

# Combining Time-Series and Graph Data: A Survey of Existing Systems and Approaches \*

Mouna Ammar  
 University of Leipzig,  
 ScaDS.AI  
 Leipzig, Germany  
 ammar@informatik.uni-leipzig.de

Marvin Hofer  
 University of Leipzig,  
 ScaDS.AI  
 Leipzig, Germany  
 hofer@informatik.uni-leipzig.de

Erhard Rahm  
 University of Leipzig,  
 ScaDS.AI  
 Leipzig, Germany  
 rahm@uni-leipzig.de

## ABSTRACT

We provide a comprehensive overview of current approaches and systems for combining graphs and time series data. We categorize existing systems into four architectural categories and analyze how these systems meet different requirements and exhibit distinct implementation characteristics to support both data types in a unified manner. Our overview aims to help readers understand and evaluate current options and trade-offs, such as the degree of cross-model integration, maturity, and openness.

## 1. INTRODUCTION

Graph and time-series data are needed in many areas and are typically managed in separate data management systems. Graph data is widely used, e.g., in social networks, micromobility, and finance [69, 87, 90, 70], to model complex relationships and apply analysis ranging from pattern matching and traversal to graph machine learning. Graph data is either managed in native systems such as Neo4j or JanusGraph, which are based on a property graph model, or in more general data stores such as key-value stores or relational database management systems (DBMS) [11]. Time series are another essential data structure for recording measurements or events over time (e.g., stock prices, sensor readings) and for supporting time-related analytics such as prediction and anomaly detection [72, 78]. Again, this kind of data can be managed either in native stores (e.g., time-series DBMS such as TimescaleDB [85] and InfluxDB [29]) or in more general data stores.

Applications increasingly demand the combined analysis of both graph data and time series [79, 46, 60], notably in machine learning. For example, the connections (e.g., road segments) of a traffic network can be extended with time-series data to cap-

ture vehicle counts or average speeds, enabling identification of congestion and its influence on neighboring areas. Similarly, the analysis of sensor time series in an Internet of Things (IoT) environment, e.g., for anomaly detection, can be improved by accounting for sensor topology and potential dependencies among sensors.

Storing graph data and time-series data in separate data stores is a minimal approach for such applications, as they would have to manage different systems and query languages. Furthermore, they would have to implement all queries and analysis tasks that require both data types. Hence, we see a strong need for closer integration of graph data and time series within a single system that provides not only specific support for both data types but also new (hybrid) operations on them. A vision for such a powerful hybrid approach has been presented in [2], but a complete implementation remains lacking.

In this work, we therefore focus on surveying existing approaches for combining graph and time-series data. We categorize solutions into several architectural integration types and evaluate the different systems against a list of requirements and system characteristics. We consider four integration architectures: (i) Single model (**SM**) systems that use a single data model (either a native graph or time series model or another data model) to manage both graph and time series data, (ii) Extended Single Model (**E-SM**) systems that extend a main model with dedicated extensions for graph/time series support , (iii) **Polyglot** systems integrating multiple engines via middleware, and (iv) Multi Model (**MM**) systems that do not have a single preferred data model but support multiple equally important models in one engine. Our survey aims to help readers understand the current options and their strengths and limitations. The shown methodology

\*This paper is submitted to SIGMOD Record.

can also be applied to systems not covered in this paper. We release the codebase used to assess all four architectures for executing cross-model queries in [1].

We make the following contributions:

1. We propose a set of requirements as well as relevant system characteristics for the combined management and analysis of graph and time series data
2. We describe four architectural approaches to combining graphs and time series, along with their advantages and disadvantages.
3. We comparatively evaluate 20 systems of the different categories w.r.t. the introduced requirements and characteristics.

The remainder of the paper is organized as follows. Section 2 presents related surveys, Section 3 outlines the requirements and system characteristics for the comparative evaluation Section 4 describes the integration architectures. Section 5 surveys representative systems and research prototypes. We synthesize the open challenges in Section 6 and conclude our work in Section 7.

## 2. RELATED SURVEYS

Multiple surveys have been conducted on solely time-series databases [35, 34, 9, 38, 31, 88, 95] and on graph databases [11, 17, 3, 30, 94, 44]. Besta et al. [12] classify more than 50 systems by data models, storage layouts, and query execution strategies, highlighting fundamental design tensions in handling dynamic, large-scale graphs. None of these surveys addresses approaches to supporting both time-series and graph data, as we do in this work. There are several models and implementations for *temporal graph database systems* extending property graphs ([28, 21, 27, 57, 58, 92, 93]) or the RDF graph data model [98, 97]. These approaches can manage and analyze the evolution of graph data over time. However, they do not cover time series as we aim for. Lu and Holubová [47] provide a general overview of multi-model databases and a comparison of different systems, but without considering support for time series data and without taking specific requirements into account as we do here.

## 3. CRITERIA FOR COMPARISON

We define a set of key requirements and desirable system characteristics for a data management system that supports both graph and time-series

data. This is essentially a wish list for a perfect solution, which current solutions will only partially cover. Still, we believe they provide a reasonable basis for comparing available systems and identifying currently missing functionality.

### 3.1 System Requirements

#### Requirements Group 1: Data model and Streaming

- D1 Individual model coverage: There should be data model support for graph data, e.g., for property graphs with labeled data nodes and edges, as well as for time series data.
- D2 Unified hybrid data model: Ideally, there should be a unified data model that supports a seamless integration of both graph and time series data as first-class citizens, as well as their combined representation (e.g., time series properties of graph nodes or interlinked time series).
- D3 Temporal evolution: Support is needed for both current and historical data to capture how graphs and time series evolve over time.
- D4 Streaming data support: The system should support ingestion, processing, and querying of unbounded high-velocity data streams, in particular continuously arriving time series and graph updates, with low-latency processing, proper handling of out-of-order or late events, and integration with stored state [80, 89]

#### Requirements Group 2: Queries and transformations

- Q1 Cross-model query engine: It should be possible to execute queries on both graph and time-series data, and to mix them in a single query. Users must be able to join, filter, aggregate, and pattern-match across the two models without manually orchestrating multiple query engines. In addition to hybrid queries (e.g., time-aware grouping), support should also include extraction operators (e.g., extracting time series from graph data) and hybrid updates that span both data models with consistent semantics. Ideally, this capability is exposed through a unified query abstraction or query language [26, 41]
- Q2 Bidirectional model transformation (interoperability and flexibility): Support is needed to transform data between the single data models (from graph to time series and back) as well as between integrated hybrid model and single-model (graph-only or time-series-only)

