Managing semantic evolution in databases: from theory to implementation

Pedro Ivo Siqueira Nepomuceno[1]
Advisor: Kelly Rosa Braghetto[1]

[1]Instituto de Matemática, Estatística e Ciência da Computação – Universidade de São Paulo
São Paulo – SP – Brazil

{pedro.siqueira, kellyrb}@ime.usp.br

Abstract. *Semantic heterogeneity in long-term datasets emerges as categories, groupings, and units change over time, requiring users to manually adapt queries and results. The master's thesis proposes a formal framework to address this issue through two strategies: query rewriting and data preprocessing. It introduces storage models and algorithms that manage semantic evolution via discrete, time-stamped operations (translation, merging, and splitting), enabling queries to be written as if the data were homogeneous. A prototype, MellowDB, was evaluated using Brazilian mortality data (1979–2021). Results show both approaches are production-ready, with data preprocessing generally outperforming query rewriting except in highly write-heavy scenarios.*

1. Introduction

Historical data analysis is essential for data-driven decision-making (g, a), yet long-lived datasets often suffer from semantic heterogeneity: the meaning, grouping, or naming of attribute values changes over time (e, i; n, e). Examples include municipality name changes () and shifts in disease classification standards (e.g., ICD-9 to ICD-10) (0, 2), which require users to manually reconcile inconsistencies when querying historical data. Even when the database schema remains unchanged, such semantic evolution can make records from different periods incompatible, leading to complex, error-prone queries and unreliable longitudinal analyses (n, e).

The thesis argues that semantic heterogeneity can be automatically addressed during query processing at a feasible computational cost. To this end, it formalizes the concept of semantic evolution and introduces a theoretical framework that defines Semantic Evolution Operations, including translation, merging, and splitting. Two complementary strategies are proposed: a data preprocessing approach that resolves semantic discrepancies at insertion time, and a query rewriting approach that handles them lazily at query time (l, o; e, l). Both strategies are supported by dedicated storage models and algorithms that preserve the complete semantic history of a dataset.

From a Computer Science perspective, this work contributes a formal model for value-level evolution distinct from traditional schema evolution, bridging theory and system design. It introduces novel storage abstractions, query processing algorithms, and a prototype system (MellowDB) that demonstrates the practical feasibility of managing semantic heterogeneity in production-like environments. Experimental results show that semantic reconciliation can be automated with low overhead, providing a scalable foun-

County	Year	Population
Moji Mirim	2014	91,027
Moji Mirim	2015	91,483
Mogi Mirim	2016	91,929
Mogi Mirim	2017	92,365
Mogi Mirim	2018	92,715

Table 1. Mogi Mirim population estimation by IBGE. Until 2015, the county name in the database was "Moji Mirim". Since 2016, however, the records present the new name for the county: "Mogi Mirim".

dation for consistent longitudinal analysis in evolving databases. Preliminary results obtained during this research were published at the International Conference on Database and Expert Systems Applications (DEXA) (p, e), and the final consolidated framework and evaluation were later published in the Future Generation Computer Systems journal (p, e).

2. Database, Schema and Semantic Evolution

Database evolution has long been a central topic in Computer Science. As early as the 1990s, researchers observed that a significant portion of development effort arises from post-deployment modifications (d, o). Over time, new application features, regulatory requirements, performance optimizations, and organizational changes demand adjustments in how data is structured and interpreted (a, r). Far from being a solved issue, database evolution remains an active research area, with recent work continuing to propose models and techniques for efficiently and safely managing evolving data systems (a, r; i, h; ö, t; e, l).

A substantial portion of the literature has focused on schema evolution, which concerns structural modifications to the logical organization of data. Common transformations include adding, deleting, or renaming attributes, as well as merging or splitting tables (r, u). In relational systems, such changes directly affect the schema definition, while in NoSQL systems they emerge through the implicit schema enforced by applications (e, l). Schema evolution raises important challenges, particularly regarding backward compatibility, data migration, and query correctness when multiple schema versions coexist (r, u,e).

Beyond structural changes, databases also experience semantic evolution, in which the meaning of attribute values changes over time without necessarily altering the schema itself (n, e). Examples include shifts in classification standards, category granularity, naming conventions, measurement units, or domain interpretations. Unlike schema evolution, which modifies structure, semantic evolution affects value interpretation. For example, as illustrated in Table 1, the municipality of Moji Mirim was renamed to Mogi Mirim, requiring queries over historical population data to explicitly account for the name change in order to retrieve consistent longitudinal results.

