The significance of the time-series database and an introduction about TDengine

1. From time-series data to time-series database

Understanding time-series data and how it differs from ordinary data is the first step in grasping the significance of time-series databases (TSDB). A time series is a collection of data points that may be used to track changes over time. Data from time series can monitor changes over periods of seconds, days, or even years. What is the origin of time-series data in our world? Our understanding of time-series data used to be more static, which regrettably obscured the subtleties of how underlying changes through time influenced these static values. Let's take retail sales as a typical example. Time-series data analysis is very helpful for examining sales trends on a monthly, seasonal, and annual basis. As a result, retail establishments are better able to forecast their sales for the approaching season as well as the number of workers and merchandise they will require at certain points during the year.

Based on the above definitions and examples, we can conclude the difference between time-series data and cross-sectional data. Data in a time series are observations of a particular subject made over a period of time. However, Cross-sectional data are observations made on a large number of participants all at once. The same variable is the focus of time series data over an extended period of time. Contrarily, cross-sectional data concentrates on multiple factors at the same time. Time series data and cross-sectional data are different in that time-series data focus on a single variable across time, whereas cross-sectional data focuses on multiple variables at the same time. Different data kinds require various analysis techniques. Consequently, it is critical to determine the suitable data type.

According to its name, the time-series database is a unique database created to store time-series data. As previously said, a time series is a collection of data points with timestamps connected to them. A (TSDB) is a computer system created to store and retrieve data records that are a part of a "time series." Each data point's relationship to other data points is critically framed by the timestamps. Time series data frequently consist of a continuous stream of information, such as sensor measurements and intraday stock prices. Large amounts of time-stamped data can be stored in a time-series database in a format that supports quick insertion and quick retrieval to support complicated analysis of the data.

The following are some elements a TSDB should equip:

- Relational databases are organized in table rows, with each row representing an entry and each column representing various aspects of that entry. This is known as

a columnar layout. Some applications rely on column layouts, much like those used for high-frequency stock trading. Many specialized databases place a strong emphasis on columnar organization because they combine time series data with column-oriented storage.

- Such a database will arrange data according to timed occurrences using timestamps. Applications can stream information based on timestamps using this method. Time series databases can store system-specific data types that reflect timestamps, whereas relational databases can only create timestamps.
- For applications in banking, retail, or data analysis, time-series databases should allow specialized types of time series analysis.
- Systems that manage time-series data frequently gather enormous amounts of data—hundreds of gigabytes per day—from various sources. Compression techniques that make use of time series features can be used on time series platforms and databases. By compressing data depending on the variations between database items from one timestamp to the next, delta encoding, for instance, can reduce data storage sizes.

TSDBs function by recording a combination of fixed and dynamic values. The ideal format for time series records to be stored in a repository is one that allows for rapid time-based writes and reads. The sequence of the data points becomes a natural property of the data because the records are time-stamped. A stream processing engine can then treat the ordered data like a data stream by using this order to send the data to it. Utilizing a quick stream processing engine will help to guarantee the speed of your TSDB.

2. Why does TSDB matter?

Why we can't just utilize relational database management systems (RDBS), which are usually thought of as general-purpose database systems, to deal with the time-series data is a question that some people may have. This is due to TSDB's core capabilities, which allow it to store significant amounts of time-stamped data in a manner that supports quick insertion and quick retrieval in order to facilitate complicated analysis of the data. There are two key benefits to using a TSDB for your time series data, that is scalability and usability.

- Scalability

Normal databases are not built to handle the scale at which time-series data accumulates. Relational databases typically perform badly with extremely large datasets, but NoSQL databases perform well at scale. On the other hand, time-series databases, whether relational or NoSQL-based, bring efficiencies that can only be achieved when time is treated as a first-class citizen. They are able to deliver tremendous scalability thanks to these economies, which range from performance upgrades like faster queries at scale

and higher ingest rates to improved data compression. Therefore, scalability refers to a time series database's specialization in handling a higher volume of writes with eventual consistency, even over distributed storage, which means less anxiety for the individuals who care about that data.