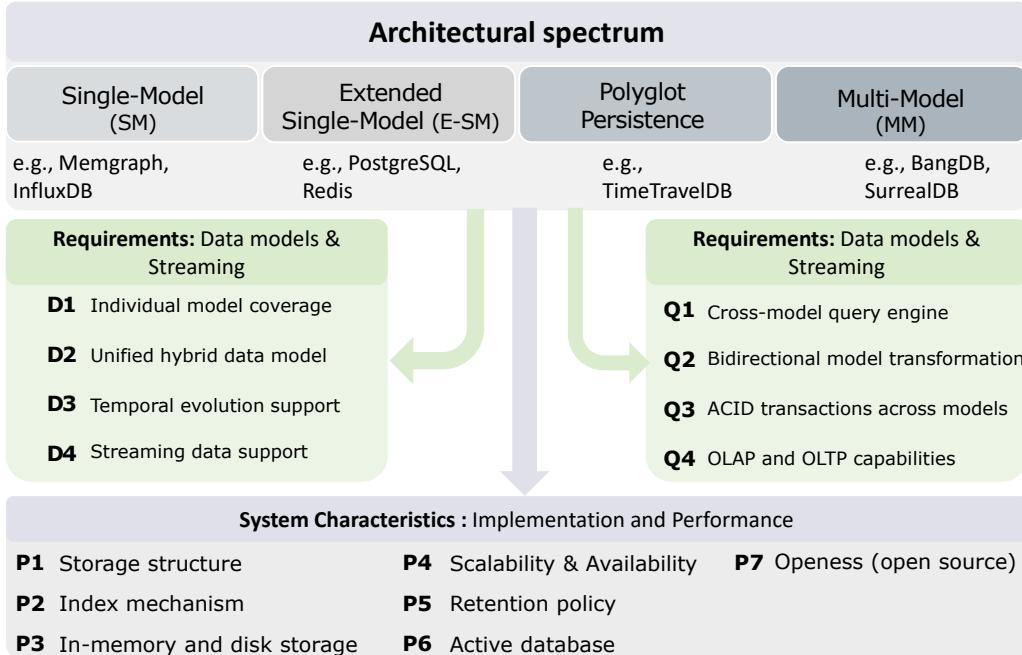


Figure 1: Overview of the comparison approach of systems for combining graph and time series data

formats, as well as for import and export of single-model and hybrid data.

- Q3 ACID transactions across data types: Atomic, consistent updates that span graph and time-series entities in the same transaction.
- Q4 OLTP and OLAP capabilities: Support for both transactional OLTP workloads (point queries, updates, pattern matching, snapshot queries) and analytical OLAP workloads (aggregations, global graph and time series algorithms) without requiring separate ETL [8] or systems. Such hybrid HTAP capabilities [45] are beneficial because hybrid graph and time series applications inherently combine transactional operations (e.g., graph traversal) with analytical operators (e.g., time series aggregations).

### 3.2 System Characteristics

The posed data model and query requirements are challenging to meet and require advanced implementation techniques to achieve fast processing and scalability with large datasets. Given that multiple system options partially meet the posed requirements, we do not specify implementation requirements. Instead, we identify relevant system characteristics related to implementation and performance that influence how well the posed requirements can be met. These aspects will also be considered in our comparative evaluation of current implementations.

#### Implementation and Performance Characteristics

- P1 Storage structure of graph and TS data, e.g., adjacency matrix, B<sup>+</sup>-tree, log-structured merge (LSM) tree, columnar storage.
- P2 Graph-TS Index mechanisms: Indexing support for graph and time series data as well as for composite indexing, e.g., to jointly capture graph entities with their time-series attributes.
- P3 In-memory and disk storage: Support for in-memory storage and processing (e.g., for fast analysis) and external storage for persistence and durability.
- P4 Scalability and availability mechanisms: Support horizontal scaling through data distribution (e.g., sharding) to handle growing volumes of graph and time-series data while maintaining query performance and maintaining fault tolerance through replication.
- P5 Retention policy. Built-in lifecycle rules (e.g., time-to-live, downsampling) of historical data control how long graph and time-series data are retained.
- P6 Active database support to react to changes, e.g., ECA rules (Event-Condition-Action), triggers, or standing/continuous queries [49].

P7 Openness. Commercial vs. open-source license. Systems released under an OSI-approved license [63] (e.g., Apache 2.0, GPL, PostgreSQL license, MIT, BSD) are considered open-source systems. Other systems under licenses such as SSPL/BSL, with or without available source, are considered closed-source.

Figure 1 summarizes the criteria and characteristics introduced in our comparison approach.

## 4. INTEGRATION ARCHITECTURES

We categorize existing systems for combining graph and time series data into four architectural approaches: *Single-model (SM)*, *Extended Single-model (E-SM)*, *Polyglot persistence system*, and *Multi-model database (MMDB)*<sup>1</sup>. The gross architectures of these solutions are shown in Figure 2. Table 1 summarizes some of the benefits and challenges of the four architectures and lists sample implementations. This section discusses the architectures in more general terms, while the next section will comparatively evaluate selected systems across the four integration architectures. In our repository [1], we provide solutions for several DBMS architectures for storing and analyzing graph and time-series data in a mobility use case (bike sharing).

### 4.1 Single-model systems (SM)

This approach uses a single-model DBMS to store data of other types. Three variations of this approach can be distinguished:

- *Graph database systems that store time series data.* This method natively supports graph data processing, e.g., for a property graph model (e.g., Neo4j [62], JanusGraph [33], TigerGraph [84], Memgraph [55]) or a temporal graph approach ( [56, 53] but also stores and manage time series data, e.g., within node or edge properties.
- *Time-series DBMS storing graph data.* This is the symmetric approach where a native time series DBMS such as InfluxDB [29] aims at also storing graph data, e.g., by treating nodes and edges as time-stamped events.
- *Other database systems storing graph and time series.* In this approach, a DBMS with no native support for graph or time series data, e.g., a relational or NoSQL DBMS, is used to store and process these kinds of data. In a relational DBMS (e.g., MySQL [61]), one can store this data in different tables and use SQL to analyze

graph and time-series data, which typically have substantial limitations in performance and analysis. Object-relational DBMSs address some of these limitations through dedicated extensions for non-relational data, thereby enabling an E-SM architecture. In document stores such as MongoDB [59]), each time series point can be modeled as a document carrying a timestamp. Such documents can then be *linked* (via references/-foreign keys) or *embedded* to form a graph-shaped structure, as detailed further in our solution available in this repository [1]. Key-value stores (e.g., RockDB [76]) can encode graph adjacency lists as values keyed by node identifiers and time series as sorted sets or lists.

Systems of type SM are typically most limited w.r.t. our posed requirements since they have dedicated support for at most one of the two data types, thereby lacking advanced query support for the other data type as well as for cross-model queries. Even temporal graph DBMS with query languages such as T-GQL [21] and T-PGQL [77] do not support time-series analysis. Similarly, non-graph DBMS have neither dedicated support for complex graph queries (e.g., k-hop pattern matching), nor for cross-model queries.

### 4.2 Extended single-model (E-SM)

Systems within the E-SM architecture are based on a mature core data model, such as the relational model, but extend it (e.g., via plugins or modules) to support additional data types, including JSON, graph, and time-series data. This leads to multi-model capabilities, but the different models are not equally important as in the architectural type MM. Furthermore, the additional data types are often mapped to the original model’s storage structures; i.e., data are ultimately stored according to the core model’s paradigm, which can impose restrictions on possible queries and performance.