While semantic heterogeneity has been extensively studied in the context of integrating multiple databases, often using ontologies and taxonomy mappings (k, a; f, s; r, e), temporal semantic shifts within a single evolving database remain comparatively underexplored and pose distinct challenges for longitudinal analysis.

To address database evolution, the literature proposes several approaches. One major class consists of lazy techniques, which defer adaptation until query time (e, l). Among these, query rewriting dynamically transforms a query written for one schema version so that it can operate over another, avoiding physical changes to stored data (l, o). Other lazy approaches process records on demand when they are first accessed, sometimes caching the transformed result for subsequent use (e, l). These techniques prioritize flexibility and minimize upfront costs but may introduce runtime overhead.

In contrast, eager techniques handle evolution at the moment changes occur, typically through data migration, materialized transformations, or compatibility layers such as views and delta-based mechanisms (r, e). Some systems also support the coexistence of multiple schema versions, allowing different applications to interact with distinct representations simultaneously (r, e; l, o). Each strategy involves trade-offs between storage overhead, computational cost, backward compatibility, and system complexity. These established approaches to schema evolution provide conceptual foundations that can be adapted and extended to address semantic evolution, which is the central focus of this thesis.

3. Theoretical Framework

The theoretical foundation for managing semantic evolution in historical data draws selectively on conceptual elements introduced in PRIMA (o, o), particularly the notion of discrete evolution operations and the preservation of original data versions in raw form. However, while PRIMA focuses on transaction-time databases and structural schema evolution, the present framework extends these principles to address value-level semantic heterogeneity. It formalizes semantic evolution as a sequence of explicitly defined operations that capture how meanings, groupings, or interpretations change over time, without requiring retroactive modification of historical records. In this setting, users are allowed to formulate queries solely in terms of the current semantic version, while the system transparently handles all necessary rewriting and adaptation.

The framework preserves historical records in their original semantic forms and represents changes through explicit *Semantic Evolution Operations*, which describe when and how transformations occur. It provides generic formal definitions of records, collections, and queries, establishing the notation required for the storage models and algorithms developed later. In addition to a general formalization of semantic evolution operations, three concrete operations—translation, merging, and splitting—are defined to support implementation and experimental evaluation.

3.1. Initial Definitions

Definition 1 *Attribute and Domain*

Each attribute-value pair consists of an attribute name a and an associated value v. The notation $V[a]$ will be used to represent "the value of attribute a". All values of an attribute a must be contained in a set $D(a)$, the domain of attribute a.

Definition 2 *Record*

A record d contains a timestamp t representing when it became valid, and a set V of attribute-value pairs as defined in Definition 1. Thus, a record can be represented as:

$$d = (t, V) \qquad V = \{(a_1, v_1), (a_2, v_2), \ldots, (a_m, v_m)\}$$

Definition 3 *Data Collection*

A data collection C is a set of n records $C = \{d_1, d_2, \ldots, d_n\}$, where each record d_i is defined as in Definition 2.

According to these definitions, records may contain only simple values. That is, a value is neither another record nor another attribute-value pair. Although having nested complex objects instead of a simple value is a common design in many data models, such as semi-structured data and NoSQL databases, the semantic evolution operations that will be shown in this thesis only apply to simple values. Thus, including complex structure in record notation brings no immediate advantage. However, the operations' definition can be extended to include simple values inside nested structures.

3.2. Semantic Evolution Operations

A central assumption of the framework is that every instance of semantic heterogeneity within a data collection originates from a specific and identifiable event in time, referred to as a *Semantic Evolution Operation*. These operations do not necessarily result from deliberate changes introduced by data custodians; rather, they often emerge from external forces that alter semantic interpretation, such as official renamings (e.g., Moji Mirim to Mogi Mirim), revisions of classification standards (e.g., ICD-9 to ICD-10), taxonomy updates in scientific domains, or curriculum restructuring in education. Although externally driven, such events introduce semantic divergence within the data and must be explicitly modeled to ensure consistent reasoning across versions. The notion of a Semantic Evolution Operation therefore serves as the core abstraction for representing and formalizing these moments of semantic change within the framework.

