Time Series Database Architectures and Use Cases

Published: 26 March 2020 **ID:** G00365928

Analyst(s): Sumit Pal

With an increasing need to ingest, manage and analyze time series data, the time series database market is evolving fast. This document gives data and analytics technical professionals insight into time series database architectures and capabilities.

Key Findings

- Time series databases solve two fundamental problems centered on read/write asymmetry: quickly ingesting and persisting large volumes of data, and reading data at an exponentially higher rate.
- Storage efficiency is extremely important for time series databases to ensure that they consume as few bytes possible of storage per data point. This is important to optimize storage and retrieval costs.
- Time series databases that have been built from ground up for time series data are significantly faster and more scalable than those that sit on top of non-purpose-built databases, primarily because the architecture is based on time series data characteristics.
- Organizations should not use time series databases when the data does not have the time dimension and analytics does not involve time-based insights.

Recommendations

To effectively manage enterprise time series data, data and analytics professionals should:

- Select time series databases based on their architectures across storage, single-node and multinode architecture and query capabilities (especially as data volumes and numbers of metrics scale), reliability, storage costs, and ingest and query performance. These will be critical decision factors for selecting a data store.
- Evaluate time series database vendor claims carefully based on domain requirements of ingest throughput, read latencies and concurrency. There is no perfect time series database for both high-speed ingest and low latency, highly concurrent querying requirements.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

- Scrutinize the pros and cons of querying time series databases because some do not support ANSI SQL. When looking to connect time series databases to business intelligence and data virtualization tools, carefully evaluate compatibility and capability issues on both sides.

Table of Contents

List of Tables

List of Figures

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Analysis

Time series data is everywhere. Time is an inexorable part of everything that is observable. With the growth of internet-connected and mobile devices and the decreasing costs of storage, the demand for time series databases has grown over the last decade. Increasingly, organizations are interested in analyzing this data to derive insights and make data-driven decisions. Technology advancements have improved the efficiency of collecting, storing and analyzing time series data, resulting an increased appetite to consume this data.

What Is Time Series Data?

Time series data is simply any set of values with a time stamp where time is a meaningful component of the data. A real-world example of a time series is stock currency exchange price data. Time series data is measurements or events that are tracked, monitored, down-sampled and aggregated over time. This could be infrastructure metrics, application performance monitoring (APM), network data, sensor data, events, clicks, trades in a market and other types of analytics data.

> The key difference with time series data from regular data is that it is always queried over the time dimension.

Time series data comes in two forms: regular and irregular. Regular time series consist of measurements gathered at regular intervals of time (every *n* seconds), for example, data coming from sensors like smart meters at a regular cadence. Irregular time series data is event-driven, either by a user or other external events such as clickstream data or data coming from a user interacting with a webpage. Once summarizations are applied on irregular time series data, it can be transformed to become a regular time series.

A time series dataset mainly has the following characteristics:

- New data is always stored and recorded as a new entry. Only data appends are allowed.
- Older data is immutable (no updates).
- Data is usually stored in chronological order.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

- All data is time-stamped.

Why Time Series Data

Time series data is evanescent and voluminous, which calls for different considerations for storage and retrieval than other types of data stores. Time series databases need to be designed from the ground up to address scalability, to handle large amounts of data used for example algorithmic trading, advanced time series analysis, and anomaly and trend detection.

Time series databases are optimized for the storage of high volumes of sequential data across time. They are often organized as columnar data stores that can write large amounts of data quickly. These systems can sometimes tolerate data loss, because the data is being gathered very quickly, and that data is usually used for monitoring and other applications that require aggregated datasets rather than high-fidelity, highly important individual transactions.

Some of the most common use cases for time series data include:

- **Internet of Things (IoT):** Much of the data generated by sensors in machine-to-machine communication in IoT is collected as time series. Time series databases for IoT enable deep insights from the data captured from devices. IoT device data is typically characterized through hundreds to hundreds of thousands of sensors or metrics, real-time queries under highly concurrent load with alerting, stream processing, and machine learning on gigabytes to terabytes of data. IoT data is unique in that it is generated at very high volumes and needs to be analyzed in complex ways quickly and reliably.
- **Systems monitoring:** With the shift from monolithic to microservices-based architectures, there has been an exponential increase of the number of metrics site reliability engineers (SREs) or DevOps teams collect and need to monitor in real time. This is all time series data that needs to be collected, stored and analyzed to discover usage pattern trends abnormalities. Time series databases provide ideal capabilities for this kind of data at this scale.
- **Business analytics:** Analyzing time series data from retail, marketing, advertising and transportation, etc., allows quick identification of trends and pricing, helping businesses to extract meaningful statistics and other characteristics.
- **Financial tech:** Time series financial analysis allows users to better understand the marketplace and improves their ability to generate quality forecasts.

Large real-time time series databases are becoming a key component of digitalization and Industry 4.0 transformation. Some established time series databases are enhancing their capabilities across multiple dimensions — scalability, performance, and SQL support, and integration with BI tools and other enterprise and analytic tools.

Apart from existing and established time series database offerings, cloud providers have entered the market with Microsoft Azure's Time Series Insights and Amazon Timestream. This is happening for multiple reasons. Cloud vendors entering the IoT market want to integrate their IoT platform and services by providing the most optimal storage for time series data. Storing time series data in non-

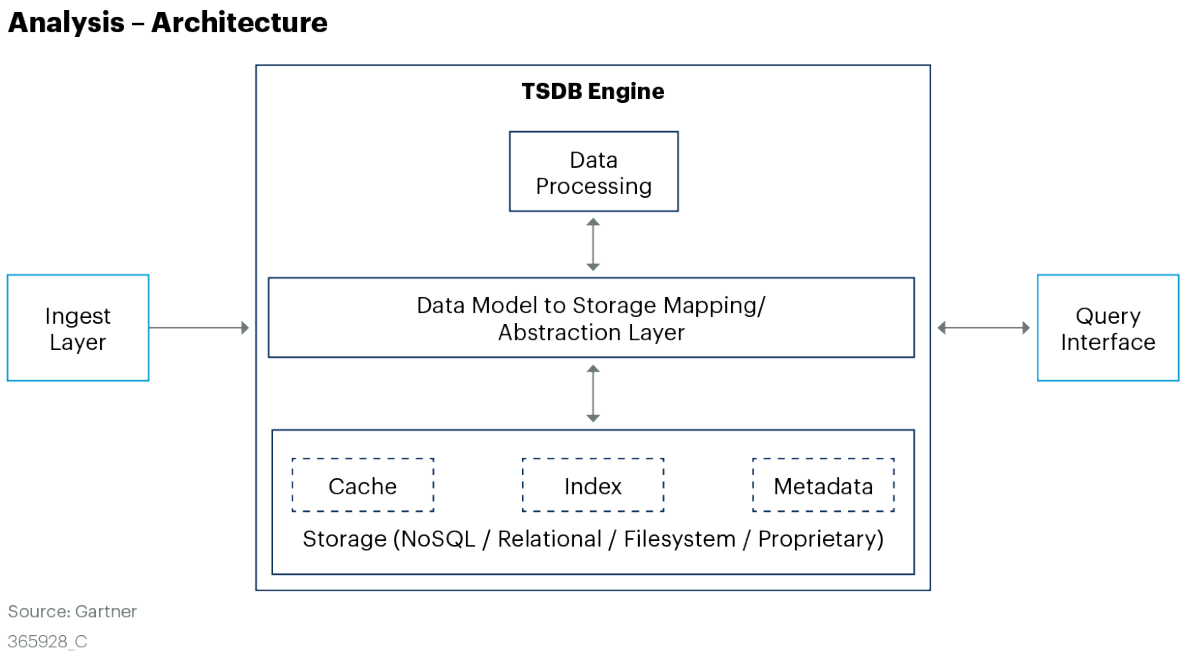
This research note is restricted to the personal use of tanlb@viettel.com.vn.