The E-SM approach is explicitly supported by the object-relational extensions of the relational model, which enable user-defined data types and procedures, e.g., to support non-relational data. The SQL standard has already defined such extensions for document data (XML, JSON) and graph data [22] but not yet for time-series data (although SQL window predicates are useful for time-series analysis). The operators of the extended data types can be used in SQL queries, thereby supporting cross-model queries and data type transformations. Existing DBMS, such as PostgreSQL [71] and Oracle, provide extensions that support both graph and time-series data, thereby extending the SQL stan-

<sup>1</sup>These architectures are similarly applicable when other kinds of data should be combined.

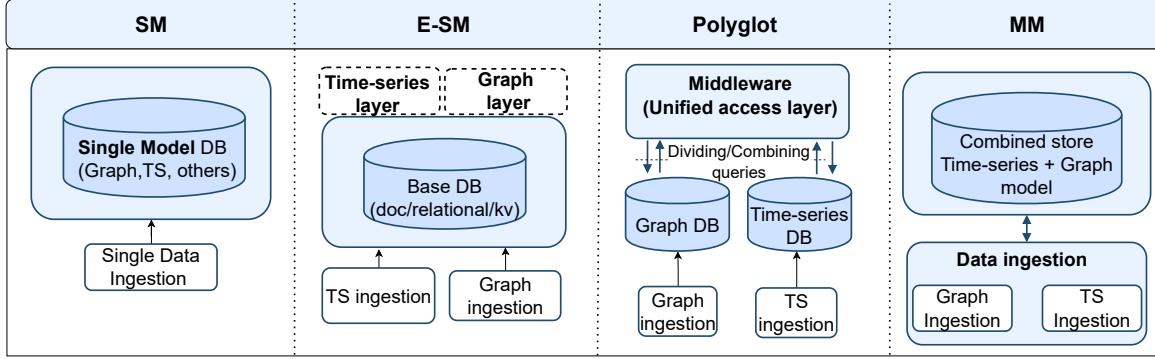


Figure 2: Sketches of integration architectures: SM, E-SM, Polyglot, and MM

dard.

The key-value store Redis [73] provides extensions for both graph and time-series data. For the wide-column data store Cassandra [15, 16], there are several options for storing both data types. Cassandra can either be used as a storage backend with both Janusgraph (for graph data) [32] and KairosDB [37] (for time series data) built on top of it, or use DataTax Graph [20], a native extension of the commercial Cassandra distribution to store graph data and time series data directly in Cassandra, which already provides optimized storage and some analysis capabilities. An example query is provided in [1]. The discussion shows that there are substantially different implementations within the E-SM category, even among relational DBMS. The E-SM approach can support critical requirements such as cross-model queries, but expressiveness and performance depend on the chosen mappings and implementations based on the core data model.

### 4.3 Polyglot persistence system

Polyglot persistence [23, 91] is an architectural concept that integrates multiple data types and technologies to support applications with diverse data processing needs by leveraging the strengths of different DBMS to optimize specific data model, query, or scalability requirements [24]. The core motivation behind polyglot persistence is that no single database can fit well all data types and workloads [39]. For our purposes, this approach could be used to integrate a graph DBMS (e.g., Neo4J) with a time-series DBMS (e.g., InfluxDB).

The downside is that all cross-system features must be provided by an additional middleware layer on top of the native systems (Figure 2 (Polyglot)). For query processing, cross-model queries are either to be implemented by the applications or the middleware could support specific cross-model query

features or even a hybrid query language based on suitable cross-model schemata. In that case, the middleware would have to split cross-model queries into single-model subqueries and to combine the results for the subqueries. Obviously, this is very challenging, making it difficult to support complex cross-model queries with acceptable performance [39, 40, 42].

We are not aware of a commercially available polyglot system that combines graph and time-series DBMS. We therefore included a research prototype developed at Leipzig University, TimeTravelDB, which is designed to store both graph and time-series data natively and consists of a new query language, Time-Travel Query Language (TTQL), to manage models and their combinations.

An example query for such a polyglot approach is presented in [1].

### 4.4 Multi-model database (MM)

Multi-model DBMS (MMDB) [96] provides a unified platform for managing data across multiple models and data types, such as documents, graphs, relational tables, and time series, within a single system. While polyglot persistence relies on separate, specialized data stores for different data models, an MMDB integrates these models, offering a more flexible and consistent approach to data management with fewer data synchronization issues [47]. In contrast to E-SM systems, all supported data models are treated equally as ‘first-class citizens’, ensuring consistent performance and capabilities across all models [26, 47]. MMDB systems use a single engine to natively handle all supported data models without relying on additional layers, modules, plug-ins, or middleware [5, 26]. While there can be support for multiple query languages for different models, there should ideally be a unified query language that seamlessly manages and queries all

Table 1: Overview of System Architectures for Graph + Time Series

Category	Key Benefits	Key Challenges	Example systems
<b>Single-Model</b>	leveraging mature DBMS. ACID across models since one engine.	optimized for at most one of the models (graph or time series), no support for hybrid queries	Neo4j [62], InfluxDB [29], Memgraph [55], MongoDB [59], Aion [83], ClockG [53]
<b>Extended Single-Model</b>	like for SM plus dedicated support for graph/time series data	extensions must align with the core model; potential performance overhead	PostgreSQL [71], Redis [73], Couchbase [18], MariaDB [50], Cassandra [15], Oracle [66]
<b>Polyglot Persistence</b>	native support of both data models.	high integration complexity, high latency for queries on both kinds of data, no cross-model transactions, no mature implementation	TimeTravelDB prototype [86]
<b>MMDB</b>	single engine supports multiple models, ACID across models	complex to implement; restricted maturity of current implementations; graph/time series processing less optimized than in native DBMS	ArangoDB [6], OrientDB [68], ArcadeDB [7], BangDB [10], SurrealDB [82]

supported data models. This would support complex multi-model queries without the need to switch between different query languages [47]. For example, ArcadeDB supports Cypher and Gremlin for graph queries in addition to SQL-like functionality. By contrast, ArangoDB provides a complex unified query language called AQL [54].

While MMDB systems appear promising, they are challenging to implement with robust functionality and performance, depending on the extent to which the system can integrate data from different models. This is also complicated by the lack of established standards and best practices for integrating and optimizing various data models [47, 41, 48]. We will see that the current MMDBs remain quite restricted in meeting our requirements. The close integration of graph and time series data with a uniform query language, as envisioned in [2], could be supported by an advanced MMDB but has not yet been implemented.

## 5. COMPARATIVE EVALUATION

In our evaluation, we compare 15 commercial data management systems and five research prototypes of the four integration architectures w.r.t. the introduced requirements (Table 2) and implementation characteristics (Table 3)<sup>2</sup>. While it is not possible to consider all relevant systems, we identified

<sup>2</sup>For space reasons, we moved some requirements, such as support for OLTP and OLAP, to Table 3

representative solutions for each of the four categories that at least partially meet the requirements for processing both graph and time-series data. In the following, we discuss the results separately for the four architectures.

