

Topic Purpose **The role of time series database**

Student's Name **Peng Ruijie**（彭瑞杰）

Class schedule The Design and Application of Time-series Database

Study no 2019141460484

AD Industry Software engineering

Tutor Jeff Tao

July 8, 2020

**The role of time series databases**

## Time series database definition

Time series databases are mainly used to refer to the processing of time-stamped (sequentially changing according to time, i.e. time-serialised) data, with time-stamped data also known as time series data. The storage and processing of time-series big data is often handled using relational databases, but the inherent disadvantages of relational databases make it impossible to store and query data efficiently. By using a special storage method, the time-series big data solution allows for efficient storage and fast processing of large volumes of time-series big data, and is an important technology for solving the problem of large volumes of data processing. The technology uses special data storage methods to greatly improve the processing capability of time-related data, halving the storage space and greatly increasing the query speed compared to relational databases. The superior query performance of time series functions far exceeds that of relational databases. In the relational database, the time is the main key to do a column for sorting can also achieve the requirements of time series data query, but the efficiency is far inferior to the time series database.

Time-series databases are databases that are optimised for time-stamped data, which means that specific usage scenarios will only produce data with significant timestamps. We come across data from smart meters, car driving records, machine tool equipment monitoring, for example, and it is clear that the amount of data processing generated in such and such usage scenarios is in the trillions. It is in this context that the chronological database comes into its own.

## Characteristics of chronology column database

• Write smooth, continuous, high concurrency and high throughput: Sequential data is written smoothly, unlike application data, which is usually proportional to the number of visits to the application, which usually has peaks and troughs. The generation of time series data is usually at a fixed time frequency, which is not restricted by other factors, and the data generation speed is relatively stable.

• Write too much and read too little: 95%-99% of operations on sequential data are write operations, typically writing too much and reading too little data. This is related to the characteristics of the data, such as monitoring data. You may have a lot of monitoring items, but you may not really read the data, and usually only care about a few specific key indicators or in a specific scenario.

• Write the latest data in real time, no update: Write the sequential data in real time, and each write is the latest data, which is related to the characteristics of data generation, because the data generation advances over time, while the newly generated data is written in real time. There is no update in data writing. In the dimension of time, as time goes on, every data is new data and there is no update of old data. However, manual data revision cannot be ruled out.

• Large amount of data: In the Internet of Things scenario, the size of the entire data is TB or even PB.

• Clear hot and cold: time series data have typical hot and cold characteristics. The more historical data, the lower the probability of being queried and analyzed.

• Time-sensitive: Time-series data is time-sensitive and usually has a retention period. Data beyond this retention period can be considered invalid and can be recycled. On the one hand, the more historical data, the less available value; On the other hand, low-value data can be cleaned up to save storage costs.

• Multi-precision data storage: It is mentioned in the characteristics of query that timing data needs a multi-precision query for storage cost and query efficiency, as well as multi-precision data storage.

## The main time series database

1. Tdengine

TDengine allocates a fixed area of memory to the database, and newly inserted data, is written to memory first. TDengine also ensures that the last record of any device must be in memory, so that if an application wants to get the latest data or status of each device, it will get it directly from memory, and this design allows the system to eliminate the need for This design allows the system to eliminate the need for software such as Redis.

TDengine is designed to be completely transparent to the user in terms of the handling of historical and real-time data, with no distinction between historical and real-time data. All you need to do is specify the time period in the SQL statement and TDengine automatically decides whether to fetch the data from memory, from the local hard disk or from network storage, making the implementation of the application simple.

The data for each device is stored in blocks, and each block is already pre-aggregated (e.g. sum, maximum, minimum, etc.), so it is possible to perform various statistical operations for a device for a time period without having to scan the raw data, resulting in a significant performance increase. Even if some calculations require scanning of the raw data, the speed of the calculation and analysis is dramatically increased as the data is stored piece by piece in a continuous manner and the read speed is far faster than a general purpose database. And because of the structured storage, after decompression, no parsing is required, and the data can be read into memory for direct calculation, the speed of calculation and analysis is also greatly improved compared to NoSQL databases.

TDengine defines a new concept - the super table - to describe devices of the same type. TDengine has also designed a special mechanism that requires only one scan of the data file for multiple device data aggregation, which significantly reduces the number of IO operations and increases the speed of aggregation calculations. TDengine is ideal for use as a data warehouse for the Internet of Things, Internet of Vehicles and Industrial Internet.

2. InfluxDB

InfluxDB is an open source distributed timing, event and metrics database written in Go language without external dependencies. The database is now primarily used to store data involving large amounts of timestamps, such as DevOps monitoring data, app metrics, loT sensor data and real-time analytics data.

As the highest ranked open source timing database, InfluxDB supports data storage policy (RP) and data archiving (CQ), can be queried in real time, data can be found immediately after being indexed at write time, has a built-in HTTP interface, is easy to install and manage, and is very efficient at reading and writing data.

3. Kdb+

kdb+/q is officially known as the world's fastest time series database, it uses a unified database to handle real-time data and historical data, while having CEP (complex event processing) engine, in-memory database, disk database and other functions. The nature of columnar storage makes it exceptionally easy to operate statistical analysis for a particular column.

Compared with general databases or big data platforms, kdb+/q has faster speed and lower total cost of ownership, which is ideal for massive data processing and is mainly used for massive data analysis, high-frequency trading, artificial intelligence, Internet of Things and other fields. In the financial sector, where latency is critical, kdb+ has a unique advantage.

4. Prometheus

Prometheus is an open source system monitoring and alerting framework, created by former Google employees working at SoundCloud in 2012, developed as a community open source project, officially released in 2015 and officially joined the Cloud Native Computing Foundation the following year.

As a new generation monitoring framework, Prometheus has a powerful multi-dimensional data model, with a variety of visual graphical interfaces, using pull mode to collect time series data, you can use the push gateway to push time series data to the Prometheus server side.

## Challenges encountered with sequential databases

Many people may think that adding a timestamp column to a traditional relational database will work as a time-series database. It is true that this is fine when the amount of data is small, but a small amount of data is presented with limited latitude, little detail, low confidence and even less use for big data analysis. It is clear that temporal databases are designed to solve massive data scenarios.

It can be seen that a temporal database needs to solve the following problems

- Writing of time-series data: how to support the writing of tens of millions of data points per second.

- Reading of time-series data: how to support grouping and aggregation of hundreds of millions of data at the second level.

- Cost-sensitive: the cost of storing large amounts of data is a problem. How to store this data at a lower cost will be one of the most important issues to be addressed in a time-series database.

Here we will only try to answer the question of how to solve the problem of writing and reading large amounts of data from the perspective of data storage.