time-series data stores causes an impedance mismatch with the read, write workloads for time series analysis and storage optimization of voluminous time series data.

High-Level Architecture

Time series databases have all the components in a typical data store, with the added layer of the time series optimized data model, which varies across vendors to best represent the incoming data. The storage, query and data model layer are discussed extensively in the Details section of the document. Figure 1 shows a high-level architecture for a time series database.

Figure 1. High Level Architecture



- The ingest layer is used to ingest high-velocity data from multiple data sources with a wide variety of protocols like UDP, TCP or HTTP and interfaces.
- The storage layer is responsible for storing the data in the right format in concert with the associated data model that the native vendor implements to store the data. Storage layer also encompasses compression, caching, index and metadata storage.
- The metadata component stores the associated metadata primarily for internal bookkeeping by the database engine. This metadata layer is not exposed to the end users of the database. Metadata is used internally by the database engine for query optimization and data validation.
- The indexing component is an optional component primarily meant to quickly search and gain access to the underlying, often dense, time series data. Indexing is an expensive option from

This research note is restricted to the personal use of tanlb@viettel.com.vn.

storage and creation for high-velocity incoming data. Different databases use different data structures from inverted index to radix trees and log structured merge (LSM) trees.

- The caching component is primarily meant to improve query latency by avoiding querying the underlying storage engine to retrieve and deserialize the stored data.
- The data model component abstracts the way the time series data is internally represented with the storage engine of the time series database. This is explained in the Details section of the document.
- The query layer provides capabilities to query the data store, either using SQL or proprietary language along with client libraries and APIs to interface with the database.

Types of Time Series Databases

Time series databases can be classified into four or five categories based on their underlying storage implementation:

- Historian databases based on proprietary technology, which is still widely used IoT and manufacturing domains. Time series databases like operational historians, for example, are essentially powerful databases used to quickly write data while being deeply integrated into production processes that provide critical monitoring and dashboarding capabilities. However, operational historians are falling behind with advances in real-time analytics and with architectures that cannot work well with modern data requirements of volume, velocity and high performance under high concurrent loads. Historians were developed to handle large amounts of data from industrial machinery, but they are not designed to be distributed, to operate in real time or to support advanced analytics.
- Time series databases built on top of file systems or nonrelational data stores, which leverage the distributed architecture of these data stores to scale time series databases. These databases can be used to store unstructured or semistructured data along with time series, and have a flexible schema.
- Time series databases built on top of relational data stores leveraging the research that has gone into building optimized relational data stores. These types of time series databases have an abstraction layer on top of the relational engine to work with time series data. Such time series databases require schema and indexes to be defined upfront, unlike nonrelational ones.
- Time series databases built on proprietary database technologies built primarily for storing and analyzing time series data. These are native time series databases that are built from ground up to work with time series data, based on the usual characteristics of such data.
- Time series databases in the cloud are still evolving from pricing concepts and strategies. The cost could depend on the number of bytes of data ingested, number of queries, volume of storage memory/SSDs, retention times, complexity of queries and other parameters. Industrial time series working with very high-volume data and running thousands of queries per second to power manufacturing platforms need to be aware of the cost implications and unpredictability of total costs for time series databases in the cloud.

Table 1 lists the different categories of time series databases and sample vendors in each category.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Table 1. Type of Time Series and Corresponding Vendors

Types	Sample Vendors
Historian Database	OSIsoft (PI System), AspenTech, AVEVA (formerly Wonderware)
File System OR Nonrelational Data Store Back End	OpenTSDB, KairosDB, Blueflood DB, Gorilla, Heroic, Arctic Hawkular, Apache Chukwa, BtrDB, Databus, Kairos, SkyDB, Chronix Server, Metrictank, ClickHouse, Riak TS
Proprietary Data Stores	Elasticsearch, MonetDB, Prometheus, Apache Druid, InfluxDB, RRDtool, MongoDB Atlas, Gnocchi, Whisper, SciDB, BlinkDB, TsTables — HDF5 format, Warp 10, Akumuli, DalmatinerDB, TimeStore, BEMOSS, YAWNDB, Vaultaire, Bolt, NilmDB, QuasarDB
Relational Data Store Back End	TimescaleDB, MariaDB, MySQL, MemSQL, CrateDB
PaaS Cloud-Based	Amazon Timestream, Microsoft Azure Time Series Insights

Source: Gartner (March 2020)

Some of the advantages and disadvantages of each of the implementations are discussed in Table 2.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Table 2. Time Series Database Implementation Choices Pros and Cons

Options	Pros	Cons
Historian Database	■ Have been in use for longer timelines and hence more mature. ■ Better suited for short-term store on a dedicated .NET/Windows server. ■ Leverages existing investments in hardware and licensing. ■ Best suited for the operation technology (OT) part of the IoT architecture.	■ Proprietary technology, which may limit its ability to handle large volumes and variety in the platform. Difficult to integrate with platform databases. ■ Limited analytical options. For example, they may not support JSON formats.
NoSQL	■ Flexibility of schemas help with time series data storage ■ Better performance for writes	■ Lacks SQL interface ■ Can be difficult to manage and operate ■ Restricted options for secondary indexes ■ Complex predicates can be difficult to handle ■ JOINS not possible
Relational	■ Supports secondary index. ■ Easily support complex predicates ■ SQL interface. ■ Can easily relate it to relevant cross-sectional or lookup data stored in the same database.	■ Scaling challenges for single-machine architectures.