In Table 2, we classify systems with native stream processing as fully supporting *streaming* (e.g., with real-time ingestion and analytical pipelines), systems that rely on external stream-processing engines via connectors as partially supporting streaming, and systems without explicit streaming capabilities beyond batch operations as not supporting streaming. Regarding *bidirectional model transformation*, we classify a system supporting both directions (from graph/ts → hybrid and back) as full support, and for support of only one direction (e.g., importing hybrid data but not exporting it or exporting only a single model) as partial support. In Table 3, full support of *Graph-TS indexing* requires indexing for both data types and composite indexing. In contrast, partial support is indicated when only one of the data types is indexed or composite Graph-TS indexing is missing. Several industrial systems provide multiple editions (e.g., community and enterprise). In our comparison, a system is considered open source if it offers an OSI-licensed community edition and if the capabilities reported in our tables are available in that community edition without requiring a paid license. If the reported capabilities rely on enterprise-only features, the sys-

tem is not considered open source in our evaluation. For example, Neo4j provides both a community and an enterprise edition. While most graph query operations are supported in the community edition, scalability is only available in the enterprise edition. Accordingly, we consider Neo4j to be open source but not scalable. In contrast, the Oracle database is distributed under a commercial license. Although limited free editions exist, they are not open source. Therefore, Oracle is treated as a closed-source system, and the reported capabilities correspond to its enterprise offering.

## 5.1 Single model systems result

For systems of type **SM**, we include native property graph DBMS (Neo4j, Memgraph), native time series DBMS (InfluxDB), and a document store (MongoDB). Furthermore, we consider three research prototypes for temporal graph data (Aion, AeonG, Clock-G) and one prototype for property graphs with time-series support (Bollen et al. [13]). As expected and indicated in Table 2, no SM system fully supports both property graphs (PG) and time series (TS). Cross-model queries and ACID transactions are largely unsupported, thereby leaving essential requirements unsatisfied.

The considered *graph systems* Neo4J and Memgraph have no built-in support for time series. One could model time series by property values. Still, with simple property values, this would require a large number of  $2n$  property values for a time series of length  $n$  (one property value per event, for the timestamp and the value). In Neo4j, the entire time series can also be represented as a graph with a chain of event nodes or using the GraphAware Timetree plugin [25]. This encoding enables hybrid queries (pattern matching constrained by temporal predicates or aggregations), but at the expense of verbose, complex query formulations and, consequently, poor usability. The research prototype by Bollen et al. includes support for time series and cross-model queries in its query language GQL-TS and represents time series as a linked list of nodes [13]. Neo4J, Memgraph, and prototype [13] focus on OLTP, but OLAP-like queries are also possible. The in-memory system Memgraph has support for streaming (graph) data.

The *temporal graph prototypes* keep historical graph data and can thus provide support of evolution and snapshot analysis, but do not natively support time-series analysis or hybrid queries. The query language is usually based on Neo4J’s Cypher, extended with temporal predicates, e.g., to derive snapshots at a specific point in time. Both OLTP and OLAP

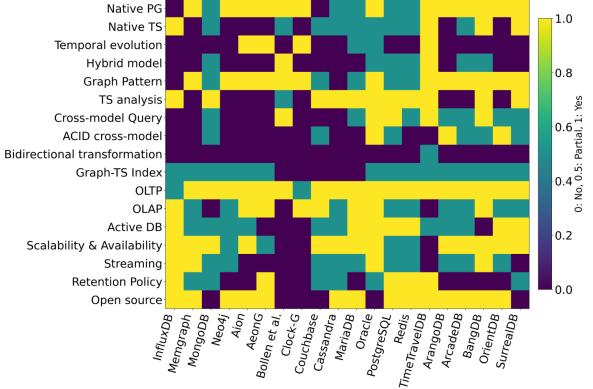


Figure 3: Coverage of requirements and characteristics across systems

queries are typically supported. The prototype AeonG<sup>3</sup> is under BSL1.1 licence, which means it’s closed source, while Aion<sup>4</sup> is open-source. No code implementation is available for Clock-G and Bollen et al.’s method.

The *time-series DBMS* InfluxDB has no built-in support for storing and analyzing graph data or cross-model queries. Time-series events (measurements) can be enriched by metadata that can be used to store node states or relationships between nodes. Hence, there are ways for users to represent and analyze time-evolving graph data, but not so much for structural graphs and complex graph traversals and analyses. InfluxDB is an OLAP system for time-series data and supports streaming data.

The other relational and NoSQL SM systems provide no native support for graph and time-series data. Still, it is possible to represent and analyze both types of data using the underlying data model and query language. In the considered *document DBMS*, MongoDB graph nodes and edges can be represented by documents of typed collections, and there is an operator `$graphLookup` for simple graph analysis. TS data can be stored in time-series collections, introduced in MongoDB 5.0, and analyzed with aggregation operators. There is support for compressing time-series data and for a TTL (time-to-live) policy for data retention. Hence, one can achieve some basic support for both data types and mixed queries, but without advanced or optimized capabilities.

The implementation characteristics (Table 3) are quite heterogeneous, with a focus on disk storage and partial support for sharding and replication.

<sup>3</sup><https://github.com/hououou/AeonG>

<sup>4</sup><https://github.com/aeon-toolkit/aeon>

Table 2: Modeling and Query Capabilities across Systems. Legend: ✓ = supported; ○ = workaround or via extension/plugin; ✗ = not supported. RP = research prototype. KV = key value. Symbols: ✓ = native/first-class; ○ = via extension or workaround; ✗ = unsupported.