Definition 4 *Semantic Evolution Operation*

A semantic evolution operation E is a function that changes the value $V[a]$ of an attribute a of a record $d = (t, V)$ to a different value r if two conditions are satisfied:

1. *$V[a]$ belongs to the set $Q \subseteq D(a)$ of values affected by the operation;*
2. *the time t when the record became valid is **previous** than time t_h when the evolution happened.*

When these two conditions are not satisfied, the function E keeps V unchanged. The function E can be formalized as:

$$E(t, V) = \begin{cases} (t, V \setminus \{(a, V[a])\} \cup \{(a, r)\}), & \text{if } t \leq t_h \text{ and } V[a] \in Q \\ (t, V), & \text{otherwise} \end{cases} \tag{1}$$

Intuitively, $V \setminus \{(a, q)\} \cup \{(a, r)\}$ can be read as "replace $V[a] = q$ by $V[a] = r$."

Due to space constraints, only the definition of the Merging operation is presented here. The complete thesis additionally defines two other Semantic Evolution Operations (Translation and Splitting) and their corresponding reverse operations which follow similar principles:

Definition 5 *Merging*

A Merging operation replaces a set of values with a single value r from a specific time t_h. Formally, it is a semantic evolution operation as defined in Definition 4 but with a non-unitary set Q and $R = \{r\}$.

To illustrate the necessity of the merging operation, consider a situation in which two previously distinct values become reported as a single aggregated value from a given year onward, so that the original values can no longer be distinguished in later records. A real-world example is the 1975 incorporation of the state of Guanabara into the state of Rio de Janeiro, after which population statistics may be reported only for the merged unit rather than for each state separately. In this case, a merging operation with $Q = \{\textit{Guanabara}, \textit{Rio de Janeiro}\}$, $r = \textit{Rio de Janeiro}$, and $t_h = 1975$ can be applied over the collection to model that, from 1975 onward, occurrences of either former value should be interpreted as the merged value. Table 2 presents the original published records used for illustration, prior to any post-processing under our framework.

State	Year	Population
Rio de Janeiro	1975	5,542,500
Guanabara	1975	4,857,700
Rio de Janeiro	1976	10,704,200

Table 2. Population estimates as published in 1975 and 1976.

3.3. Dealing with Semantic Evolution

We propose two distinct approaches for handling semantic evolution in query processing: query rewriting and data preprocessing. The query rewriting approach handles semantic evolution at query time by automatically transforming query clauses and adjusting results to ensure semantic consistency across versions. Rather than modifying stored data or requiring preprocessing, this lazy strategy delegates adaptation to the execution phase, shielding users from historical heterogeneity while preserving the raw collection unchanged (l, o; e, l).

In contrast, the data preprocessing approach addresses semantic evolution at insertion time. Instead of rewriting queries dynamically, it eagerly processes records as they are inserted, incorporating semantic transformations in advance and building a Semantic Evolution Aware Collection optimized for consistent querying (e, l).

4. Storage Model

Following the formalization of semantic evolution, we define a storage model that represents semantic transformations while preserving computational and storage feasibility. The model ensures controlled storage overhead and efficient query execution, allows the full reconstruction of semantic transformation history, and provides access to the original (raw) version of every record. Based on this model, algorithms for insertion and querying can be defined, with requirements of practical performance and abstraction: users should be able to query data without being exposed to semantic evolution details. The effectiveness of both the query rewriting and data preprocessing approaches depends on this storage design: it is particularly central to the preprocessing strategy, where new semantic versions are materialized upon evolution, while in the rewriting strategy, it mainly supports metadata management without altering stored records.

The proposed storage model comprises three main components. The *Raw Collection* stores records in their original form with minimal alterations. The *Semantic Versions*

Collection maintains metadata describing semantic versions and the evolution operations that connect them. The *Processed Collection*, used by the data preprocessing approach, stores eagerly transformed records aligned with semantic versions. By allowing records to span intervals of semantic validity instead of duplicating them for every version, this structure reduces storage overhead while preserving full semantic traceability.

5. The prototype: MellowDB

To validate the proposed storage model and algorithms, we developed MellowDB, a middleware library written in Python and released under the MIT License, designed to manage semantic evolution in document-oriented NoSQL databases. Although currently implemented on top of MongoDB (n, o), its data model and algorithms are DBMS-agnostic and can be adapted to other document stores. The API was intentionally designed to resemble PyMongo, lowering the learning curve and enabling seamless integration into existing MongoDB-based systems. The full source code is publicly available at `https://github.com/pisn/semantic_heterogeneous_database`.