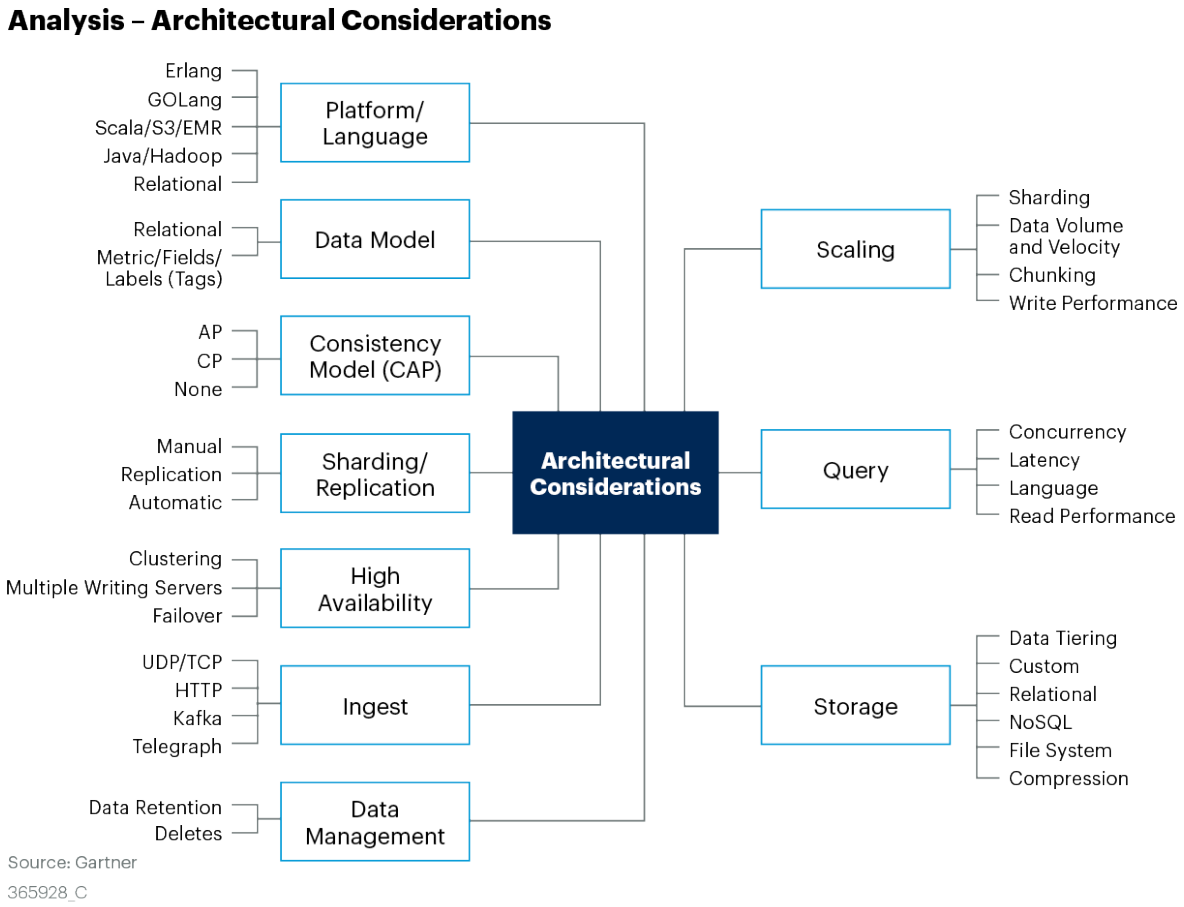
Source: Gartner (March 2020)

How to Choose a Time Series Database

There are several factors to consider when evaluating a time series database for your workload. Figure 2 shows some of the architectural considerations when selecting a time series database.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Figure 2. Architectural Considerations



When evaluating time series databases, follow the guidelines below along with a clear understanding of your domain and application requirements and data characteristics.

- **Data volume and velocity:** Have clear SLAs on how much data needs to be stored and how quickly the volumes will grow. Make sure the database of choice is able to scale with the data throughput and velocity, and not lose data as it is being ingested. Pick databases that provide autoscaling capabilities as data volume grows.
- **Data model:** Look into the data model of the underlying time store in terms of flexibility and the ease with which schemas can be extended and new indexes added. Dynamic schemas, text search, and user-defined functions are often required for a wide range of industrial time series analyses.
- **Query language support:** Most time series databases do not support fully compliant ANSI SQL. Look into query flexibility, ability to perform search, SQL joins, user-defined queries, subselects, anomaly detection, geospatial, window functions and CTEs. Select a database

This research note is restricted to the personal use of tanlb@viettel.com.vn.

where the query language eases selection of series, continuous querying capabilities and transformation of queried data. Some databases have better performance with recent data while some perform better on aggregates on historical data. Carefully analyze capabilities of time series databases for advanced time series analytical functions (gap filling, interpolation, etc.).

- **Data type:** Is your data regularly spaced or irregularly spaced? Understand the data types required by your domain, since some time series databases do not have good support for strings, Boolean or nested data types.
- **Performance:** Be skeptical about all claims that every time series database makes. Each solution had its pros and cons. Perform your benchmarking based on your workloads, throughputs, latency, concurrency and complexity of queries. It is essential to understand performance characteristics as the cardinality of the series increases over time and at higher concurrency.
- **Operational management capabilities:** If you want to deploy the database for industrial IoT-based operations, such as automated monitoring, automated backups and point-in-time restore, automatic upgrades, maintenance and automatic data retention policies are essential.
- **Integration:** Evaluate carefully how the database interfaces and integrates with external ingestion and data access tools and platforms, especially BI tools, and compatibility with tools like Prometheus, Grafana and Telegraf for ingest and visualization.
- **Product distributions:** Evaluate carefully the capability differentiations and distinctions between open-source and enterprise versions of the databases.
- **Scale-out capabilities:** Understand the scale-up (on same node) and scale-out (across multiple nodes) capabilities of the data stores before making a choice. Typically, for IoT workloads, choose a time series database that provides elasticity by adding as data grows and delete nodes as data ages out without any downtime.

Strengths

Some of the strengths of time series databases are given below:

- Most time series databases can ingest high velocity time series data across a wide spectrum of ingestion protocols and data transfer mechanics. This is important to provide wider integration capabilities across a variety of data sources.
- Support for multiple compression capabilities to reduce storage footprints. This is important for faster data scanning to support lower query latencies.
- Support for both regular and nonregular time series data within the same database. This allows time series data to be used for multiple-purpose use cases.
- Time series databases optimize for large scans over many records. They facilitate better query performance through column-oriented storage as well as preaggregation.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

- Some time series databases have data life cycle management processes to handle data as it ages. This can include aggregating older data to a higher level of detail, in addition to creating data retention policies.

Weaknesses

Here are some of the weakness of the time series databases on the market:

- Time series database vendors have not standardized a language for querying time series databases. Databases built on top of relational data stores have SQL support, while other databases that are proprietary or built on top of nonrelational data stores do not have SQL support.
- Some databases, like Apache Druid, do not support granularity beyond the millisecond level, which can be limiting for certain applications.
- There's limited integration with time series analysis tools outside the ecosystem, especially BI tools and other ingestion tools.
- Some databases are limited to single-node, and multinode capabilities are available only for enterprise editions.
- Some time series databases are not very good at handling high-cardinality data and exhibit higher query latencies for large cardinalities.
- Determining the optimal configuration settings for performance/throughput and concurrency can be tricky.
- Databases that do not support joins across measurements can be very limiting and can hamper querying capabilities to provide unified reporting capabilities.

