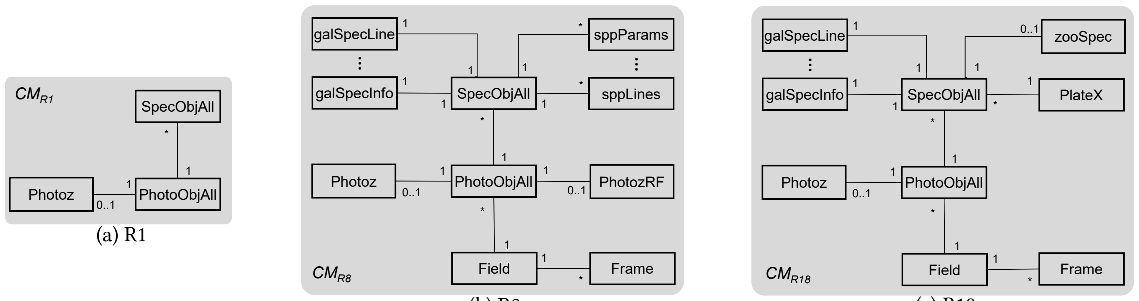
As sketched in Figure 1, we focus on what happens *next* (i.e., after a schema design has been chosen and the system/application is in production). For multiple reasons, the conditions considered at design time are continuously evolving (e.g., new data must be stored, new queries appear or they are executed at different rates), overturning the fitness of schema designs to the optimization problem. Consequently, the database should be refactored to the schema design that proves to be optimal under the new conditions. Intuitively, the *sweet spot* of interesting solutions are the ones showing the most gain with minimum effort (i.e., those in blue in Figure 1). However, refactoring a database can be costly from multiple perspectives (design and execution of the migration process in the first place) and the trade-off between the benefits of refactoring and its effort should be carefully evaluated. Moreover, the evaluation of database refactoring should not be a once-in-a-while activity: inspired by the DevOps philosophy of continuous evolution in an agile software development environment, database refactoring should be treated as a continuous problem as well. It is known that the performance of query execution can improve by migrating the corresponding data between DBMSs, even when the migration time is included [4, 5]. Our experiments, based on real astronomic data, show that, although the corresponding data migration takes some days

DOLAP 2025: 27th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2025, March 25, 2025, Barcelona, Spain

*Corresponding author.

enrico.gallinucci@unibo.it (E. Gallinucci); matteo.golfarelli@unibo.it (M. Golfarelli); wafaa.radwan@jawwal.ps (W. Radwan); gabriel.zarate@estudiantat.upc.edu (G. Zarate); alberto.abello@upc.edu (A. Abelló)

[1] In domain-driven design, workload information is used as well, but it is not the main driver.



(a) R1 (b) R8 (c) R18

Figure 2: Conceptual models of the main tables in SDSS SkyServer.

of execution, schema optimization reduces query cost by an order of magnitude, and consequently pays off in the long term (notice we are not considering here the effort of application code evolution). Thus, a structured and automated approach is even more crucial to ensure the feasibility of continuous evolution.

The main contributions we provide in this paper are:

1. An experimental setting that allows to analyze database refactoring (not considering changes in application code).
2. A detailed empirical analysis of the performance impact of schema evolution in the SkyServer database.
3. A framework proposal able to explore schema designs alternative to the current one, and give recommendations based on the evaluation of the trade-off between migration effort and the gain under different optimization criteria.

The outline of the paper is as follows. The related literature is presented in Section 2. Section 3 introduces our use case. Section 4 explains and exemplifies the motivation behind the research problem. Section 5 defines the experimental setting. Section 6 presents the evaluation of SkyServer, grounded on which we define our framework described in Section 7. Conclusions are drawn in Section 8.

2. Related work

The workload-driven nature of NoSQL data modeling has been established since the dawn of NoSQL databases [1]. Indeed, results suggest that the schema alternatives affect the database performance in different NoSQL models [6]. Over the last decade, researchers have worked to support DBAs and data engineers in the complicated task of finding the best logical model for a given workload. The most recent existing works mainly differentiate for (i) focusing on a single [7, 8, 9] or multiple data models [10, 11, 12, 13, 14], (ii) considering only the conceptual model of the data [11] or including workload queries, with [7, 12, 13, 14, 8] or without query frequencies [10, 15, 9], and (iii) directly generating one [10, 11, 14, 9] or more target schemas [15], or evaluating more of them, based on a single criterion [12, 15, 9] or multiple thereof [7, 13, 8]. The common factor between all these works is the limited focus on the *initial* design of a logical schema (i.e., none of them considers the challenge of implementing such schema by refactoring an existing one).

Research work on database evolution also started in the relational world and then propagated to the NoSQL side, where the *schemaless* characteristic[2] makes databases more easily subject to schema changes, which highly impacts their performance. Researchers have looked for patterns in the evolution of schemas in both relational [16] and NoSQL databases [17, 18] (and beyond [19]). Recent efforts to support and/or automate the management of schema evolution have been directed toward keeping track of different schema versions [20], propagating manually-defined schema modification operations (SMO) to the database [21] and to queries [22], and evaluating multiple strategies to apply schema changes to the data [23, 24]. Overall, this is still an open research field, and none of the mentioned works goes in the direction of recommending if, how, and/or when a (NoSQL) database should be refactored. Recommendations to (relational) database refactoring have been given, but mostly focused on finding and resolving issues such as inconsistencies [25] and anti-patterns [26]. More recently, [18] proposes a first approach to migration strategy planning of NoSQL databases, but still without deciding whether migrating is worth or not, or how to do it.

3. SkyServer Case study

SkyServer[3] is a publicly available relational database designed to map the cosmos, made available by the Sloan Digital Sky Survey (SDSS) organization. Over the years, it has integrated more and more data in successive extensions. Access to SDSS data is provided via a web interface, where users can query and download data through either SQL or interfaces designed for both professional astronomers and educational purposes.

Besides the astronomical data themselves, the server also makes public SkyServer Traffic Log,[4] which captures statistics on SQL queries being executed. This includes columns such as `theTime` (datetime of the query), `webserver` (URL of the server), `winname` (Windows name of the server), `clientIP` (client's IP address), and `sql` (the SQL statement executed), among others. It also captures performance metrics like `elapsed` (query execution time), `busy` (CPU time used by the query), and `rows` (number of rows returned by the query), providing a comprehensive snapshot of server activity in the last two decades (since 2003). Thus, we analyzed three different database schemas and corresponding snapshots of this log as in Release 1 (December 2003), Release 8 (December 2013), and Release 18 (December 2023). The corresponding schemas (CM_{R1}, CM_{R8}, and CM_{R18}) are summarized in Figure 2 and include changes in both

[2]The term refers to the fact that schema information is attached directly to each data item, thus imposing no constraint at the level of the table/collection of data.