Architecture	System	Primary Model	Native PG	Native TS	Temporal evolution	Hybrid model	Streaming	Query Language	Graph Pattern	TS analysis	Cross-model Query	Bidirectional transformation	Acid cross-model
SM	InfluxDB	Time Series	✗	✓	✗	✗	✓	InfluxQL	✗	Aggregations, grouping, join, etc	✗	✗	✗
	Memgraph	Property Graph	✓	✗	✗	✗	✓	openCypher	✓	✗	✗	✗	✗
	MongoDB	Document	○	○	✗	○	○	MQL	○	\$bucket, \$window, etc. in agg. pipeline	○	✗	○
	Neo4j	Property Graph	✓	✗	✗	✗	○	Cypher	✓	✗	✗	✗	✗
SM (RP)	Aion	Temporal Graph	✓	✗	✓	✗	✗	Temporal Cypher	✓	✗	✗	✗	✗
	AeonG	Temporal Graph	✓	✗	✓	✗	✗	Temporal Cypher	✓	✗	✗	✗	✗
	Bollen et al.	Property graph with time series	✓	○	✗	✓	✗	GQL-TS	✓	Windows, aggregations and TS pattern	✓	✗	✗
	Clock-G	Temporal Graph	✓	✗	✓	✗	✗	T-Cypher	✓	✗	✗	✗	✗
E-SM	Couchbase	Document	✗	○	✗	✗	○	SQL++	○	_TIMESERIES()	✗	✗	○
	Cassandra*	Wide-column store	○	○	✗	✗	○	CQL	✗	Aggregation functions \$merge, \$out, \$group, etc.	✗	✗	✗
	MariaDB	Relational	○	○	○	✗	○	SQL	○	Aggregations, statistical functions over window	○	✗	✗
	Oracle	Relational	✓	○	○	○	✓	SQL, PGQL	✓	ML, TS regression, forecasting, statistics	✓	✗	✓
	PostgreSQL*	Relational	○	○	✗	○	○	SQL	○	Timescale: GROUP BY, time filters, aggregations	✓	✗	○
	Redis*	Key-value	○	○	✗	○	○	Commands / Cypher (graph)	○	RedisTS: TS.GET, TS.RANGE, etc.	○	✗	✗
	Polyglot TimeTravelDB (RP)	TTPGM (1)	✓	✓	✓	✓	✗	TTQL (Extended Cypher)	(Ex- ✓	Native (FROM, TO, SHALLOW), aggregations	✓	○	✗
MM	ArangoDB	Document + Graph + KV	✓	✗	✗	✗	○	AQL	✓	✗	○	✗	✓
	ArcadeDB	Document + Graph + Time Series + KV	✓	○	✗	○	○	SQL-like	✓	✗	○	✗	○
	BangDB	Document + Graph + Time Series	✓	✓	✗	○	✓	C++, SQL-like	✓	Join, filter, aggregations, sliding window, etc.	✓	✗	○
	OrientDB	Document + Graph + KV	✓	✗	✗	✗	○	SQL-like	✓	✗	○	✗	✓
	SurrealDB	Document + Graph + Time Series (+others)	✓	✓	✗	✗	✗	SurrealQL	✓	Aggregations, GROUP BY, etc.	○	✗	○

**Notes:** Asterisks (\*) indicate plugin-based capabilities. RedisGraph EOL on Jan 31, 2025. (1) TimeTravel Property Graph Model

Table 3: Implementation characteristics. Legend: ✓ = native/first-class. ○ = partial or via extension/plugin. ✗ = not supported. CQ = Continuous query. CQN = continuous query notification. Scalability & Availability codes: S = Sharding; R = Replication; SR = Sharding + Replication. KV=key-value

Architecture	System	Storage Structure	Graph-TS Index	Disk/In-Memory	OLTP	OLAP	Active DB	Scalability & Availability	Retention Policy	License
SM	InfluxDB	Time-Structured Merge Tree (TSM)	○	Disk	○	✓	✓ CQ	R (1)	✓	MIT (open source)
	Memgraph	In-memory (skip lists)	○	In-memory	✓	○	○ Trigger	R	○	BSL 1.1 (source-available)
	MongoDB	B-tree (WiredTiger)	○	Disk	✓	✗	○ Trigger	SR	○	SSPL (source-available)
	Neo4j	Fixed-size record store	○	Disk	✓	○	○ Trigger	✗ (1)	✗	GPL v3 (Community)
SM (RP)	Aion	TimeStore + LineageStore	○	Both	✓	✓	○ Event listener	SR	✗	Open source
	AeonG	Current store (MVCC) + historical KV (anchor+delta)	○	Disk	✓	✓	✗	S (4)	✓	RP
	Bollen et al.	Fixed-size record store (Neo4j)	✗	Disk	✓	✗	✗	✗	✗	RP
	Clock-G	δ-Copy+Log (Cassandra backend)	✗	Disk	○	✓	✗	✗	✗	RP
E-SM	Couchbase	B-tree (Couchstore/-Magma)	✗	Both	✓	✓	○ Eventing service	SR	○	BSL 1.1 (source-available)
	Cassandra*	Log-structured merge-tree (SSTables)	✗	Disk	✓	○	○ Java trigger	SR	○	Apache 2.0
	MariaDB	Row store (InnoDB) (2)	✗	Both	✓	✓	✓ Trigger	SR	✗	GPL v2
	Oracle	row store table (5)	○	Both	✓	✓	✓ Trigger/CQN	SR	○	Commercial
	PostgreSQL*	Row store (B-tree / HEAP)	○	Disk	✓	○	✓ Trigger	S (3)	✓	PostgreSQL License
Polyglot (RP)	Redis*	In-memory (hash + skip list)	○	In-memory	✓	○	✓ Trigger	S	✓	BSD 3-Clause
	TimeTravelDB	Record store + columnar	○	Disk	✓	✗	○ Trigger + CQ	S	✓	Open source
MM	ArangoDB	LSM-tree (RocksDB)	○	Disk	✓	○	○ Spring event	SR	✗	BSL 1.1
	ArcadeDB	LSM-tree	○	Disk	✓	○	○ Java listener	R	✗	Apache 2.0
	BangDB	B-tree + Hash	○	Both	✓	✓	✗	SR	✗	BSD 3-Clause
	OrientDB	Paginated Storage	Local	Both	✓	○	✓ Trigger	SR	✗	Apache 2.0 (Community)
	SurrealDB	LSM tree (RocksDB)	○	Both	✓	○	✓ Live queries	SR	○	BSL 1.1 (source-available)

Notes: (1) Entreprise = HR. (2) ColumnStore engine available for time series. (3) H with Citus. (4) Distributed variant AeonG-D. (5) In-memory column store available

## 5.2 Extended single-model (E-SM) results

For type E-SM, we consider three relational DBMS (Oracle, PostgreSQL, MariaDB) and three NoSQL systems (Redis, Couchbase, Cassandra). These systems primarily support graph and time-series data via extensions (requirement D1), partially support cross-model queries (Q1), and partially support ACID across data types (Q3), but still lack a unified hybrid model (D2) to integrate both data types within a single data structure. There is support for streaming data (D4), but not much for temporal evolution (D3). In E-SM systems, scalability guarantees provided by the core engine do not necessarily extend to graph and time series extensions, and vice versa. Therefore, in this survey, we consider a system to fully support scalability only if both the core engine and its extension are scalable.

The relational DBMS Oracle provides strong support for property graphs and for analytic analysis of time-series data stored in relational tables. It can also manage the temporal evolution of both graph and time-series data with support for historical querying (D3). The graph capabilities of Oracle Database 23ai include an extensive library of graph algorithms (more than 80 [64]). Time-series analysis is supported through SQL functions [65] (e.g., `TIME_BUCKET`, `AGO`, `TODATE`) and Oracle Machine Learning for SQL for time-series forecasting [67]. Oracle can thus serve both OLTP and OLAP workloads. Streaming support is possible via Oracle Cloud Infrastructure and a Kafka-compatible event streaming service [36] which can be used for real-time ingestion of graph and time-series data into Oracle.

PostgreSQL is a relational database that provides extensions for time-series data via TimescaleDB [85] and for graph data via Apache AGE [4]. This also includes support for cross-model queries (Q1) and *partially* ACID across models (Q3) provided that Timescale and AGE tables are managed by the same Postgres instance. However, extension boundaries and heterogeneous storage paths often limit the depth of hybrid support. PostgreSQL supports only partial data streaming (D4) via its TimescaleDB extension, which enables real-time data ingestion and querying.