Guidance

The following section provides guidance to technical professionals looking to use time series data stores for their data management solutions.

Time series data comes in two forms: ones that can store traditional regular metrics, and ones that can store irregular events. Time series databases such as Graphite, RRD and OpenTSDB support only regular time series metrics. However, modern time series databases like InfluxDB are optimized for both types of data.

Follow the guidelines as outlined in the previous section to choose time series databases. Some Gartner clients also need to store time series data and to perform graph processing on the time series data. For such organizations, multimodel data stores are recommended.

Before selecting a time series database:

- Understand the characteristics of the domain data.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

- Evaluate the read/write throughput and query latency requirements, especially under concurrent workloads across selected databases, before making a selection.
- Ensure the database is scalable incrementally, from a storage and query performance perspective, by adding more nodes.
- Understand the query capabilities and interfaces of these databases before integrating BI or analytics tools. Be sure that functions like changing the granularity, grouping by time spans, or performing autoregressive integrated moving average (ARIMA) time series analyses are supported.

There is no one single time series database that suits every use and supports all features. Having enterprise-ready performance is a crucial step of readiness for enterprise adoption.

You should prepare your team. There are a number of factors to consider when choosing either an open-source or commercial time series data store. Your requirements can be very different from those of large organizations with engineering and operations staffing. To leverage the power of time series data stores, teams must be empowered and trained with the right skill sets. Build the right skill sets to choose and use time series databases and leverage their power with the different ways of querying connect data.

Operational complexity is often ignored when selecting solutions. When everything else is equal between selected time series databases, the primary differentiator then becomes efficiency and overall complexity of operation. An optimal scenario would be for a time series database to automatically scale up and down as additional storage or compute is needed.

The Details

This section discusses the internals of a time series database, starting with the architectural components and a deeper dive into some of the important components — query engine, storage and the data model.

Architecture

Time is central in the overall platform architecture design. However, just being able to query on time doesn't cover all the requirements of an effective and efficient time series data store. To achieve the right scalability and usability of large and granular time series data, it is necessary to devise a combined strategy in data models and storage, and query engine design.

Time series databases should be optimized for fast reads and writes of unique points across multiple data series. Time series databases need to solve two fundamental problems — high write throughput and high query rates — with high concurrency . Each data point has a time stamp and actual field values and metadata. Fields are typically not indexed, while the metadata is indexed.

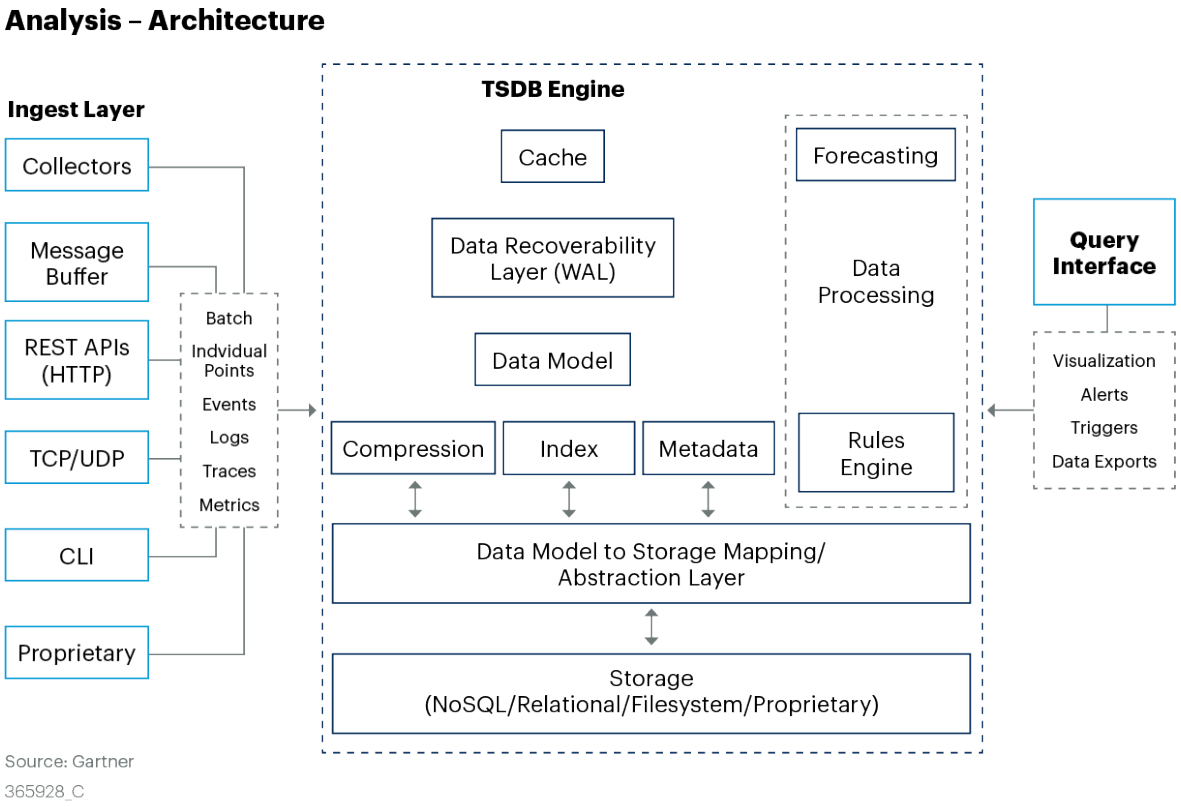
The real-time nature of time series data makes it necessary to expose new data in queries as fast as possible, although not every point matters in the series. Most time series data is summarized into intermediate values, since trends provide more insights than individual data points. Most time series

This research note is restricted to the personal use of tanlb@viettel.com.vn.

databases partition the data into chunks, each with its index and its own isolated storage modules. This benefits queries with a time predicate or a time-based aggregate. Filters result in less data scan and allow databases to choose an optimized plan and perform calculations in memory, compared to reading from disk. It also allows better use of algorithms like hash aggregates versus group aggregates.

Figure 3 shows the internal architectures and components of a time series database.

Figure 3. Architecture for Time Series Databases



The following sections cover details related to the architectural aspects of time series databases — data model, storage and query.

Querying Time Series Databases

With time series data, one is always asking questions over time. Time series queries are frequently aggregations over time, scans of points within a bounded time range that are then reduced by summary function like mean, max, or window-based analytic functions. Time series applications tend to query in two patterns.