[3]https://skyserver.sdss.org/dr18

[4]https://skyserver.sdss.org/log/en/traffic/sql.asp

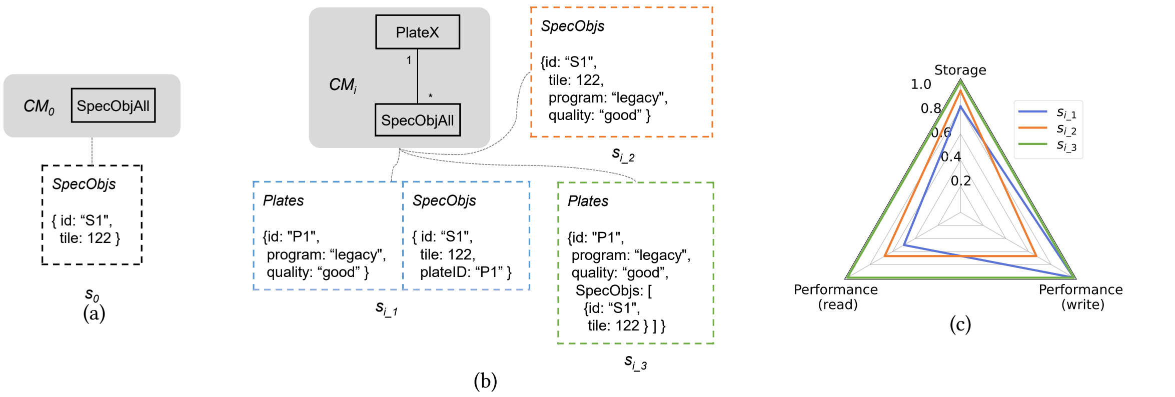


Figure 3: Example of schema evolution: (a) the initial conceptual schema CM_0 and the initial database schema s_0; (b) the evolved conceptual schema CM_i at time i and three alternative database schemas s_{i_1}, s_{i_2}, and s_{i_3}; and (c) comparison of the three potential databases on the maximization of three different objectives.

the number of tables and attributes, and how the latter are placed in the former, but without information loss.

In these schemas, we find data captured from different regions of the sky called `Fields`, where different objects are observed as `PhotoObj`. Spectroscopic data are also captured for each one of these astronomical objects, and stored per wavelength intervals into `SpecObj`. All measurements are done through aluminum `Plates` that allow to precisely plug individual spectrographs to the telescope through optical fibers.[5]

4. Motivation

In any kind of DBMS, the choice of the initial schema design is based on conditions (i.e., the conceptual representation of the domain and the estimated workload) that can change – either because they were not accurate or because they have evolved, but this is even more so in NoSQL systems. In this section, we present a couple of comprehensive and small examples to illustrate the problems, before moving to a larger one with real data that demonstrated the true impact.

4.1. Domains evolve

Plenty of research papers show database schemas need to evolve to accommodate changes in the domain (e.g., new information to be added, obsolete information to be removed, data type changes), from the '90s [27] to most recent times [28], in both relational [29, 16] and NoSQL databases [30, 31], looking for patterns in schema updates [16], studying the repercussions on the related application code [29], managing multiple schema versions [22] and designing frameworks to automate schema evolution [17]. An interesting pattern emerging from multiple research work [27, 29, 16] is that, in the early stages of the application lifespan, relational databases typically undergo an *inflation* phase, where multiple operations are carried out to add new schema information. In this sense, NoSQL databases are even more appealing due to their *schemaless* nature, which lets them easily accommodate schema additions to move on, and makes them more suitable in agile development [32]. Nevertheless, this does not mean their performance is optimum regardless of how you store data and still require reconsidering it.

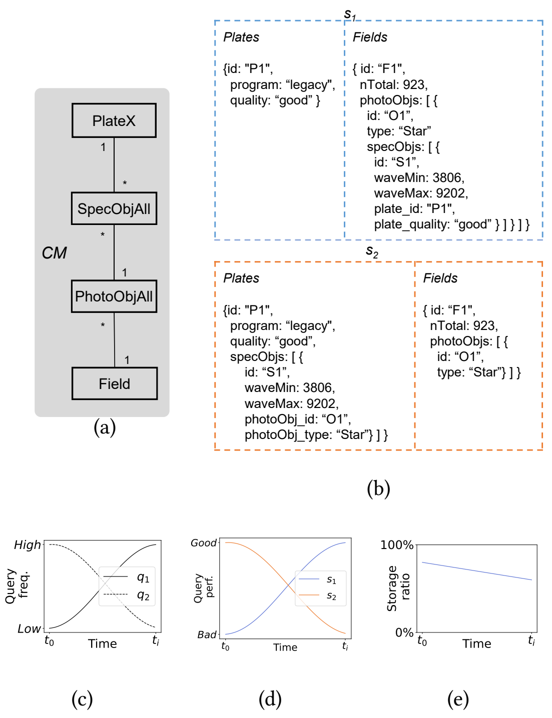
The frequency of schema changes depends on the domain and application [16]: in some cases it can be pervasive (in [27], the authors found that all the tables over the 20 analyzed databases were somehow affected by the evolution process), while in others it was completely absent (in [19], 70% of the database schemas over the 195 analyzed open source software projects demonstrate the absence or very small presence of change). Nevertheless, in the latter case, the authors verify that the absence of evolution does not mean that application requirements are static, but rather that DBAs/developers are reluctant to evolve database schemas to avoid the effort. A similar insight is found in [33], which studied schema evolution in 29 data-intensive applications using either relational or NoSQL databases. The study found that complex refactoring operations are seldom carried out, due to the lack of tools to support them.

From these studies, we conclude that: (1) it is very difficult to have a perfect understanding at design time of how schema information must be modeled; (2) the conditions to modify database schemas can mature at any time; (3) there is reluctance to change a database schema once it reaches a certain maturity level, and such changes are aimed at minimizing refactoring efforts.

As a result: (a) the updated schemas tend to be simple variations of the initial one, despite the choice of the latter being based on a significant degree of uncertainty at design time; (b) the pure minimization of refactoring efforts potentially leads to missing big opportunities hidden by the scarecrow of complex refactoring, steering instead towards possible *antipatterns*, i.e., bad practices in schema design that are intended to solve certain problems but eventually lead to other problems [34, 26] (which, in turn, will require further modifications to remodel the data).

Example 1. *An exemplification of schema evolution on a document-based database is shown in Figure 3. Let CM_0 be the initial conceptual schema with only one entity; database schema s_0 is created with a single collection of* `SpecObjAll` *(Figure 3a shows a sample document). Later on, at time i (Figure 3b), the conceptual schema evolves to CM_i to organize spectral readings into plates. To accommodate this change with minimum effort, DBAs would be inclined to evolve the database towards the schema design of s_{i_1} or s_{i_2}, but they would probably avoid s_{i_3}, even though it might be the optimal schema – as hinted by the radar chart in Figure 3c. Inspired by [35], the chart shows a comparison between the three databases in terms of the maximization of three objec-*

[5]The whole catalog of tables is available at https://skyserver.sdss.org/dr18/MoreTools/browser



s_1:

Plates

```
{id: "P1",
  program: “legacy",
  quality: “good” }
```

Fields

```
{ id: “F1",
  nTotal: 923,
  photoObjs: [ {
    id: “O1”,
    type: “Star”
    specObjs: [ {
      id: “S1”,
      waveMin: 3806,
      waveMax: 9202,
      plate_id: "P1",
      plate_quality: “good” } ] } ] }
```

s_2:

Plates

```
{id: "P1",
  program: “legacy",
  quality: “good”,
  specObjs: [ {
      id: “S1”,
      waveMin: 3806,
      waveMax: 9202,
      photoObj_id: “O1”,
      photoObj_type: “Star”} ] }
```

Fields

```
{ id: “F1",
  nTotal: 923,
  photoObjs: [ {
    id: “O1”,
    type: “Star”} ] }
```

Figure 4: Example of workload evolution: (a) the conceptual model UML; (b) two possible database schemas, s_1 (the initially chosen one) and s_2; (c) change of queries' frequency in time; (d,e) change of optimality of the two database schemas on query performance and storage occupation.