MellowDB implements both query rewriting and data preprocessing strategies described in the theoretical framework, delegating semantic transformation logic to the DBMS through MongoDB aggregation pipelines, which helps control memory usage in large collections.

6. Performance Evaluation

The practical viability of the proposed strategies was evaluated through systematic experimentation. Implementing both query rewriting and data preprocessing enabled a direct performance comparison. The rewrite strategy automates what knowledgeable users typically perform manually (rewriting queries and adapting results to account for semantic changes) thereby reducing cognitive burden and minimizing the risk of inconsistencies. In contrast, the data preprocessing approach shifts semantic handling to insertion time, potentially introducing non-negligible overhead that may or may not be compensated by improved query performance. These lead to the central experimental questions:

1. Are the proposed approaches practically feasible for real-world use?
2. Assuming both approaches are viable, under which conditions does one outperform the other? What factors, such as data volume, evolution frequency, or query complexity, should guide the decision of which approach to adopt?

The experiments were conducted using a real-world dataset containing 30 years of mortality records published by , with one record per cause of death, municipality, and year. The dataset exhibits semantic heterogeneity due to a classification change in 1996, when Brazil transitioned from a national standard based on ICD-9 to one based on ICD-10. Although derived from the ICD, the Brazilian classification does not map one-to-one to ICD codes, as each national code may correspond to a single ICD entry or to grouped codes (0, 2). Additionally, the dataset includes municipality name changes and territorial splits over time, introducing a second type of semantic evolution. Altogether, 170 semantic evolution operations were modeled in the experiments, making this dataset a comprehensive and realistic test case for evaluating the proposed approaches.

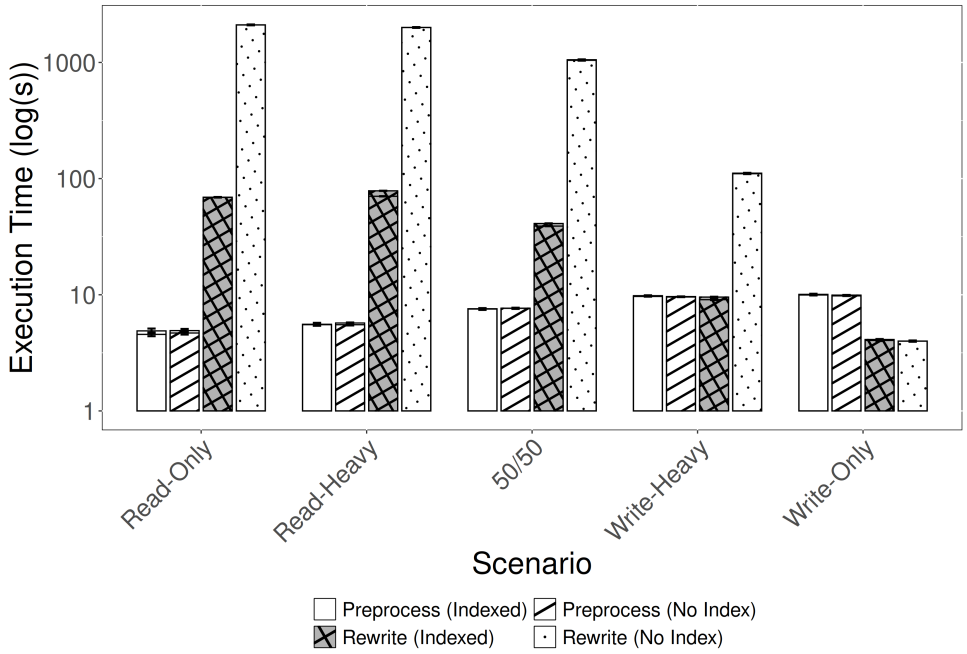

Figure 1. Scenarios performance comparison with and without indexes. The execution time is on a logarithmic scale. 500 operations in each experiment.