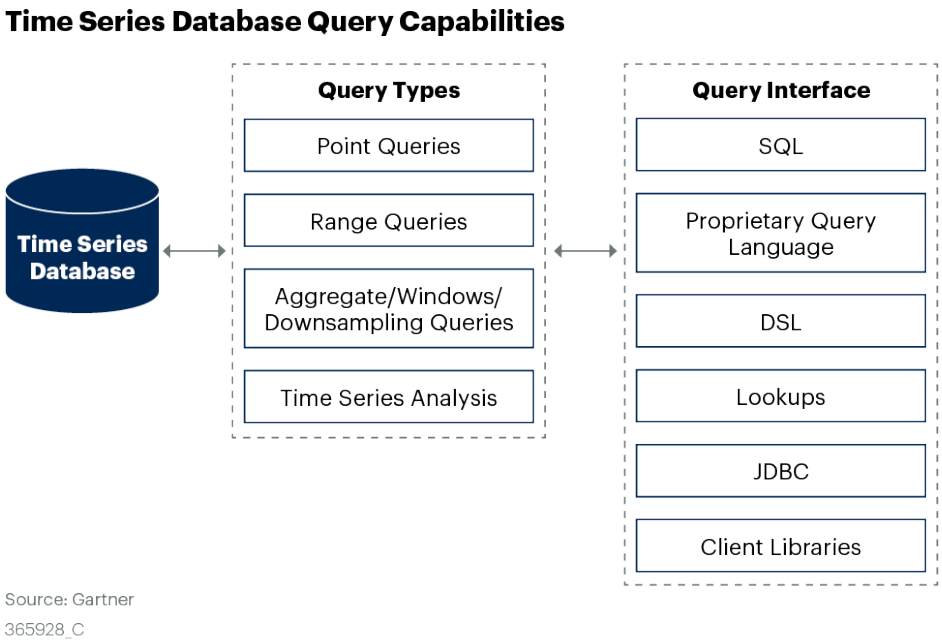
- Queries based on window on the time dimension produce window-based aggregates.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

- Point queries, often over the most recent point in a series.

Figure 4 shows the different query types in time series data and the access methods.

Figure 4. Time Series Database Query Capabilities



Time series data has its own specific query and computing methods, which roughly include the following types.

Point Queries

Depending on the underlying data model, either name, metadata or location can be used to query a data point. Typically, the timeline needs to be located first, which is retrieval-based on a combination of one or more metadata fields.

Range Queries

Most range queries in are based on the time dimension. Range queries have a starting timeline and ending timeline. Once these timelines are located, the data within these timelines are queries. Data for these timelines can be all colocated or spread out across multiple partitions, shards or chunks of datasets.

Aggregation

Data aggregation is a crucial component for reads performance. Aggregations can be asynchronously applied during ingestion, allowing for write performance optimizations as well.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Aggregations are accomplished by applying an aggregation function over data spanning a time interval. For range queries for multiple timelines, the results are usually aggregated. It is expected that time series datastores will support simple to complex aggregate functions like count, percentiles and cumulative_sum that are often needed to construct data visualizations from the raw data.

Downsampling

The logic of downsampling is similar to that of aggregation. The difference is that downsampling is for a single timeline instead of multiple timelines. Downsampling aggregates the data points in a time range on a single timeline. One of the primary purposes of downsampling is to display the data points in a large time range. Another is to reduce storage costs.

Time Series Analysis

Time series analysis is a vast and well-researched field. Time series databases could be easily used to perform advanced time series analysis with functionality like detecting stationarity of data, removing the stationarity, and calculating ARIMA- and SARIMA-based calculations on the underlying data.

Apart from the above querying requirements, time series also has specific querying needs, such as:

- Joining or merging two different time series into one.
- Resampling, which is aggregating a high-frequency time series into a lower-frequency time series.
- Trimming to quickly remove old data.

Query Engine

The query engine in a time series database typically performs the following steps to execute a query:

- Determines the type of query: point/aggregate/range query.
- Parses the query; performs semantic checking and build the query plan.
- Separates the time range and the condition expression for filtering data.
- Determines which shards/chunks (for a distributed time series data store) need to be accessed, using the list of measurements and the time frame of the query.
- Directs the storage engine to create the iterators for each shard/chunk.
- Merge the iterator outputs performing postprocessing on the data. These iterators are nested, forming a tree. The iterator tree is executed bottom-up, reading, filtering and merging data to produce a final result set.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Query Optimization

Most time series data queries are over summarized data because it provides trends. Real-time nature of time series makes it necessary to expose new data in queries as fast as possible. Optimizing query performance requires finding the initial point for each series and leveraging columnar storage to efficiently scan a sequence of points following that initial point. Time series databases usually store data on the disk in a column-oriented form that allows fast aggregations.

Most time series databases automatically maintain an in-memory index to make filtering efficient. Time series databases built on top of relational stores leverage secondary indexes on other filtering criteria. Secondary indexes are not possible for time series data stores built on top of nonrelational data stores. Indexes can be further optimized to maintain sketches of series and measurements for fast cardinality estimates.

Some time series data stores support intelligent rules for automatic rollups with caching and querying of lower-granularity data, mostly required for dashboards that zoom from short to longer time frames. Some time series databases don't push queries down to the nodes, which means it has to pull all the data points for a query across our network to the query engine to perform the calculations. This can result in larger latency due to network and I/O bottlenecks, and can quickly degrade for concurrent workloads.

Query Interface

Time series databases based on relational data stores typically support SQL-based query interfaces. However, there is no standard querying language for most time series databases. As cardinality increases, performance can drop dramatically, depending on the underlying storage architecture. SQL queries can easily become very verbose for complex time series calculations when using subselects and common table expressions (CTEs).

Select time series databases with a query language that eases selection of series, continuous queries, and transformation of queried data. The querying engine should enable advanced queries, such as adding and dividing time series together, and calculating percentiles on the fly.

If use cases require advanced time series analytics, carefully evaluate the capabilities of the query language and the engine if it supports time series analytic-related functions. Also evaluate if third-party libraries can be plugged in to support analytic query capabilities.

Data Model

Every time series database has an underlying data model that encapsulates and creates an abstraction of how the incoming data is stored in the underlying data store. Based on the underlying storage mechanism (as listed in Table 2), the data model is created on data ingestion for each data point, compressed and then sent to the storage layer.

Time series data can be thought of as a collection of data points or tuples — a set of fields. Each data point has a time stamp and a measurement associated with it. For example, metrics from a database server could be number of bytes sent, number of incoming connections, etc. These data

This research note is restricted to the personal use of tanlb@viettel.com.vn.

points also carry metadata about the measurement, for example, a fully qualified hostname generating the series.

One of the leading time series databases, InfluxDB, uses a model that is described below. A data point has four components: a measurement, metadata, fields and their values, and a time stamp.