tives: storage occupation, and performance (speed) of read and write queries. For storage, we assume 30 bytes for IDs, 50 bytes for strings, 8 bytes for numbers, and a ratio of 2 products per category. To estimate query performance in this example, we used the cost model by [12] and assumed two read queries (q_1 and q_2) and two write queries (q_3 and q_4) as follows:

```
q1 = SELECT s.* FROM SpecObjAll s
     WHERE s.id = <s_id>;
q2 = SELECT s.* FROM SpecObjAll s
     JOIN PlateX p WHERE p.id = <p_id>;
q3 = UPDATE SpecObjAll s SET
     s.tile = <s_tile> WHERE s.id = <s_id>;
q4 = UPDATE PlateX p SET
     p.quality = <p_quality>
     WHERE p.id = <p_id>;
```

The indicators are normalized on a scale from 0 (worse) to 1 (best) using the complementary of the min-max normalized value. For instance, given $\varphi(s)$ as the average query execution time on schema s, and x and y as the minimum and maximum values for $\varphi(s_{i\_k})$, $k \in \{1, 2, 3\}$, query performance for the j-th schema is calculated as $\frac{y-\varphi(s_{i\_j})}{y-x}$. ♢

4.2. Workloads evolve

Similarly to schema evolution, early studies on the evolution of query workloads date back to the 80's [36] and continue to most recent times [37]. The evolution of workloads can be traced back to common patterns [38, 39].

- Changes in frequency (i.e., the same queries are executed with different frequencies and/or ratios), either with cyclic patterns (e.g., daily or monthly), with occasional spikes (e.g., due to unexpected popularity increase of the application), or more stable changes (e.g., due to new users from different time zones).
- Changes in queries (i.e., the existing queries are formulated differently, e.g., due to a change at domain level) or new/old queries are added/removed (e.g., due to the addition/removal to/from the application in the latest release).

When the initial schema design is chosen, the workload is assumed to be constant in terms of the queries using it and their corresponding frequencies. This simplification is understandable (if not essential) to choose a (sub-)optimal initial schema design. However, given the strong dependence of schema design optimality on the workload, the evolution of the latter can have a tremendous impact and ignoring it can lead to a progressive distancing from the objectives that the initial schema design was originally maximizing.

Example 2. *An exemplification of workload evolution on a document-based database is shown in Figure 4. Let CM be the initial conceptual model with four classes (Figure 4a) and two different database schemas s_1 and s_2 (Figure 4b). Assume that s_1 is the one chosen at design time, given the following workload.*

```
q1 = SELECT o.*, s.* FROM Field f
     JOIN PhotoObj o JOIN SpecObj s
     WHERE f.id = <f_id>;
q2 = SELECT p.*, s.*
     FROM Plate p JOIN SpecObj s
     WHERE p.id = <p_id>;
```

Figure 4c shows that the frequency of the two queries inverts from the initial deployment at t_0 to the one at t_i. Consequently, the optimality of the two database schemas with respect to query performance changes accordingly (Figure 4d). Storage ratio can also change (Figure 4e shows the database size in s_2 divided by the size in s_1) if the data grows unevenly. Reusing storage assumptions from Example 1, the values in the figure are calculated assuming a ratio between fields and spectral readings that grows from 1:1 to 10:1 and the average number of items per order growing from 2 to 10. ♢

5. Experimental setting

To empirically demonstrate our point, we analyzed in detail the real effect of schema evolution on SkyServer performance. For this, we considered three points in time corresponding to releases R1, R8 and R18, respectively. Firstly, we characterized the different workloads identifying the most common query patterns (i.e., ignoring mostly unique queries in the very long tail of frequencies). Then, we recreated the database at the point in time of each of the three releases and populated them with a sample of the data available in SkyServer. Finally, we measured both the cost of queries in each schema as well as the cost of moving the data from one to another. All tests have been executed on a PostgreSQL 15 instance, running on a server with an i7-8700 CPU and 64 GB of RAM. To guarantee reproducibility, all the corresponding code is publicly available in GitHub.[6]

In the following, we use numbers (1, 2, and 3) to refer to database schemas in different points in time, and Greek letters (α, β, and γ) to refer to the corresponding workloads. A summary of the experimental setup is shown in Figure 5 and detailed in the following sections; in the figure, the yellow area indicates the database schemas over which each workload is executed.

[6]https://github.com/enricogallinucci/nosql-refactoring

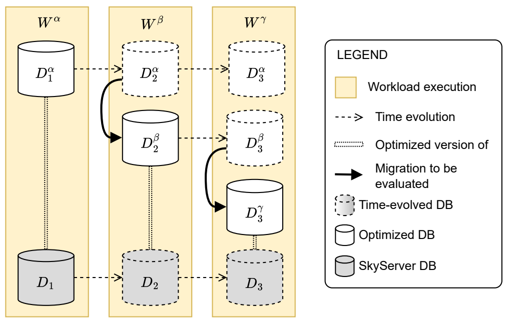


Figure 5: DBs and workloads in our experimental setup

5.1. Schemas

The schema of each database in Figure 5 is generated as follows. **Original databases** (D_1, D_2, D_3) are populated exactly as provided by SDSS. Nevertheless, to make the performance comparable after optimization, we did not implement classic 1NF, but encoded all them into a flat JSON document (without any subdocument or array) that was then stored in a relational table in PostgreSQL. **Optimized databases** (D_1^α, D_2^β, and D_3^γ) are optimized for the workload in the corresponding point in time following the hints in [12]. More concretely, to decide on join materialization, as well as vertical and horizontal partitioning, we:

1. Followed a greedy algorithm taking each query in the order indicated by their criticality (i.e., queries with higher value for the product of their frequency and cost were considered first) and:
   (i) Created a JSON document per row considering all the tables involved in the query (i.e., join materialization or embedding).
   (ii) Took from each source table only the attributes necessary for the query (i.e., vertical partitioning).
   (iii) Applied filters of the query (i.e., horizontal partitioning).

   Notice that (a) once a critical query has generated some optimization, this is not undone by less critical queries, and (b) we allowed *intra-table* redundancies (e.g., materializing the many-to-one join between `Frame` and `Field`, which replicates field data for every frame), but not *inter-table* ones (i.e., once we materialize the join between `PhotoObjAll` and `Photoz` in a single table, we do not create another standalone copy of the later).
2. Generated a separate vertical partition of the corresponding table to store all attributes not used in any query.
3. Created secondary indexes for any attribute in the selection predicates for both the original as well as optimized schemas.

Time-evolved databases (D_2^α, D_3^α, and D_3^β) correspond exactly to databases optimized for an obsolete workload, but extended with the concepts introduced in the new release in the form of one extra table per new concept, so all queries can be executed. For instance, D_2^α is the time-evolution of D_1^α, which preserves the pre-existing data and optimizations for workload α, but adds the flat new tables introduced by D_2. These schemas are crucial to put optimizations under the test of time and evaluate whether effectiveness is held upon workload evolution or if it would be more convenient to migrate the data to the schema optimized for the new workload.