To assess how each strategy behaves across different operational profiles, we divided the tests into workload classes based on the proportions of read and write operations. Specifically, we considered *read-only* workloads (100% reads), *write-only* workloads (100% writes), and three mixed workloads: a *write-heavy* scenario with 95% writes and 5% reads, a *read-heavy* scenario with 95% reads and 5% writes, and a balanced workload with a 50/50 split between reads and writes. Figure 1 presents a logarithmic-scale comparison of execution times for the query rewriting and data preprocessing approaches (with and without indexes), each tested with 500 operations. The logarithmic scale effectively highlights the magnitude of performance differences between the two approaches.

The experimental results reveal a clear trade-off between the two strategies. In the data preprocessing approach, execution time deteriorates in write-intensive scenarios because semantic transformations are applied during insertion, whereas the query rewriting approach introduces no overhead at insertion time. Conversely, in read-intensive scenarios the preprocessing strategy performs significantly better, since records are already semantically aligned, while the rewriting strategy must execute semantic adjustments during query processing. Numerically, preprocessing adds an average overhead of 0.01 seconds per insertion, whereas rewriting adds approximately 4.32 seconds per query. The proportion of heterogeneous operations (15% versus 30%) showed minimal impact on performance in both approaches, indicating that overhead is driven more by the chosen strategy than by the frequency of semantic evolution events.

Indexing substantially alters this balance. While rewriting only outperforms preprocessing in strictly write-only scenarios, preprocessing already becomes advantageous with a small percentage of read operations. The introduction of indexes dramatically reduces the query overhead of the rewriting approach—from approximately 4.21 seconds

Table 3. Average execution times (mean ± standard deviation) of operations in the data preprocessing and query rewriting approaches, both with and without indexes.

Operation	Indexing?	Execution Time (s)	
		Data Preprocessing	Query Rewriting
Insertion	Not Indexed	0.0196 ± 0.0001	0.0080 ± 0.0001
	Indexed	0.0199 ± 0.0002	0.0081 ± 0.0001
Query	Not Indexed	0.0098 ± 0.0004	4.2122 ± 0.0375
	Indexed	0.0098 ± 0.0006	0.1380 ± 0.0009

Table 4. Collection statistics in MongoDB.

Collection	Size	Storage Size	Number of Documents
Raw Collection	1.15 GB	208 MB	6,179,083
Versions Collection	85 KB	56 KB	171
Processed Collection	3.28 GB	478 MB	11,089,626

to 0.13 seconds—significantly narrowing the performance gap. Even so, in environments dominated by insertions (e.g., around 98% writes), rewriting may remain preferable. Overall, the results presented in Table 3 confirm that both strategies are computationally feasible for production use, with the optimal choice depending primarily on workload characteristics.

In addition to performance overhead, storage efficiency was evaluated using MongoDB's `stats()` function (0, 2), analyzing both logical size and physical *storageSize* for each collection (Table 4). As expected, the Processed Collection occupies more space than the Raw Collection due to materialized semantic versions; however, thanks to the interval-based design introduced in the storage model, its size grows far less than a naive implementation would suggest (i.e., not proportional to the total number of semantic evolutions). Nevertheless, even in the absence of evolutions, the data preprocessing approach requires at least roughly twice the storage of the query rewriting strategy, and this gap increases as more evolutions are introduced and records are split across semantic intervals. Therefore, while preprocessing significantly reduces time overhead compared to rewriting, it does so at the cost of increased storage consumption.

7. Conclusion

Semantic heterogeneity is a subtle yet persistent challenge in data systems: as meanings evolve due to domain changes, standards, or regulations, queries may still run while producing incomplete or misleading results. This semantic drift is hard to detect and, when handled manually, requires time-consuming and error-prone query and result adjustments. Despite its practical relevance, the literature has largely emphasized schema evolution, leaving temporal semantic evolution within a single database comparatively underexplored.

This thesis addresses that gap by defining Semantic Evolution Operations, such as translation, merging, and splitting, and proposing two complementary handling strate-

gies: query rewriting and data preprocessing, supported by a storage model that preserves raw data and semantic version history. These contributions extend database evolution beyond structural adaptation, offering a value-level evolution model that bridges formalization, algorithms, and system design, and advancing Computer Science by introducing computationally feasible mechanisms for meaning-aware data management.