The measurement provides a way to associate related points that might have different associated metadata or fields. Metadata about the data point is a dictionary of key-value pairs to store with a point. The fieldset is a set of typed scalar values — the data being recorded by the point. Fields represent the multiple metrics being tracked for a measurement and the values associated for those metrics. A set of metadata called the tagset represents the metadata fields and corresponding values describing the measurement that is being stored.

For example, for the above measurement of metrics from a database server:

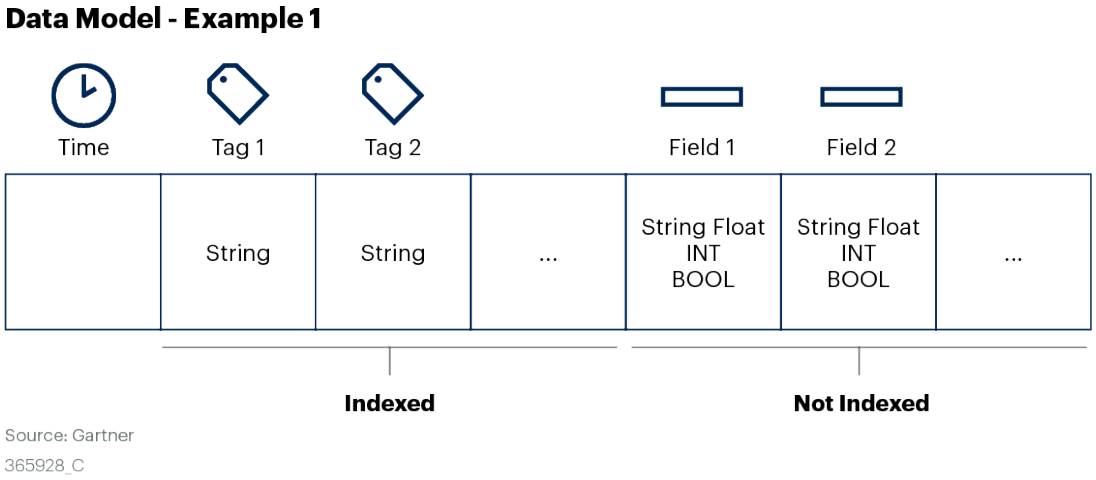
SQL Server;IP=72.12.45.124,Location=DC1;BytesSent=32,IncomingConnection=100

- The measurement is SQL Server.
- The metadata is IP, Location. The keys, IP and Location, in the tagset are called tag keys. The values, 72.12.45.124 and DC1, in the tagset are tag values.
- The fieldset is BytesSent=32, IncomingConnection=100. The keys, BytesSent and IncomingConnection, in the fieldset are called field keys. The values, 32 and 100, in the fieldset are called field values.

Each point is stored in a database with an associated retention policy. A series is defined as a combination of retention policy + measurement + tagset.

This data model looks as shown in Figure 5.

Figure 5. Data Model Example 1



Source: Gartner
365928_C

This research note is restricted to the personal use of tanlb@viettel.com.vn.

There are advantages and disadvantage of this model.

Advantages:

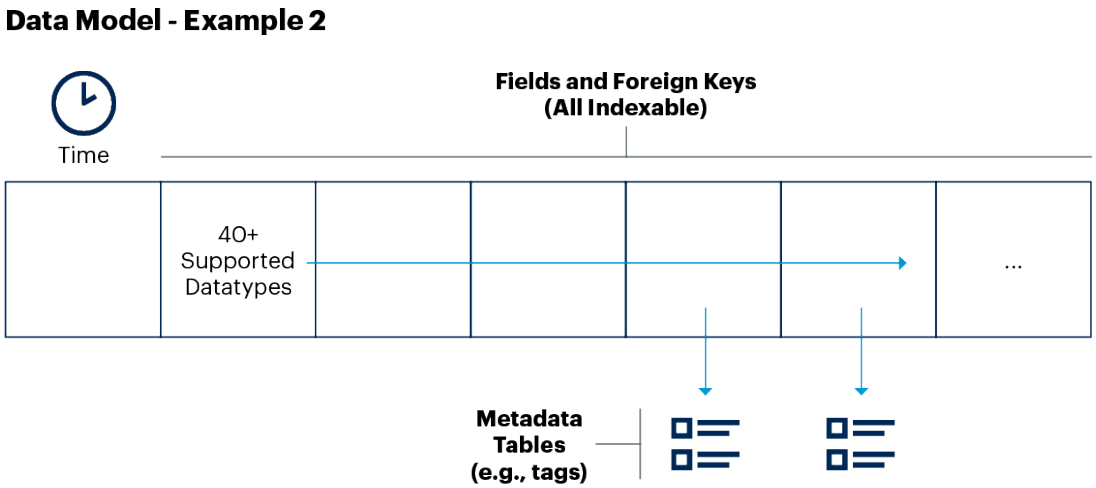
- If the data fits the model, it is easy to get started.
- It does not have to think about creating schemas or indexes.

Disadvantages

- Enforcing data validation can be problematic.
- Lack of support for creating any additional indexes.

Another leading time series database, TimescaleDB, which is based on Postgres, has a different data model for storing time series data. The data model is based on the relational data model and looks like Figure 6.

Figure 6. Data Model Example 2



Source: Gartner
365928_C

With the above example, in this model the data would look something like Tables 3, 4 and 5.

Table 3. Time Series Table

Time stamp	BytesSent	IncomingConnections	TagID1	TagID2
2020-02-11T06:38:45.481	32	100		

Source: Gartner (March 2020)

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Table 4. Tag Table1

TagID	TagValue
IP	72.12.45.124

Source: Gartner (March 2020)

Table 5. Tag Table2

TagID	TagValue
Location	DC1

Source: Gartner (March 2020)

Some of the advantages and disadvantages of this model are:

Advantages:

- A narrow or wide table, depending on how much data and metadata to record per reading.
- Choice to have multiple indexes to improve query speed, or fewer indexes to reduce disk usage.
- Denormalized metadata within the measurement row, or normalized metadata that lives in a separate table, either of which can be updated at any time.
- A rigid schema that validates input types.
- Check constraints that validate inputs, for example checking for uniqueness or non-null values.

Disadvantages:

- Index maintenance.
- Choose schema early on
- Choosing which fields to index.
- Index creation and maintenance, and overhead.