Work.	Queries per month	Query freq. (Hz)	N. query patterns by output		
			1 row (stable)	>1 rows (stable)	>1 rows (scaling)
W^α	48,337	0.02	2	3	-
W^β	1,352,498	0.57	6	3	-
W^γ	3,485,018	1.39	6	2	7

Table 1
Workload statistics

5.2. Data

The overall size of the SkyServer database (in its latest release) is approximately 5TB and cannot be directly downloaded; consequently, we proportionally sampled the source to manage it more effectively. Sampling is based on the main table, `PhotoObjAll`, which originally contains 1.2 billion rows: we collected 4 samples of 100K, 200K, 500K, and 1M rows, ensuring that samples preserve the distribution of attributes involved in the selection predicates; then, other tables are populated with the rows linked to the ones sampled on `PhotoObjAll`. To make the performance comparable across the releases, we made the four samples of the same size, independently of the size of the database at the point in time of the release.

5.3. Workload

The workload of every release was extracted from the `SqlLog` table of the SkyServer Traffic Log for December of the corresponding year, excluding queries that were unsuccessful or involving customer user tables. Given the nature of the service, we should notice that users are not allowed to modify the database, hence, the log contains only read queries. After parsing the queries, we extracted (1) the tables involved, (2) the columns projected, and (3) the selection predicate. Firstly, the queries were clustered based on the tables they required, and a minimum threshold of 1% was fixed for the cluster to be further considered. These initial clusters were then subdivided depending on the columns projected and selection predicate used, filtering out subclusters with less than 0.5% queries, for a final count of 5, 23, and 21 clusters being considered for each release. Since we wanted to evaluate changes from one release to another, out of those clusters, we generated query patterns only for those involving tables present in more than one release.

Statistics of the final workloads, including overall query frequency (assuming uniform distribution in time) and a characterization of the included query patterns (based on the number of returned rows), are reported in Table 1.

6. Experimental evaluation

In our experiments, we first look at the space being used, then the execution time of the query workload, and finally, the cost of migrating from one schema to another.

6.1. Evaluation of storage occupation

Table 2 shows the total storage occupation (in MB) of each database schema on every scale. Intuitively, the storage increases proportionally with the scale - though this is less evident in D_2^* and D_3^* due to some tables (`Field` and `Frame`) being independent from `PhotoObjAll`. Interestingly, the

Scale	D_1	D_1^α	D_2	D_2^α	D_2^β	D_3	D_3^α	D_3^β	D_3^γ
100K	683	704	849	875	853	885	892	882	878
200K	1365	1408	1600	1652	1607	1672	1685	1666	1665
500K	3412	3520	3852	3983	3869	4035	4065	4019	4028
1M	6824	7039	7607	7868	7640	7973	8033	7941	7965

Table 2
Storage occupation (in MB) of database schemas

Scale	W^α		W^β			W^γ			
	D_1	D_1^α	D_2	D_2^α	D_2^β	D_3	D_3^α	D_3^β	D_3^γ
100K	73.8	7.6	27.8	63.7	2.0	145.9	571.2	269.7	45.5
200K	71.3	5.4	26.3	60.2	1.6	285.4	1124.4	490.9	60.7
500K	71.3	5.4	25.5	60.2	1.6	714.5	1546.4	1134.6	125.5
1M	77.9	5.4	27.8	63.7	1.0	1125.2	3249.2	2304.3	240.6

Table 3
Average execution time (in ms) of a single query

applied optimizations have no significant impact on storage, due to the absence of inter-table redundancies and to a low footprint of intra-table ones.

6.2. Evaluation of query execution times

As shown in Figure 5, the three workloads are executed over the database schemas available for the corresponding point in time (i.e., W^α, W^β, and W^γ are respectively executed over the D_1^*, D_2^*, and D_3^* versions). For each combination of workload and database schema, 11K queries have been executed by preserving the frequency of each query pattern and randomly choosing values (among the existing ones) for the selection predicates; the first 1000 queries are then discarded to minimize the impact of cold-start on the cache. Table 3 shows the average execution time (in ms) of a single query, by weighing the average execution time of each query pattern on the respective query frequency. From the results, we can make the following observations.

- The workload changes significantly across releases. This is evident not only on the average execution time of a single query (which considerably grows on W^γ), but also in the variation over different scales: as shown in Table 1, W^γ includes query patterns where the number of rows scales with the cardinality of tables, while these are not present in W^α and W^β.
- The random choice of selection predicates slightly impacts on the average execution times, especially when these are particularly low. For example, it may seem that execution times improve with the database size in D_2^β; however, the standard deviation in this case ranges from 1.9 to 2.8 in all scales for this database schema, so the variation is clearly not statistically significant.
- Optimizations have a huge impact on performances, with reductions of execution times ranging from 3 to 10 times across all workloads. This provides a solid justification for the need to implement optimized database schemas - also in light of the essentially unvaried storage occupation.
- Interestingly, optimizations carried out at a specific point in time do not outlive the workload and end up backfiring at later stages. As the characteristics of the workloads evolve, execution times sensibly increase due to previous optimizations losing effectiveness and becoming a liability. This nicely demonstrates the need for a continuous re-evaluation of database optimizations.

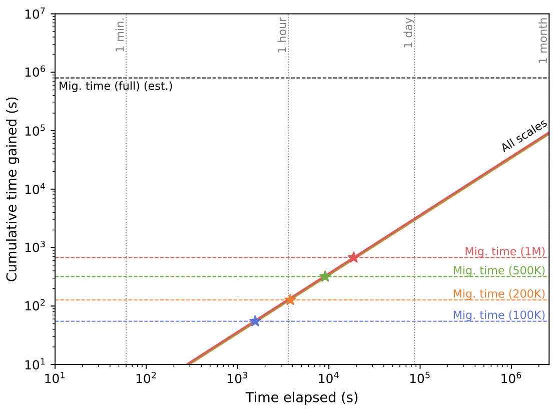


Figure 6: Study of migration convenience from D_2^α to D_2^β

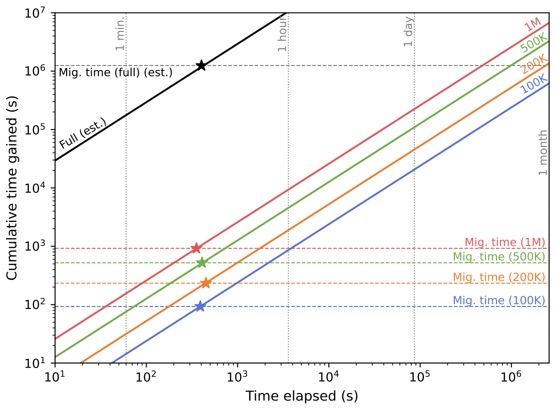


Figure 7: Study of migration convenience from D_3^β to D_3^γ

Migration	Scale	Time gain per query (ms)	Mig. time (ms)	N. queries to pass mig. time	Time to pass mig. time (min)
D_2^α to D_2^β	100K	62	55,100	893	25.9
	200K	59	127,100	2,172	63.0
	500K	61	319,200	5,236	151.9
	1M	63	675,000	10,761	312.2
D_3^β to D_3^γ	100K	172	93,000	542	6.5
	200K	378	233,300	618	7.4
	500K	910	521,900	574	6.8
	1M	1,867	922,000	494	5.9

Table 4
Migration statistics

6.3. Evaluation of migration convenience

Finally, we study the convenience of database migration; as seen in Figure 5, we focus on the migrations from D_2^α to D_2^β and from D_3^β to D_3^γ (i.e., migrating data from R8 optimized for the old R1 workload, to another schema optimized for the true R8 workload; similarly for R8 and R18).