- Usability

Additionally, built-in time-series data analysis tools including continuous queries, customizable temporal aggregations, and data retention restrictions are frequently found in TSDBs. These capabilities can still improve user experience and simplify data analysis processes, even if you're just beginning to collect this kind of data and scalability is not an issue right now. No matter how big or little your dataset is, having built-in functions and capabilities to evaluate trends immediately available at the data layer typically enables you to find opportunities you weren't aware of. We now have a query language that is specifically designed for the task at hand, which is not to see data as it relates to other elements of the schema but to view data in the context of the time in order to aggregate, define windows, or analyze patterns. There are millions of data points to search through. It doesn't matter if other databases can perform this; the issue is how we decide to allocate our resources.

For a number of use cases, developers are increasingly using time-series databases for the aforementioned benefits: asset tracking software, physical system monitoring, financial trading systems, and monitoring software systems. Let’s take the Internet of things (IoT) devices as an example. An IoT database for IoT scenarios where remote devices are continuously gathering measurements for analytical reasons would be a general example use case. The aforementioned oil well example is a typical IoT use case where the evaluation of a number of metrics from an oil well can assist with preventative maintenance. The evaluation can result in a prediction of when the equipment will fail based on trends and other factors that are represented in the data. The enormous amount of acquired data would be entered into a TSDB, after which apps would be run on the database to provide the analytics.

3. Introduction to TDengine

TDengine is an open-source big data framework. It is a big data system designed for the Internet of things (IoT). For the processing of massive amounts of data, it is a free big data platform that is extremely scalable, dependable, and performant. For lowering development and maintenance expenses, TDengine includes features including caching, stream computing, message queuing, and many more. It functions as a quick platform for storing, searching, and analyzing time-series data as well as a relational database. This open-source big data platform is at least ten times faster than other databases because of its storage design. TDengine is a high-performance, scalable time-series database with SQL support. In addition to the database engine, it offers caching, stream processing, data subscription, and other features to make development and operation

less complicated and expensive.

We can conclude from the documentation of the TDengine[1] that it makes full use of these characteristics of time series data to build its own innovative storage engine and computing engine to differentiate itself from other time-series databases, with the following advantages.

- **High Performance:** With an innovatively designed and purpose-built storage engine, TDengine outperforms other time series databases in data ingestion and querying while significantly reducing storage costs and compute costs.
- **Scalable:** TDengine provides out-of-box scalability and high availability through its native distributed design. Nodes can be added through simple configuration to achieve greater data processing power. In addition, this feature is open source.
- **SQL Support:** TDengine uses SQL as the query language, thereby reducing learning and migration costs while adding SQL extensions to better handle time series. Keeping NoSQL developers in mind, TDengine also supports convenient and flexible, schemeless data ingestion.
- **All in One:** TDengine has built-in caching, stream processing, and data subscription functions. It is no longer necessary to integrate Kafka/Redis/HBase/Spark or other software in some scenarios. It makes the system architecture much simpler, cost-effective, and easier to maintain.
- **Seamless Integration:** Without a single line of code, TDengine provides seamless, configurable integration with third-party tools such as Telegraf, Grafana, EMQX, Prometheus, StatsD, collected, etc. More third-party tools are being integrated.
- **Zero Management:** Installation and cluster setup can be done in seconds. Data partitioning and sharding are executed automatically. TDengine's running status can be monitored via Grafana or other DevOps tools.
- **Zero Learning Costs:** With SQL as the query language and support for ubiquitous tools like Python, Java, C/C++, Go, Rust, and Node.js connectors, and a REST API, there are zero learning costs.
- **Interactive Console**: TDengine provides convenient console access to the database, through a CLI, to run ad hoc queries, maintain the database, or manage the cluster, without any programming.

[1] https://docs.tdengine.com/intro/