The relational DBMS MariaDB supports multiple storage engines on the same server, including engine *OQGRAPH* to handle complex data in a graph-like structure [51], as well as bi-temporal tables for validity and transaction time [14]. There is no dedicated storage adaptation for time-series data, but such data can be handled by the *ColumnStore* engine, which is optimized for analytic queries [52].

Support for cross-model queries is low. They require an ETL-like step to communicate between engines for query processing: exporting results from one engine and importing them into another for a combined query execution.

Redis is an open-source, in-memory key-value store that includes extensions for time-series data (RedisTimeSeries [75]) and graph data (RedisGraph [74], which uses the OpenCypher query language), although RedisGraph is no longer supported. Each extension provides native query capabilities for its respective data model, but there is no built-in support for cross-model queries. Redis provides partial streaming support via the Redis Streams extension [81]. Data retention support (P5) is provided only through RedisTimeSeries (P5).

The distributed document store Couchbase has no native support for graph and time-series data. Still, it can store these data types in JSON documents and provides optimizations and dedicated analysis capabilities for time series data [19]. Its SQL-like query language, SQL++, also supports recursive queries, e.g., for reachability queries and graph traversals. There is no built-in support for cross-model queries so users have to implement the combination of graph documents with time-series documents. Streaming is supported only via connectors to an external streaming system.

The primary reason for including Cassandra is its high scalability as a wide-column store, due to its decentralized storage architecture [43], and its use as backend storage for time-series databases, such as KairosDB [37], and for graph databases, such as JanusGraph. Both can be used within a single Cassandra engine.

## 5.3 Polyglot persistence

For polyglot persistence, we consider the research prototype TimeTravelDB (TTDB) [2] due to a lack of a suitable commercial system. It combines the graph DBMS Neo4j and the time-series extension TimescaleDB of PostgreSQL. In its middleware layer it supports an extended temporal property graph data model called TTPGM (Time Travel Property Graph Model) (D1) where graph nodes can possess time-series properties. A UUID (universally unique identifier) references time-series data within graph elements. This is the “hybrid link” that maintains consistency between the two separated databases (D2).

For query processing an extended Cypher query language called TTQL (TimeTravel query language) supports temporal range queries and aggregations on time-series data [86] (Q1). Query processing en-

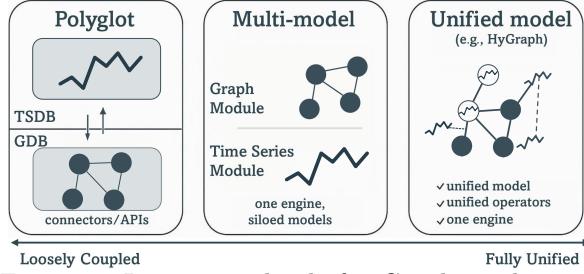


Figure 4: Integration levels for Graphs and Time series

tails splitting a user query into a Cypher component (graph) and an SQL component (time-series), executing both components, and merging the results. TimeTravelDB derives its OLTP capabilities from its underlying systems but remains weak in OLAP, providing only pattern matching with time-series constraints. The TTDB prototype supports temporality through a validity property per graph element and temporal range queries (keywords FROM, TO) (D3).

#### 5.4 Multi-model systems (MM)

We consider five multi-model implementations (*ArangoDB*, *OrientDB*, *ArcadeDB*, *BangDB*, *SurrealDB*). They all provide native support for documents and graphs, but not all support time series, as these can be stored in other data structures, such as ordered lists and key-value maps. Native TS support is provided by *BangDB* and *SurrealDB* and is planned for *ArcadeDB*, whereas *ArangoDB/OrientDB* rely on generic document collections for managing time series. A unified hybrid data model (D2) and temporal evolution (D3) are generally not provided, but there is support for cross-model queries (Q1) and cross-model ACID transactions (Q3). *ArangoDB*, *OrientDB*, and *ArcadeDB* provide graph analytics (e.g., PageRank, SSSP) and aggregation capabilities for large graphs. An example cross-model query with *ArcadeDB* is provided in [1].

### 6. OPEN CHALLENGES

The heatmap in Figure 3 summarizes the extent to which the surveyed systems align with our modeling, querying, and implementation criteria. Overall, no system satisfies all requirements. The best overall coverage is achieved by the E-SM system Oracle (approximately 73% of the defined criteria), followed by the multi-model system BangDB (about 67%) and PostgreSQL with its extensions (62%). The polyglot research prototype TimeTravelDB also has strong support for the criteria (about 70%), but, of course, lacks the maturity of the former systems.

Oracle offers commercial solutions; BangDB provides a community edition; however, PostgreSQL is available free of charge.

Despite the variety of existing systems with support of graph and time series data, there are still **significant gaps** in the posed requirements and implementation features so that a true unification of the two data models has not yet been achieved. Except for the support of time series properties in graphs in two research prototypes there is no support of a **unified hybrid data model** (requirement D2) that can combine graph and time-series data in one composite data model. Therefore, a truly hybrid query language is still missing, with full support for **bidirectional data model transformations** (Q2), import/export of hybrid data, and implementation features such as composite graph/time-series indexing (P2). For example, consider a query that must return graph paths subject to specific time-series constraints, such as temporary spike patterns on either entities or links. With existing solutions, one must first manually link time series to their entities, then retrieve all possible paths via graph querying, and finally check, for each path, whether the condition holds in the time-series data. Such a solution would require excessive time and computation, whereas in a unified environment, one can simultaneously check time-series conditions and filter out invalid paths directly.

As illustrated in Figure 4, the different integration architectures and their implementing systems vary in the extent to which they unify graph and time-series data. At one end, polyglot persistence systems loosely integrate systems of two (or more) data models but need a complex middleware to combine the data. Multi-model systems and E-SM systems achieve a closer integration within a single DBMS and its extensions (e.g., a graph database management system and a time-series database management system) via external connectors or APIs, but still treat each model separately with no unified data model and limited cross-model expressiveness. At the far end lies the vision of a truly unified data model, in which both graph and time-series data are treated as first-class citizens within a unified representation with shared semantics, indexing, and querying capabilities. The HyGraph effort [2] aims at providing such a solution.

### 7. CONCLUSIONS

We surveyed 20 systems across four architectural alternatives for combining graph and time-series data and comparatively evaluated them against a set of key requirements and implementation characteris-

tics. The proposed methodology can be applied to further systems that could not be included here. The quantitative evaluation can help readers to understand the current state of art and to find the most promising systems for a particular use case or research direction.

While there are no complete solutions that cover all requirements, we found that some commercial DBMS already provide reasonable support, e.g., for evaluating both types of data in a single query. This holds especially for E-SM systems such as Oracle and PostgreSQL, as well as for multi-model approaches such as BangDB. What is currently lacking is support for a unified hybrid data model that enables closer integration of both data types, with the backing for hybrid operators and data transformations. In future work, we plan to design a benchmark for hybrid graph and time-series processing that enables quantitative evaluation of selected systems.