Migration convenience is evaluated by measuring the gain obtained in query performance (due to database re-optimization) against the effort taken to migrate the data. Both factors are measured in terms of time: the gain is the difference in the average execution time of two database schemas, and effort is the time required to execute migration scripts. Then, the migration becomes convenient when the (cumulated) gain overcomes the effort. Table 4 summarizes the results on all sample sizes and indicates the number of queries needed to accumulate enough gain to overcome the migration effort; the same is also translated into a measure of time, based on the query frequency in the real workload, as in Table 1. The results are also reported in Figures 6 and 7, which emphasize trends over logarithmic time scales: for each sample size (identified by a different color), the cumulative time gained is shown as time elapses, migration time is shown as a flat horizontal threshold, and a star marks the turning point. Linear regression is used to estimate gain and effort on the full database size (shown as black lines).

From these results, we derive the following takeaways.

- First, we observe that migration time is proportional to the database size. As discussed in Section 6.1, this is not surprising given the low-to-no impact of replication. What is remarkable is the estimated migration time on the full scale, which achieves the order of multiple days. Though this estimate could be easily optimized by parallelizing the migration of the different tables, it shows the importance of considering workload prediction in the refactoring recommendation: the longer the time to migrate the data, the longer the required stability of the workload (or accuracy of the prediction) to ensure that the migration pays off.
- Since execution times for W^{β} are unaffected by the database size (as discussed in Section 6.2), the gain is almost identical across all scales. As a result, the bigger the database, the more time it takes to accumulate enough gain to compensate for migration times. Differently, in W^{γ}, query execution times grow with the database size, thus the gain scales accordingly. As a result, the migration becomes convenient after only 6-7 minutes, independently of the database size.
- Interestingly, the two studies reveal radically different scenarios where the recommendations to carry out database refactoring are diverse. In the samples, database migration is always fast and particularly convenient. In our projected estimates over the full database, the migration to D_2^{β} would be discouraged under the assumption that the workload significantly differs in the following month; differently, the migration to D_3^{γ} is shown to be convenient, even though the implications of the considerable migration time should be carefully considered before enacting the refactoring.

7. Framework overview

As we have just demonstrated, the evolution of schemas and workloads can dramatically change the optimality of the schema design chosen at design time, and refactoring the database can restore such optimality. The newly-optimal schema should be found as the one maximizing the trade-off between the benefits of a refactoring and the effort to design and execute it. This task opens to multiple research challenges, including the exploration of the search space of target schemas (potentially scaling to thousands of concepts), the quantification (and comparison) of the benefits from refactoring the database and the efforts to design and execute the refactoring, and the prediction of changes in the workload (potentially including millions of queries). As shown in Section 2, related work mainly explore the identification of the first target solution [40, 41, 42, 7, 8, 43, 10, 11, 12, 13, 14] or devise frameworks to manage multiple schema versions and propagate manually-defined schema transformations to the database

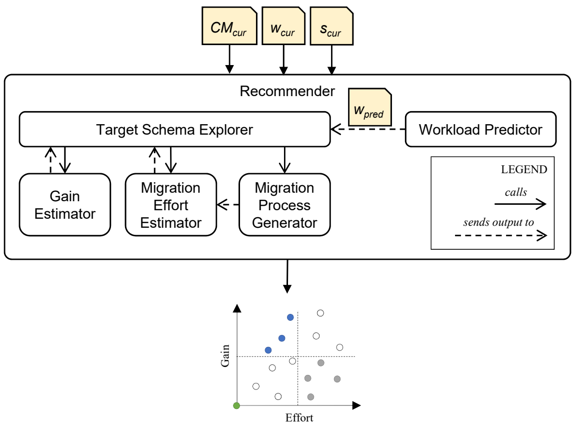


Figure 8: Overview of the proposed framework for continuous refactoring evaluation

[20, 24, 31, 21, 22, 23, 24], but do not address these research questions. In this section, we pave the way in this direction, defining a framework and illustrating some novel research challenges that this should address to achieve this goal.

The outcomes of the experiments evidently suggest that (i) database optimization is neither a one-time nor an incremental activity, as some optimizations can become counterproductive in future stages, (ii) database migration can be particularly expensive and may not be worth the trouble, and (iii) as a result, the continuous evaluation of refactoring options is fundamental to guarantee maximum efficiency under evolving workloads. Hence, we propose a multi-objective optimization to continuously evaluate and recommend the refactoring of NoSQL databases.

An overview of the proposed framework is shown in Figure 8. The main component is the Target Schema Explorer, which is in charge of enumerating and evaluating the possible target schemas. The enumeration requires the current schema s_{cur}, the current conceptual model CM_{cur}, and either the current workload w_{cur} or a prediction of a future workload w_{pred}, calculated by the Workload Predictor. Given a possible target schema s_i, its evaluation is aimed at quantifying the pros and cons of carrying out the refactoring from s_{cur}. The pros are calculated by the Gain Estimator, which measures the variations of multiple quality criteria (in terms of performance, storage occupation, etc.). The cons are calculated by the Effort Estimator, which measures the work required to carry out the refactoring (in terms of designing and executing the migration process, rewriting all workload queries, etc.). The latter estimation requires as much information as possible about the migration process, which is produced by the Migration Process Generator component. Finally, the Recommender obtains from the Target Schema Explorer the list of evaluated target schemas and produces a recommendation; given the amount and diversity of criteria to measure gains and efforts, the Recommender determines the set of relevant target schemas on the Pareto front [44].

The benefits of advancing the state-of-the-art in this direction are twofold. On the one hand, the Recommender can provide critical insights to make refactoring decisions with significant improvements to the current situation based on objective criteria and a comprehensive coverage of possible alternatives. On the other hand, the automation of this task enables its continuous adoption through the lifetime of the database and the applications running on top of it; indeed, a continuous evaluation of database refactoring minimizes the risk of undergoing major efforts at a later time to recover from a degraded state. In both cases, complete automation is hard to achieve, as the precise measurement of gains and efforts is particularly challenging and the selection of the "best" refactoring activity from the Pareto front requires business knowledge and strategic vision (i.e., skills that cannot be easily quantified and encoded). Thus, our proposal goes in the direction of *human-in-the-loop automation*: while we turn to the DevOps philosophy in the continuous application of an automated procedure to maintain a high-quality level of the database, DBAs/engineers should be able to step in at critical points to contribute with their knowledge and exploit the system to make decisions and decide the path forward.

In the following, we delve into the details of each of the framework's components, discussing current implementations and presenting the research challenges that are yet to be addressed to achieve automation.

Target schemas exploration. The aggregate-data modeling style of NoSQL databases implies a huge search space of alternative schemas that could be devised in a given domain [45]. [46] shows that there are 12 different ways to logically model a conceptual relationship between two entities in a document-based database. This search space is further amplified by the practice of replicating data in multiple collections to optimize the performance of the most frequent queries (we avoided this possibility in our experiments to keep them simpler); thus, an exhaustive generation and evaluation of all possibilities is prohibitive. The challenge is worsened by the absence of a single optimization metric to drive the exploration towards convergence. In the related work, the most common approach to schema exploration (as well as the one followed in this study) simply consists in converging to a target schema through some heuristics (e.g., [12]). However, we see huge potential behind multi-objective evolutionary algorithms (MOEAs), which are particularly suitable for the task of finding Pareto-optimal solutions [47] but not yet adopted in this context.

Gain estimation. The Gain Estimator relies on a set of Key Performance Indicators (KPIs) to quantify the (dis)advantages of migrating from the current schema to a different one under many perspectives, namely **Performance** (query execution time is crucial in NoSQL), **Storage** (redundancy is typically encouraged, but updates should not be forgotten), and **Complexity** (schemaless allows quick development, but also hides mistakes in the coding). In this study, we focused on the performance evaluation, but measured it empirically. Thus far, the proposed metrics for estimations are either oversimplistic (e.g., [12] considers the number of accessed documents) or too narrow (e.g., [48] predicts execution times using an advanced database-specific model, but limitedly to point-queries on the primary key).