Storage

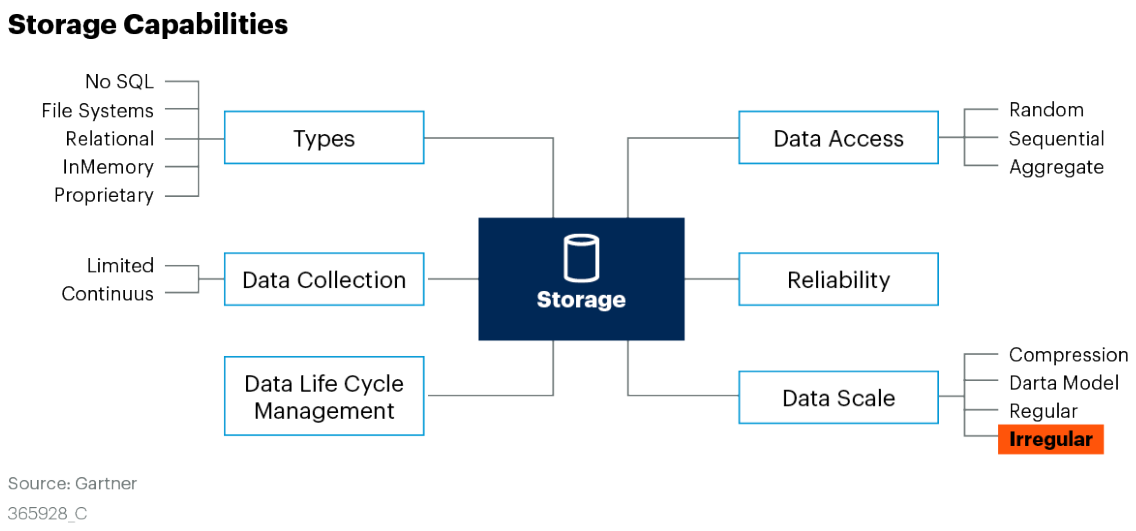
Given the constant influx of granular data from a number of data sources in modern architectures, a performant database solution for time series has to handle high write rates with the right kind of storage architecture both for in-memory and durable storage. The storage architecture is equally important to support concurrent queries for low-latency analytics for dashboards.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

In a time series database, new data is always stored and recorded as a new entry. Data is usually stored in chronological order. To scale to over a million metrics a second from thousands of devices, storing terabytes of data, and processing queries even with better parallelization, storage efficiency is extremely important. To optimize storage, time series database should consume as few bytes of storage per data point as possible with an optimized data model, and selecting high compression ratios. Most data points in time series don't change much between readings, lending to better compression and shrinkage of storage resources.

Figure 7 shows a mind map of the aspects of storage a time series database should consider when the underlying storage engine is selected or designed from ground up.

Figure 7. Storage Capabilities



Time series queries are frequently aggregations over time and scans of points within a time range that are then reduced by summary functions like mean, max, or moving windows. Columnar storage techniques compress data exceptionally well, and the data is organized on disk by column, satisfying the need to store data efficiently. However, deleting points from columnar storage is expensive, so it is often necessary to organize the columnar format into time-bounded chunks. When the time-to-live (TTL) expires, the time-bounded chunks can be deleted rather than requiring updates to persisted data.

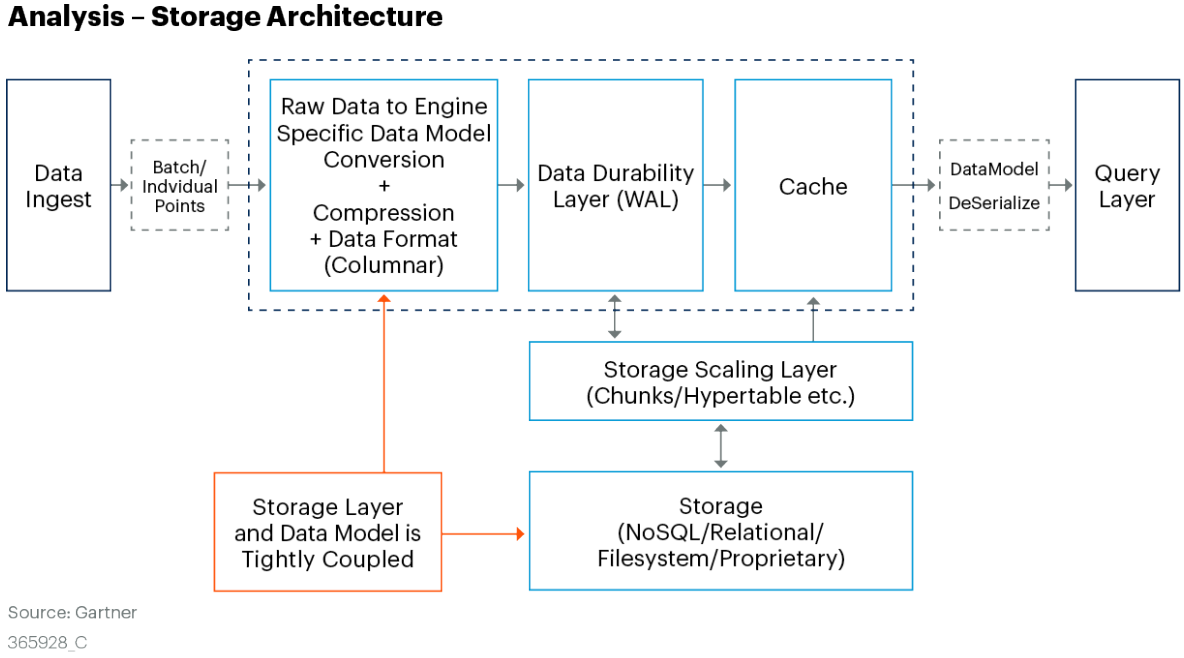
Most time series databases use append-only files for new data arrival to make new points quickly ingestible and durable. Each batch is written and file-synced to a write ahead log (WAL). The WAL is an append-only file that is only read during a recovery process. For space and disk I/O efficiency, each batch in the WAL is also compressed before writing to disk.

WAL efficiently makes incoming data durable, but it is poor for data reads, making it unsuitable for querying. To allow immediate access to the ingested data, incoming points are also written to an in-memory cache for query performance. The WAL makes new points durable. The cache makes new

This research note is restricted to the personal use of tanlb@viettel.com.vn.

points queryable. WAL and cache storage is, however, insufficient for long-term storing needs. The WAL is be replayed on startup; hence, it is necessary to constrain it to a reasonable size. Figure 8 shows a high-level architectural diagram of how data gets stored and accessed in most time series databases, from ingest to query.