## 8. REFERENCES

- [1] M. Ammar. Repository to reproduce cross-model query examples for each architecture, 2025. Created: August 2025.
- [2] M. Ammar, C. Rost, R. Tommasini, S. Agarwal, A. Bonifati, P. Selmer, E. Kharlamov, and E. Rahm. Towards hybrid graphs: Unifying property graphs and time series. In *28th International Conference on Extending Database Technology*, 2025.
- [3] S. Anuyah, V. Bolade, and O. Agbaakin. Understanding graph databases: a comprehensive tutorial and survey. *arXiv preprint arXiv:2411.09999*, 2024.
- [4] apache age. Apache age™ graph database for postgresql, 2024.
- [5] ArangoDB. Arangodb official documentation, 2023.
- [6] arangodb. Arangodb. multimodel database, 2025.
- [7] arcadedb. The next generation multi-model database supporting graphs, key/value, documents, search engine, vectors and time-series, 2021.
- [8] M. Attobrah. Etl pipeline. In *Essential Data Analytics, Data Science, and AI: A Practical Guide for a Data-Driven World*, pages 27–46. Springer, 2024.
- [9] A. Bader, O. Kopp, and M. Falkenthal. Survey and comparison of open source time series databases. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband*, pages 249–268. Gesellschaft für Informatik eV, 2017.
- [10] bangdb. Bangdb, the multimodal database for modern apps, 2025.
- [11] M. Besta, R. Gerstenberger, E. Peter, M. Fischer, M. Podstawska, C. Barthels, G. Alonso, and T. Hoefer. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *ACM Computing Surveys*, 56(2):1–40, 2023.
- [12] M. Besta, R. Gerstenberger, E. Peter, M. Fischer, M. Podstawska, C. Barthels, G. Alonso, and T. Hoefer. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *ACM Comput. Surv.*, 56(2):31:1–31:40, 2024.
- [13] E. Bollen, R. Hendrix, and B. Kuijpers. Managing data of sensor-equipped transportation networks using graph databases. *Geoscientific Instrumentation, Methods and Data Systems*, 13(2):353–371, 2024.
- [14] A. Bui. Bitemporal modeling: implementations, performance and use cases. *LU-CS-EX*, 2022.
- [15] cassandra. Cassandra. open source nosql database, 2025.
- [16] A. Cassandra. Planet cassandra, 2025.
- [17] M. E. Coimbra, L. Svitáková, A. P. Francisco, and L. Veiga. Survey: Graph databases. *arXiv preprint arXiv:2505.24758*, 2025.
- [18] couchbase. Couchbase. distributed nosql developer database platform, 2025.
- [19] Couchbase. Introducing couchbase time series, 2025.
- [20] Datastax. Datastax graph as an extension of cassandra (commercial edition), 2025.
- [21] A. Debrouvier, E. Parodi, M. Perazzo, V. Soliani, and A. Vaisman. A model and query language for temporal graph databases. *The VLDB Journal*, 30(5):825–858, 2021.
- [22] A. Deutsch, N. Francis, A. Green, K. Hare, B. Li, L. Libkin, T. Lindaaker, V. Marsault, W. Martens, J. Michels, et al. Graph pattern matching in gql and sql/pgq. In *Proc. ACM SIGMOD*, pages 2246–2258, 2022.
- [23] M. Fowler and P. Sadalage. Nosql databases polyglot persistence. *Personal Website: http://martinfowler.com/articles/nosql-intro-original.pdf*, 2012.
- [24] D. Glake, F. Kiehn, M. Schmidt, F. Panse, and N. Ritter. Towards polyglot data stores—overview and open research questions. *ArXiv preprint*, abs/2204.05779, 2022.

- [25] GraphAware. Graphaware neo4j timetree, 2024.
- [26] Q. Guo, C. Zhang, S. Zhang, and J. Lu. Multi-model query languages: taming the variety of big data. *Distributed and Parallel Databases*, 42(1):31–71, 2024.
- [27] T. Hartmann, F. Fouquet, M. Jimenez, R. Rouvoy, and Y. Le Traon. Analyzing complex data in motion at scale with temporal graphs. In *The 29th International Conference on Software Engineering & Knowledge Engineering (SEKE'17)*, page 6. KSI Research, 2017.
- [28] H. Huang, J. Song, X. Lin, S. Ma, and J. Huai. Tgraph: A temporal graph data management system. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pages 2469–2472, 2016.
- [29] influxdb. Influxdata: Influxdb time series data platform, 2024.
- [30] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. Prat-Pérez, T. Manhardt, H. Chafi, M. Capota, N. Sundaram, M. J. Anderson, I. G. Tanase, Y. Xia, L. Nai, and P. A. Boncz. LDBC graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *Proc. VLDB Endow.*, 9(13):1317–1328, 2016.
- [31] M. Jalal, S. Wehbi, S. Chilingaryan, and A. Kopmann. Scits: A benchmark for time-series databases in scientific experiments and industrial internet of things.(2022), 12: 1–12: 11, 2022.
- [32] JanusGraph. Apache cassandra as a storage backend for janusgraph, 2025.
- [33] janusgraph. Janusgraph. distributed, open source, massively scalable graph database, 2025.
- [34] S. K. Jensen, T. B. Pedersen, and C. Thomsen. Time series management systems: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2581–2600, 2017.
- [35] S. K. Jensen, T. B. Pedersen, and C. Thomsen. Time series management systems: a 2022 survey. In *Data Series Management and Analytics*. Association for Computing Machinery, 2022.
- [36] Kafka. Kafka: an open-source distributed event streaming platform., 2025.
- [37] kairosDB. Kairosdb.fast time series database on cassandra, 2025.
- [38] A. Khelifati, M. Khayati, A. Dignös, D. Difallah, and P. Cudré-Mauroux. Tsm-bench: benchmarking time series database systems for monitoring applications. *Proceedings of the VLDB Endowment*, 16(11):3363–3376, 2023.
- [39] P. P. Khine and Z. Wang. A review of polyglot persistence in the big data world. *Information*, 10(4):141, 2019.
- [40] F. Kiehn, M. Schmidt, D. Glake, F. Panse, W. Wingerath, B. Wollmer, M. Poppinga, and N. Ritter. Polyglot data management: State of the art & open challenges. *Proc. VLDB Endow.*, 15(12):3750–3753, 2022.
- [41] P. Koupil, D. Crha, and I. Holubová. Mm-quecat: A tool for unified querying of multi-model data. In *EDBT*, pages 831–834, 2023.
- [42] O. Lajam and S. Mohammed. Revisiting polyglot persistence: From principles to practice. *International Journal of Advanced Computer Science and Applications*, 13(5), 2022.
- [43] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS operating systems review*, 44(2):35–40, 2010.
- [44] ldbc. Ldbc financial benchmark, 2024.
- [45] G. Li and C. Zhang. HTAP databases: What is new and what is next. In Z. G. Ives, A. Bonifati, and A. E. Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 2483–2488. ACM, 2022.
- [46] X. Liu, Y. Xia, Y. Liang, J. Hu, Y. Wang, L. Bai, C. Huang, Z. Liu, B. Hooi, and R. Zimmermann. Largest: A benchmark dataset for large-scale traffic forecasting. *Advances in Neural Information Processing Systems*, 36:75354–75371, 2023.
- [47] J. Lu and I. Holubová. Multi-model databases: a new journey to handle the variety of data. *ACM Computing Surveys (CSUR)*, 52(3):1–38, 2019.
- [48] J. Lu, Z. H. Liu, P. Xu, and C. Zhang. Udbms: road to unification for multi-model data management. In *Advances in Conceptual Modeling: ER 2018 Workshops Emp-ER, MoBiD, MREBA, QMMQ, SCME, Xi'an, China, October 22–25, 2018, Proceedings 37*, pages 285–294. Springer, 2018.
- [49] A. Margara, J. Urbani, F. Van Harmelen, and H. Bal. Streaming the web: Reasoning over dynamic data. *Journal of Web Semantics*,