Migration effort estimation. Similarly to the gain, the estimation of the migration effort can be measured from multiple perspectives, namely **Process design** (whose cost would depend on the complexity and extension of the model), **Execution** (which should consider the impact over the currently-running workload), and **Application update** (especially relevant due to schemaless philosophy in NoSQL). For this, we can build on top of recent software refactoring work [49] and ETL evolution [50].

Migration process generation. The proper estimation of migration efforts also depends on how well the migration process can be predicted and how much it can be automated. The information returned by this component can be at several levels, namely **conceptual** (e.g., identifying which entities are involved in the migration), **logical** (i.e., defining the sequence of operations that should be carried out to migrate the data), and **physical** (i.e., producing the scripts or application code to be executed). In any case, an optimizer should be used to make the process as efficient as possible. In our study, the process was generated and optimized manually, but automation is clearly necessary. Some proposals in this direction have been made, but they only support a limited range of schema modification operations [51] and are tied to table-to-table (or collection-to-collection) mappings [52], whereas the migration of the database needs to be considered as a whole.

Workload prediction. The capability of the recommender to operate on a predicted future workload is a *bonus* feature, in the sense that the recommendation could also be given just by considering the current workload. Nonetheless, given the (possibly considerable) effort to do a migration and the (possibly continuous) evolution of the workload, the optimality of the new target schema may be lost by the time that the migration is completed – as the evidence of this study has shown. For this reason, predicting (with sufficient accuracy) what the workload will be at time $i + \Delta$ allows the recommender to consider an additional variable and to possibly converge towards the optimal solutions that require that Δ time to carry out the migration. The prediction of the evolution of workload queries is a field that has recently attracted research interest [37]. However, existing works are limited to relational databases and mostly focused on supporting a live tuning of the DBMS's configuration and/or resources [53].

8. Conclusions

In this paper, we have presented an impact study of NoSQL database refactoring over a real-world use case, motivating the research problem, supporting it with empirical evidence, and presenting a proposal for a refactoring recommender framework. Our research work will continue under two directions. On the one hand, we plan to further investigate the SkyServer use case to consider additional strategies for schema optimizations and to incorporate workload prediction into the migration convenience evaluation; by collecting additional information about the workload in the upcoming months, we will put the proposed optimizations under a more comprehensive test of time. On the other hand, we will work towards the implementation and automation of the proposed framework. Each module in the framework encompasses its own challenges, which can be addressed separately. Our main efforts will be first directed towards enabling a broad exploration of target schemas and defining a comprehensive method for estimating migration efforts considering the many variables that influence this process, including application code evolution and human effort estimation. We plan to work in close collaboration with companies dealing with evolving workloads in NoSQL databases and whose support is already shown in previous work on heterogeneous and evolving datasets [54, 55].

Acknowledgments

This work has been partially supported by the Spanish Ministerio de Ciencia e Innovación under project PID2020-117191 RB-I00/AEI/10.13039/501100011033 (DOGO4ML).

References

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

[2] J. L. Harrington, Relational database design and implementation, Morgan Kaufmann, 2016.

[3] E. Meijer, A co-relational model of data for large shared data banks, in: M. Mezini (Ed.), ECOOP 2011 - Object-Oriented Programming - 25th European Conference, Lancaster, UK, July 25-29, 2011 Proceedings, volume 6813 of *Lecture Notes in Computer Science*, Springer, 2011, p. 1. URL: https://doi.org/10.1007/978-3-642-22655-7_1. doi:10.1007/978-3-642-22655-7_1.

[4] V. Gadepally, P. Chen, J. Duggan, A. J. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, M. Stonebraker, The bigdawg polystore system and architecture, in: 2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016, IEEE, 2016, pp. 1–6. URL: https://doi.org/10.1109/HPEC.2016.7761636. doi:10.1109/HPEC.2016.7761636.

[5] R. Alotaibi, D. Bursztyn, A. Deutsch, I. Manolescu, S. Zampetakis, Towards scalable hybrid stores: Constraint-based rewriting to the rescue, in: P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, T. Kraska (Eds.), Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, ACM, 2019, pp. 1660–1677. URL: https://doi.org/10.1145/3299869.3319895. doi:10.1145/3299869.3319895.

[6] S. S. Neha Bansal, L. K. Awasthi, Are nosql databases affected by schema?, IETE Journal of Research 70 (2024) 4770–4791. URL: https://doi.org/10.1080/03772063.2023.2237478. doi:10.1080/03772063.2023.2237478.

[7] V. Reniers, D. V. Landuyt, A. Rafique, W. Joosen, A workload-driven document database schema recommender (DBSR), in: G. Dobbie, U. Frank, G. Kappel, S. W. Liddle, H. C. Mayr (Eds.), Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings, volume 12400 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 471–484. URL: https://doi.org/10.1007/978-3-030-62522-1_35. doi:10.1007/978-3-030-62522-1_35.

[8] M. Hewasinghage, S. Nadal, A. Abelló, E. Zimányi, Automated database design for document stores with multicriteria optimization, Knowl. Inf. Syst. 65 (2023) 3045–3078. URL: https://doi.org/10.1007/s10115-023-01828-3. doi:10.1007/s10115-023-01828-3.

[9] M. Mozaffari, E. Nazemi, A. Eftekhari-Moghadam, CONST: continuous online nosql schema tuning, Softw. Pract. Exp. 51 (2021) 1147–1169. URL: https://doi.org/10.1002/spe.2945. doi:10.1002/SPE.2945.

[10] A. de la Vega, D. García-Saiz, C. Blanco, M. E. Zorrilla, P. Sánchez, Mortadelo: Automatic generation of nosql stores from platform-independent data models, Future Gener. Comput. Syst. 105 (2020) 455–474. URL: https://doi.org/10.1016/j.future.2019.11.032. doi:10.1016/j.future.2019.11.032.

[11] J. Mali, F. Atigui, A. Azough, N. Travers, Modeldrivenguide: An approach for implementing nosql schemas, in: S. Hartmann, J. Küng, G. Kotsis, A. M. Tjoa, I. Khalil (Eds.), Database and Expert Systems Applications - 31st International Conference, DEXA 2020, Bratislava, Slovakia, September 14-17, 2020, Proceedings, Part I, volume 12391 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 141–151. URL: https://doi.org/10.1007/978-3-030-59003-1_9. doi:10.1007/978-3-030-59003-1_9.

[12] L. Chen, A. Davoudian, M. Liu, A workload-driven method for designing aggregate-oriented nosql databases, Data Knowl. Eng. 142 (2022) 102089. URL: https://doi.org/10.1016/j.datak.2022.102089. doi:10.1016/J.DATAK.2022.102089.

[13] E. M. Kuszera, L. M. Peres, M. D. D. Fabro, Exploring data structure alternatives in the RDB to nosql document store conversion process, Inf. Syst. 105 (2022) 101941. URL: https://doi.org/10.1016/j.is.2021.101941. doi:10.1016/j.is.2021.101941.

[14] N. Roy-Hubara, A. Sturm, P. Shoval, Designing nosql databases based on multiple requirement views, Data Knowl. Eng. 145 (2023) 102149. URL: https://doi.org/10.1016/j.datak.2023.102149. doi:10.1016/j.datak.2023.102149.