Figure 8. Storage Architecture



Vendor Comparison

The final section of this document compares four major time series vendors across the architectural considerations discussed in this document. Table 6 shows the four time series databases, their features and capabilities, across some of the most important architectural questions.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Table 6. Comparison Matrix of Selected Time Series Databases

Properties	TimescaleDB	InfluxDB	CrateDB	QuasarDB
ANSI SQL interface	Yes	InfluxQL is a SQL-like interface to query data.	Yes	Subset of ANSI SQL.
Deployment modes	On-premises, cloud, edge, Kubernetes-enabled.	Any. Single binary deployed on *nix OSs, on-prem or cloud. Offers InfluxDB as a PaaS — on a pay-as-you-go basis.	Yes	Cloud and on-premises.
Columnar indexes and compression	Hybrid row/columnar.	Yes	Yes	Columnar store + compression.
Data aging/retention	Automated support.	Yes	Yes — optimized with partitions, but requires external code.	Data retention based on storage size.
MQTT interface	Yes, third-party tools, OSS, commercial.	Supported it in all products via different methods.	No. Has been removed with 4.0	No
Data model	Relational data model.	There are a number of organizing principles including: measurements, fields, tags and time stamp. A brief description can be found here.	Dynamic schema.	Time series stored as tables — column-oriented.
Support for joins	Yes.	Yes	Yes	Yes
Support of nested data	Yes. JSON support.	Yes	Yes	No
Support of blob data	Yes	Yes, as an encoded string.	Yes	Yes
Written in	C	Go	Java	C++ 20
Support for estimate-based query	Yes	Yes	Yes	No
Postaggregation filtering (group by having)	Yes. Supports “Having” clause and postaggregation filters.	Yes	Yes	Yes
Core architecture	Hypertable with time partitioning.	More information about our enterprise	Masterless (no SPOF).	Distributed hash table clustering

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Properties	TimescaleDB	InfluxDB	CrateDB	QuasarDB
		clustering model can be found here.		and multithreaded networking engine.
Consistency model	Postgre with MVCC, supporting read committed, repeated reads, and serializable.	Eventual consistency.	Eventually consistent.	MVCC transactions
Scalability model	Replica clustering via Postgres, both horizontal and vertical scaling.	Single server (scale out in Enterprise Edition).	Scale out.	Symmetric clustering based on distributed hash table.
Partitioning	Yes, using Hypertable with multidimensional partitioning.	Yes	Automatic	Transparent partitioning based on multiple dimensions (time, tables, etc.).
High availability	Yes. PostgreSQL replication with HA replicas and automated failover/ leader election.	Only in enterprise version.	Replication and failover replicas.	Yes
User-defined functions	Yes. Using PL/pgSQL extensions, UDFs in Python, JavaScript.	Yes	Yes	No
Text search	Yes	Yes	Yes	Yes (regex)
Geospatial queries	Yes, via PostGIS.	Yes — set for release in 1Q20.	Yes	Yes
Backup and restore	Yes. Supports both full dump and restore, and incremental streaming backups.	Yes	Yes	Native replication built in, configurable level of redundancy.
UI for monitoring / admin	Yes. Monitored/ administered via OSS and commercial tools including pgAdmin, Dbeaver.	Yes	Yes	Yes
Additional index support	B-tree, Hash, GiST, SP-GiST, GIN and BRIN. Both single and composite indexes are supported.	Tags are used for indexing.	Automated — everything is indexed.	Yes (stochastic indexes)
Time stamp granularity	Native time stamps are in microsecond granularity. Rich support for time	Nanosecond	Milliseconds	Nanosecond

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Properties	TimescaleDB	InfluxDB	CrateDB	QuasarDB
	stamps with time zones, time stamps without time zones, and date types.			
Autoaggregation	Yes. Automated continuous aggregations and “just-in-time aggregates.”	Aggregation can be customized and then run automatically as needed.	No	No
Advanced analytical functions	Yes	Yes	Yes	Yes
Containerization	Yes. Includes containers (via Docker or K8s).	Yes	Yes — works well with Docker and Kubernetes.	Yes
Metadata storage	Supported. Store metadata in separate tables.	boltdb or etcd	Yes	Yes
Data life cycle management	Automated data life cycle management, data retention, continuous aggregation, downsampling, data tiering.	Retention policies are inbuilt as a first-class concept; evicts the data after the specified period.	Yes (scripting with DevOps).	Yes (data retention support built in).
Security ACLs	RBAC at database, schema, table, column, and row-level security. Integration with LDAP, SSL and others.	Yes, in enterprise product. See: “Manage Security in InfluxDB Enterprise.”	Yes	YES (advanced ACL)
High cardinality support / performance	Supports high cardinality.	Yes	Yes	Unlimited
Max column number support /series	1600+ columns by default per hypertable.	No limit.	~10k	Unlimited
Cloud availability	Yes. Timescale Cloud is available on AWS, GCP and Azure.	Yes	AWS, Microsoft Azure, GCP.	Yes
Replication and sharding	Traditional database primary/replica clustering via PostgreSQL native capabilities.	Yes	Yes	Yes
Ingest	Optimized for high-velocity ingest (ODBC/JDBC), bulk copy and many others.	200+ data sources via Telegraf agent.	Bulk and single.	Via connectors, native APIs, and tooling.

This research note is restricted to the personal use of tanlb@viettel.com.vn.

Properties	TimescaleDB	InfluxDB	CrateDB	QuasarDB
Compression ratio	Highly optimized, type-specific compression algorithms.	See "In-memory indexing and the Time-Structured Merge Tree."	Depends on data around 0.7	From 1 to 100x (data dependent)
Secondary index	Yes. Table and hypertable support B-tree, Hash, GiST, SP-GiST, GIN and BRIN and as either single-key and composite-key indexes.	No	Yes	
Data type	Most supported.	String, int, float, Boolean, time stamp.	Boolean, char, smallint, integer, bigint, real, double precision, text, ip, time stamp with time one, time stamp without time zone, interval, geo_point, geo_shape, object, array.	
ACID support	Yes	No	No — we are eventually consistent and have a read after write consistency	
Query optimizer	PostgreSQL query optimizer.	Yes	Yes — cost based for some cases.	
ANSI SQL interface	Yes	InfluxQL is a SQL-like interface to query data.	Yes	Subset of ANSI SQL.

Source: Gartner (March 2020)

Gartner Recommended Reading

Some documents may not be available as part of your current Gartner subscription.

"Assessing the Optimal Data Stores for Modern Architectures"

"2020 Planning Guide for Data Management"

"Hype Cycle for Data Management, 2019"

This research note is restricted to the personal use of tanlb@viettel.com.vn.

“5 Useful Ways to Use Artificial Intelligence and Machine Learning With Your Logical Data Warehouse”

“Market Guide for Event Stream Processing”

“Market Guide for AIOps Platforms”

“Market Guide for Database Platform as a Service”

This research note is restricted to the personal use of tanlb@viettel.com.vn.

GARTNER HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
USA
+1 203 964 0096

Regional Headquarters
AUSTRALIA
BRAZIL
JAPAN
UNITED KINGDOM

For a complete list of worldwide locations,
visit http://www.gartner.com/technology/about.jsp

© 2020 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates. This publication may not be reproduced or distributed in any form without Gartner's prior written permission. It consists of the opinions of Gartner's research organization, which should not be construed as statements of fact. While the information contained in this publication has been obtained from sources believed to be reliable, Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Although Gartner research may address legal and financial issues, Gartner does not provide legal or investment advice and its research should not be construed or used as such. Your access and use of this publication are governed by Gartner Usage Policy. Gartner prides itself on its reputation for independence and objectivity. Its research is produced independently by its research organization without input or influence from any third party. For further information, see "Guiding Principles on Independence and Objectivity."

This research note is restricted to the personal use of tanlb@viettel.com.vn.