- 25:24–44, 2014.
- [50] mariadb. Mariadb. end-to-end database solution, 2025.
- [51] mariadb\_graph. Oqgraph storage engine in mariadb server, 2023.
- [52] mariadb\_ts. Analysis of financial time series data using mariadb columnstore, 2023.
- [53] M. Massri, Z. Miklos, P. Raipin, and P. Meye. Clock-g: A temporal graph management system with space-efficient storage technique. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2263–2276. IEEE, 2022.
- [54] K. Mavrogiorgos, A. Kiourtis, A. Mavrogiorgou, and D. Kyriazis. A comparative study of mongodb, arangodb and couchdb for big data storage. In *Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing*, pages 8–14, 2021.
- [55] Memgraph. A graph database, 2025.
- [56] Y. Miao, W. Han, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, E. Chen, and W. Chen. Immortalgraph: A system for storage and analysis of temporal graphs. *ACM Transactions on Storage (TOS)*, 11(3):1–34, 2015.
- [57] O. Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.
- [58] V. Z. Moffitt and J. Stoyanovich. Temporal graph algebra. In T. Rompf and A. Alexandrov, editors, *Proceedings of The 16th International Symposium on Database Programming Languages, DBPL 2017, Munich, Germany, September 1, 2017*, pages 10:1–10:12. ACM, 2017.
- [59] mongodb. Mongodb: A document database, 2024.
- [60] S. M. Mousavi, Y. Sheng, W. Zhu, and G. C. Beroza. Stanford earthquake dataset (stead): A global data set of seismic signals for ai. *IEEE Access*, 7:179464–179476, 2019.
- [61] MySQL. Mysql: A relational database, 2025.
- [62] neo4j. Neo4j: A graph database, 2024.
- [63] Opensource.org. Osi approved licenses, 2025.
- [64] Oracle. Integrated graph database features., 2025.
- [65] Oracle. Model time series data., 2025.
- [66] Oracle. Oracle: A relational database management system (rdbms)., 2025.
- [67] Oracle. Time series machine learning., 2025.
- [68] orientdb. Orientdb document & graph database, 2025.
- [69] A. Patel and J. Dharwa. Graph data: the next frontier in big data modeling for various domains. *Indian Journal of Science and Technology*, 10(21):1–7, 2017.
- [70] J. Pokorný. Graph databases: their power and limitations. In *IFIP International Conference on Computer Information Systems and Industrial Management*, pages 58–69. Springer, 2015.
- [71] PostgreSQL. Postgresql: The official site for postgresql, the world’s most advanced open source database, 2024.
- [72] S. Rani et al. Review on time series databases and recent research trends in time series mining. In *2014 5th International Conference-Confluence The Next Generation Information Technology Summit (Confluence)*, pages 109–115. IEEE, 2014.
- [73] redis. Redis. end-to-end database solution, 2025.
- [74] RedisGraph. Redisgraph is a queryable graph database built on redis..
- [75] RedisTimeSeries. Ingest and query time series data with redis.
- [76] RockDB. Rockdb: A persistent key-value store for fast storage environments, 2025.
- [77] C. Rost, K. Gomez, P. Christen, and E. Rahm. Evolution of degree metrics in large temporal graphs. In *Datenbanksysteme für Business, Technologie und Web (BTW 2023)*, volume P-331 of *LNI*, pages 485–507. Gesellschaft für Informatik e.V., 2023.
- [78] S. Schmidl, P. Wenig, and T. Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797, 2022.
- [79] Z. Shao, Z. Zhang, F. Wang, and Y. Xu. Pre-training enhanced spatial-temporal graph neural network for multivariate time series forecasting. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 1567–1577, 2022.
- [80] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.
- [81] R. Streams. Introduction to redis streams.
- [82] surrealdb. Surrealdb. the ultimate multi-model database for ai agents and applications, 2025.
- [83] G. Theodorakis, J. Clarkson, and J. Webber. Aion: Efficient temporal graph data management. In *EDBT*, pages 501–514, 2024.
- [84] TigerGraph. A graph database, 2025.

- [85] Timescaldb. Timescaldb: An open-source time-series sql database, 2024.
- [86] timetraveldb. Timetraveldb. a polyglot persistence combining neo4j and timescaledb, 2025.
- [87] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual ACM Southeast Conference*, pages 1–6, 2010.
- [88] L. K. Visperas and Y. Chodpathumwan. Time-series database benchmarking framework for power measurement data. In *2021 Research, Invention, and Innovation Congress: Innovation Electricals and Electronics (RI2C)*, pages 25–30. IEEE, 2021.
- [89] Y. Watanabe, S. Yamada, H. Kitagawa, and T. Amagasa. Integrating a stream processing engine and databases for persistent streaming data management. In *International Conference on Database and Expert Systems Applications*, pages 414–423. Springer, 2007.
- [90] N. white paper. “overcoming sql strain and sql pain”, April 2015.
- [91] L. Wiese. Polyglot database architectures= polyglot challenges. In *LWA*, pages 422–426, 2015.
- [92] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, 2014.
- [93] Y. Xu, J. Huang, A. Liu, Z. Li, H. Yin, and L. Zhao. Time-constrained graph pattern matching in a large temporal graph. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pages 100–115. Springer, 2017.
- [94] L. Yu. An empirical evaluation of temporal graph benchmark. *CoRR*, abs/2307.12510, 2023.
- [95] H. Yuanzhe, Q. Xiongpai, C. Yueguo, L. Yaru, S. Xiaoguang, T. Yu, Z. Xiao, and D. Xiaoyong. Ts-benchmark: A benchmark for time series databases. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021.
- [96] C. Zhang, J. Lu, P. Xu, and Y. Chen. Unibench: A benchmark for multi-model database management systems. In *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018, Rio de Janeiro, Brazil, August 27–31, 2018, Revised Selected Papers 10*, pages 7–23. Springer, 2019.
- [97] F. Zhang, Z. Li, D. Peng, and J. Cheng. RDF for temporal data management - a survey. *Earth Sci. Informatics*, 14(2):563–599, 2021.
- [98] F. Zhang, K. Wang, Z. Li, and J. Cheng. Temporal data representation and querying based on RDF. *IEEE Access*, 7:85000–85023, 2019.