The approaches were implemented in the MellowDB prototype and validated with real-world longitudinal data. Data preprocessing generally outperforms query rewriting in read-intensive scenarios with minimal overhead (often around 0.01 seconds per operation), while rewriting may be preferable in highly write-dominant workloads; preprocessing also incurs higher storage costs, mitigated by the interval-based model. Overall, the results show that semantic heterogeneity can be managed automatically with controlled time and space overhead, reinforcing semantics as a first-class concern in evolving data infrastructures.

References

Asfand-E-Yar, M. and Ali, R. (2020). Semantic integration of heterogeneous databases of same domain using ontology. *IEEE Access*, 8:77903–77919.

Brahmia, Z., Grandi, F., and Oliboni, B. (2024). A literature review on schema evolution in databases. *Computing Open*, 2:2430001:1–54.

Brazilian Health Ministry. Tabnet public health information. `https://datasus.saude.gov.br/informacoes-de-saude-tabnet/`. Accessed: 2025-01-23.

Chillón, A. H., Klettke, M., Ruiz, D. S., and Molina, J. G. (2024). A generic schema evolution approach for NoSQL and relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):2774–2789.

Curino, C., Moon, H. J., Deutsch, A., and Zaniolo, C. (2013). Automating the database schema evolution process. *The VLDB Journal*, 22(1):73–98.

Golfarelli, M., Lechtenbörger, J., Rizzi, S., and Vossen, G. (2006). Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. *Data & Knowledge Engineering*, 59(2):435–459.

Hakimpour, F. and Geppert, A. (2005). Resolution of semantic heterogeneity in database schema integration using formal ontologies. *Information Technology and Management*, 6:97–122.

Herrmann, K., Voigt, H., Behrend, A., Rausch, J., and Lehner, W. (2017). Living in parallel realities: Co-existing schema versions with a bidirectional database evolution language. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD/PODS, pages 1101–1116.

Instituto Brasileiro de Geografia e Estatística - IBGE. Alterações topomínicas - 2022.

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. (2014). Big data and its technical challenges. *Commun. ACM*, 57(7):86–94.

Klettke, M., Storl, U., Shenavai, M., and Scherzinger, S. (2016). NoSQL schema evolution and big data migration at scale. In *2016 IEEE International Conference on Big Data*, Big Data, pages 2764–2774.

Li, X., Madnick, S. E., and Zhu, H. (2013). A context-based approach to reconciling data interpretation conflicts in web services composition. *ACM Transactions on Internet Technology (TOIT)*, 13(1):1–27.

Mergen, S. L. S. and Heuser, C. A. (2006). Data translation between taxonomies. In *International Conference on Advanced Information Systems Engineering*, pages 111–124. Springer.

Moller, M. L., Klettke, M., Hillenbrand, A., and Störl, U. (2019). Query rewriting for continuously evolving NoSQL databases. In *International Conference on Conceptual Modeling*, ER, pages 213–221.

MongoDB, I. (2025). Mongodb.

MongoDB, Inc. (2025). db.collection.stats(). `https://www.mongodb.com/pt-br/docs/manual/reference/method/db.collection.stats/`. Accessed: 2025-04-27.

Moon, H. J., Curino, C. A., Deutsch, A., Hou, C.-Y., and Zaniolo, C. (2008). Managing and querying transaction-time databases under schema evolution. *Proceedings of the VLDB Endowment*, 1(1):882–895.

Nepomuceno, P. I. S. and Braghetto, K. R. (2023). Managing semantic evolutions in semi-structured data. In *International Conference on Database and Expert Systems Applications*, pages 179–185. Springer.

Nepomuceno, P. I. S. and Braghetto, K. R. (2026). Managing semantic evolution in databases: From theory to implementation. *Future Generation Computer Systems*, 177:108257.

Roddick, J. F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393.

Secretaria de Estado de Saúde de Minas Gerais (2014). Mortalidade cid-10 – lista de tabulação cid-br. `http://tabnet.saude.mg.gov.br/Notas_tecnicas/Mortalidade_CID-10_Lista_CID-BR.pdf`. Accessed: July 24, 2025.

Störl, U., Klettke, M., and Scherzinger, S. (2020). NoSQL schema evolution and data migration: State-of-the-art and opportunities. In *EDBT*, volume 20, pages 655–658.

Ventrone, V. (1991). Semantic heterogeneity as a result of domain evolution. *ACM SIGMOD Record*, 20(4):16–20.

World Health Organization (2019). *International Classification of Diseases, 11th Revision (ICD-11)*. World Health Organization, Geneva.