[15] W. Y. Mok, A conceptual model based design methodology for mongodb databases, in: 7th International Conference on Information and Computer Technologies, ICICT 2024, Honolulu, HI, USA, March 15-17, 2024, IEEE, 2024, pp. 151–159. URL: https://doi.org/10.1109/ICICT62343.2024.00030. doi:10.1109/ICICT62343.2024.00030.

[16] I. Skoulis, P. Vassiliadis, A. V. Zarras, Growing up with stability: How open-source relational databases evolve, Inf. Syst. 53 (2015) 363–385. URL: https://doi.org/10.1016/j.is.2015.03.009. doi:10.1016/j.is.2015.03.009.

[17] S. Scherzinger, S. Sidortschuck, An empirical study on the design and evolution of nosql database schemas, in: G. Dobbie, U. Frank, G. Kappel, S. W. Liddle, H. C. Mayr (Eds.), Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings, volume 12400 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 441–455. URL: https://doi.org/10.1007/978-3-030-62522-1_33. doi:10.1007/978-3-030-62522-1_33.

[18] S. Fedushko, R. Malyi, Y. Syerov, P. Serdyuk, Nosql document data migration strategy in the context of schema evolution, Data & Knowledge Engineering 154 (2024) 102369. URL: https://www.sciencedirect.com/science/article/pii/S0169023X24000934. doi:https://doi.org/10.1016/j.datak.2024.102369.

[19] P. Vassiliadis, Profiles of schema evolution in free open source software projects, in: 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021, IEEE, 2021, pp. 1–12. URL: https://doi.org/10.1109/ICDE51399.2021.00008. doi:10.1109/ICDE51399.2021.00008.

[20] K. Herrmann, H. Voigt, A. Behrend, J. Rausch, W. Lehner, Living in parallel realities: Co-existing schema versions with a bidirectional database evolution language, in: S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, D. Suciu (Eds.), Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, ACM, 2017, pp. 1101–

1116. URL: https://doi.org/10.1145/3035918.3064046. doi:10.1145/3035918.3064046.

[21] P. Koupil, J. Bártík, I. Holubová, *MM-evocat:* A tool for modelling and evolution management of multi-model data, in: M. A. Hasan, L. Xiong (Eds.), Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022, ACM, 2022, pp. 4892–4896. URL: https://doi.org/10.1145/3511808.3557180. doi:10.1145/3511808.3557180.

[22] L. Caruccio, G. Polese, G. Tortora, Synchronization of queries and views upon schema evolutions: A survey, ACM Trans. Database Syst. 41 (2016) 9:1–9:41. URL: https://doi.org/10.1145/2903726. doi:10.1145/2903726.

[23] A. Hillenbrand, M. Levchenko, U. Störl, S. Scherzinger, M. Klettke, Migcast: Putting a price tag on data model evolution in nosql data stores, in: P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, T. Kraska (Eds.), Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, ACM, 2019, pp. 1925–1928. URL: https://doi.org/10.1145/3299869.3320223. doi:10.1145/3299869.3320223.

[24] I. Holubová, M. Vavrek, S. Scherzinger, Evolution management in multi-model databases, Data Knowl. Eng. 136 (2021) 101932. URL: https://doi.org/10.1016/j.datak.2021.101932. doi:10.1016/j.datak.2021.101932.

[25] S. Chang, V. Deufemia, G. Polese, M. Vacca, A logic framework to support database refactoring, in: R. R. Wagner, N. Revell, G. Pernul (Eds.), Database and Expert Systems Applications, 18th International Conference, DEXA 2007, Regensburg, Germany, September 3-7, 2007, Proceedings, volume 4653 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 509–518. URL: https://doi.org/10.1007/978-3-540-74469-6_50. doi:10.1007/978-3-540-74469-6_50.

[26] P. Khumnin, T. Senivongse, SQL antipatterns detection and database refactoring process, in: T. Hochin, H. Hirata, H. Nomiya (Eds.), 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2017, Kanazawa, Japan, June 26-28, 2017, IEEE Computer Society, 2017, pp. 199–205. URL: https://doi.org/10.1109/SNPD.2017.8022723. doi:10.1109/SNPD.2017.8022723.

[27] D. Sjøberg, Quantifying schema evolution, Inf. Softw. Technol. 35 (1993) 35–44. URL: https://doi.org/10.1016/0950-5849(93)90027-Z. doi:10.1016/0950-5849(93)90027-Z.

[28] P. Vassiliadis, F. Shehaj, G. Kalampokis, A. V. Zarras, Joint source and schema evolution: Insights from a study of 195 FOSS projects, in: J. Stoyanovich, J. Teubner, N. Mamoulis, E. Pitoura, J. Mühlig, K. Hose, S. S. Bhowmick, M. Lissandrini (Eds.), Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023, OpenProceedings.org, 2023, pp. 27–39. URL: https://doi.org/10.48786/edbt.2023.03. doi:10.48786/edbt.2023.03.

[29] C. Curino, H. J. Moon, L. Tanca, C. Zaniolo, Schema evolution in wikipedia - toward a web information system benchmark, in: J. Cordeiro, J. Filipe (Eds.), ICEIS 2008 - Proceedings of the Tenth International Conference on Enterprise Information Systems, Volume DISI, Barcelona, Spain, June 12-16, 2008, 2008, pp. 323–332.

[30] M. Klettke, U. Störl, M. Shenavai, S. Scherzinger, Nosql schema evolution and big data migration at scale, in: J. Joshi, G. Karypis, L. Liu, X. Hu, R. Ak, Y. Xia, W. Xu, A. Sato, S. Rachuri, L. H. Ungar, P. S. Yu, R. Govindaraju, T. Suzumura (Eds.), 2016 IEEE International Conference on Big Data (IEEE BigData 2016), Washington DC, USA, December 5-8, 2016, IEEE Computer Society, 2016, pp. 2764–2774. URL: https://doi.org/10.1109/BigData.2016.7840924. doi:10.1109/BigData.2016.7840924.

[31] A. H. Chillón, D. S. Ruiz, J. G. Molina, Towards a taxonomy of schema changes for nosql databases: The orion language, in: A. K. Ghose, J. Horkoff, V. E. S. Souza, J. Parsons, J. Evermann (Eds.), Conceptual Modeling - 40th International Conference, ER 2021, Virtual Event, October 18-21, 2021, Proceedings, volume 13011 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 176–185. URL: https://doi.org/10.1007/978-3-030-89022-3_15. doi:10.1007/978-3-030-89022-3_15.

[32] U. Störl, M. Klettke, S. Scherzinger, Nosql schema evolution and data migration: State-of-the-art and opportunities, in: A. Bonifati, Y. Zhou, M. A. V. Salles, A. Böhm, D. Olteanu, G. H. L. Fletcher, A. Khan, B. Yang (Eds.), Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020, OpenProceedings.org, 2020, pp. 655–658. URL: https://doi.org/10.5441/002/edbt.2020.87. doi:10.5441/002/edbt.2020.87.

[33] B. A. Muse, F. Khomh, G. Antoniol, Refactoring practices in the context of data-intensive systems, Empir. Softw. Eng. 28 (2023) 46. URL: https://doi.org/10.1007/s10664-022-10271-x. doi:10.1007/s10664-022-10271-x.

[34] B. Karwin, S. Antipatterns, Avoiding the pitfalls of database programming, The Pragmatic Bookshelf (2010) 15–155.

[35] M. Athanassoulis, M. S. Kester, L. M. Maas, R. Stoica, S. Idreos, A. Ailamaki, M. Callaghan, Designing access methods: The RUM conjecture, in: E. Pitoura, S. Maabout, G. Koutrika, A. Marian, L. Tanca, I. Manolescu, K. Stefanidis (Eds.), Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016, OpenProceedings.org, 2016, pp. 461–466. URL: https://doi.org/10.5441/002/edbt.2016.42. doi:10.5441/002/edbt.2016.42.

[36] S. Salza, M. Terranova, Workload modeling for relational database systems, in: D. J. DeWitt, H. Boral (Eds.), Database Machines, Fourth International Workshop, Grand Bahama Island, March 1985, Springer, 1985, pp. 233–255.

[37] X. Huang, S. Cao, Y. Gao, X. Gao, G. Chen, Lightpro: Lightweight probabilistic workload prediction framework for database-as-a-service, in: C. A. Ardagna, N. L. Atukorala, B. Benatallah, A. Bouguettaya, F. Casati, C. K. Chang, R. N. Chang, E. Damiani, C. G. Guegan, R. Ward, F. Xhafa, X. Xu, J. Zhang (Eds.), IEEE International Conference on Web Services, ICWS 2022, Barcelona, Spain, July 10-16, 2022, IEEE, 2022, pp. 160–169. URL: https://doi.org/10.1109/ICWS55610.2022.00036. doi:10.1109/ICWS55610.2022.00036.

[38] L. Ma, D. V. Aken, A. Hefny, G. Mezerhane, A. Pavlo,

G. J. Gordon, Query-based workload forecasting for self-driving database management systems, in: G. Das, C. M. Jermaine, P. A. Bernstein (Eds.), Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, ACM, 2018, pp. 631–645. URL: https://doi.org/10.1145/3183713.3196908. doi:10.1145/3183713.3196908.

[39] S. S. Elnaffar, P. Martin, An intelligent framework for predicting shifts in the workloads of autonomic database management systems, in: Proc of 2004 IEEE International Conference on Advances in Intelligent Systems–Theory and Applications, 15-18, 2004, pp. 1–8.

[40] A. Chebotko, A. Kashlev, S. Lu, A big data modeling methodology for apache cassandra, in: B. Carminati, L. Khan (Eds.), 2015 IEEE International Congress on Big Data, New York City, NY, USA, June 27 - July 2, 2015, IEEE Computer Society, 2015, pp. 238–245. URL: https://doi.org/10.1109/BigDataCongress.2015.41. doi:10.1109/BigDataCongress.2015.41.

[41] M. J. Mior, K. Salem, A. Aboulnaga, R. Liu, Nose: Schema design for nosql applications, IEEE Trans. Knowl. Data Eng. 29 (2017) 2275–2289. URL: https://doi.org/10.1109/TKDE.2017.2722412. doi:10.1109/TKDE.2017.2722412.

[42] C. de Lima, R. dos Santos Mello, On proposing and evaluating a nosql document database logical approach, Int. J. Web Inf. Syst. 12 (2016) 398–417. URL: https://doi.org/10.1108/IJWIS-04-2016-0018. doi:10.1108/IJWIS-04-2016-0018.

[43] F. Abdelhédi, A. A. Brahim, F. Atigui, G. Zurfluh, Umltonosql: Automatic transformation of conceptual schema to nosql databases, in: 14th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2017, Hammamet, Tunisia, October 30 - Nov. 3, 2017, IEEE Computer Society, 2017, pp. 272–279. URL: https://doi.org/10.1109/AICCSA.2017.76. doi:10.1109/AICCSA.2017.76.

[44] A. Ben-Tal, Characterization of pareto and lexicographic optimal solutions, in: Multiple Criteria Decision Making Theory and Application: Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20–24, 1979, Springer, 1980, pp. 1–11.

[45] M. Hewasinghage, N. B. Seghouani, F. Bugiotti, Modeling strategies for storing data in distributed heterogeneous nosql databases, in: J. Trujillo, K. C. Davis, X. Du, Z. Li, T. W. Ling, G. Li, M. Lee (Eds.), Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings, volume 11157 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 488–496. URL: https://doi.org/10.1007/978-3-030-00847-5_35. doi:10.1007/978-3-030-00847-5_35.

[46] M. Hewasinghage, S. Nadal, A. Abelló, Docdesign 2.0: Automated database design for document stores with multi-criteria optimization, in: Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, F. Guerra (Eds.), Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021, OpenProceedings.org, 2021, pp. 674–677. URL: https://doi.org/10.5441/002/edbt.2021.81. doi:10.5441/002/edbt.2021.81.

[47] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2002) 182–197. URL: https://doi.org/10.1109/4235.996017. doi:10.1109/4235.996017.

[48] M. Hewasinghage, A. Abelló, J. Varga, E. Zimányi, Managing polyglot systems metadata with hypergraphs, Data Knowl. Eng. 134 (2021) 101896. URL: https://doi.org/10.1016/j.datak.2021.101896. doi:10.1016/j.datak.2021.101896.

[49] R. Rasool, A. A. Malik, Effort estimation of etl projects using forward stepwise regression, in: 2015 International Conference on Emerging Technologies (ICET), 2015, pp. 1–6. doi:10.1109/ICET.2015.7389209.

[50] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Metrics for the prediction of evolution impact in ETL ecosystems: A case study, J. Data Semant. 1 (2012) 75–97. URL: https://doi.org/10.1007/s13740-012-0006-9. doi:10.1007/s13740-012-0006-9.

[51] U. Störl, M. Klettke, Darwin: A data platform for schema evolution management and data migration, in: M. Ramanath, T. Palpanas (Eds.), Proceedings of the Workshops of the EDBT/ICDT 2022 Joint Conference, Edinburgh, UK, March 29, 2022, volume 3135 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3135/dataplat_short3.pdf.

[52] C. Forresi, E. Gallinucci, M. Golfarelli, H. B. Hamadou, A dataspace-based framework for OLAP analyses in a high-variety multistore, VLDB J. 30 (2021) 1017–1040. URL: https://doi.org/10.1007/s00778-021-00682-5. doi:10.1007/s00778-021-00682-5.

[53] V. V. Meduri, K. Chowdhury, M. Sarwat, Evaluation of machine learning algorithms in predicting the next SQL query from the future, ACM Trans. Database Syst. 46 (2021) 4:1–4:46. URL: https://doi.org/10.1145/3442338. doi:10.1145/3442338.

[54] E. Gallinucci, M. Golfarelli, S. Rizzi, Schema profiling of document-oriented databases, Inf. Syst. 75 (2018) 13–25. URL: https://doi.org/10.1016/j.is.2018.02.007. doi:10.1016/j.is.2018.02.007.

[55] C. Forresi, M. Francia, E. Gallinucci, M. Golfarelli, Streaming approach to schema profiling, in: A. Abelló, P. Vassiliadis, O. Romero, R. Wrembel, F. Bugiotti, J. Gamper, G. Vargas-Solar, E. Zumpano (Eds.), New Trends in Database and Information Systems - ADBIS 2023 Short Papers, Doctoral Consortium and Workshops: AIDMA, DOING, K-Gals, MADEISD, PeRS, Barcelona, Spain, September 4-7, 2023, Proceedings, volume 1850 of *Communications in Computer and Information Science*, Springer, 2023, pp. 211–220. URL: https://doi.org/10.1007/978-3-031-42941-5_19. doi:10.1007/978-3-031-42941-5_19.