

Subscribe to DeepL Pro to edit this document.  
Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)for more information.

**Why do we need a time-series database?**

Most of the databases in use today are relational databases, so first we need to know what a relational database is and what features it has.

Relational database, is a database built on the basis of the relational database model, with the help of concepts and methods such as set algebra to deal with the data in the database, but also a set of tables organized into a formal descriptive form, the essence of the role of the form is loaded with data items of a special collection, the data in these tables can be accessed or reconvened in many different ways without the need to reorganize Database tables. The definition of a relational database results in a table of metadata or a formal description of tables, columns, ranges, and constraints. Each table (sometimes referred to as a relationship) contains one or more kinds of data represented by columns. Each row contains a unique data entity that is the kind of data defined by the column. When creating a relational database, you are able to define the range of possible values for a data column and further constraints that may be applied to that data value. In other words, the first major function of a relational database is to store data, and the second major function is to define the data and the data between different tables to facilitate later manipulation of the data.

So what are the advantages of relational databases over ordinary tables for storing data? First, relational databases are more convenient to use. Compared with a series of time-consuming operations such as looking up tables, relational databases support data operations using SQL language, which greatly reduces the workload and improves the accuracy of data operations. Second, it is easier to maintain the data in relational database. Because the relational database defines complete and rich entity integrity, referential integrity and user-defined integrity, it greatly reduces the probability of data redundancy and data inconsistency in maintenance, and can effectively reduce the occurrence of errors. Third, the security of relational database is better. In the relational database by selectively exposing to the user part of the interface and the use of views can effectively protect the security of the relational database.

Although the use of relational databases is very widespread in the era of big data, but relational databases still have some disadvantages. First, the high concurrency read and write ability is poor: the concurrency of website class user access is very high, and the maximum number of connections to a database is limited, and the hard disk I/O is limited, so it can not meet the simultaneous connection of many people. Second, the read and write efficiency of massive data is low: if the amount of data in the table is too large, the read and write rate will be very slow each time. Then, poor scalability: In a general relational database system, the ability of data processing can be improved by upgrading the hardware configuration of the database server, i.e., vertical expansion. But the vertical expansion will eventually reach the bottleneck of hardware performance and cannot cope with the demand of the explosive growth of Internet data. Another way of expansion is horizontal expansion, that is, the use of multiple computers to form a cluster to jointly complete the storage, management and processing of data. This horizontally scaled cluster can meet the demand for storage and processing of massive data by decentralized storage and unified management of data. However, due to the characteristics of relational database such as data model, integrity constraints and strong consistency of transactions, it is difficult to achieve a highly efficient and easily horizontally scalable distributed architecture. Finally, relational databases are very difficult for time-related data processing.

To solve the problems of relational databases, we had to develop a new data storage model to accommodate the ever-expanding volume of data.

The storage and processing of temporal big data are often handled by 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 temporal big data solution makes it possible to efficiently store and quickly process massive temporal big data, which is an important technology for solving massive data processing. This 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.

In simple terms, a time series database is a database dedicated to storing and processing time series data, supporting efficient reading and writing of time series data, highly compressed storage, interpolation and aggregation. This data is commonly found in metrics from servers and applications, readings from IoT sensors, user interactions on websites or applications, or trading activity in financial markets.

Here's the question: What do Tesla Autopilot, Wall Street automated trading algorithms, smart homes, transportation networks that enable intra-day lightning arrivals, and open data released by the New York City Police Department all have in common? The answer is that these applications all rely on a form of data that measures how things change over time, where time is not just a metric, but a master axis of coordinates. To summarize, these data have three characteristics: arrivals are almost always recorded as new entries; the data usually arrive in chronological order; and time is a principal axis (either regular intervals or irregular). This is the data of time series, which gradually play a greater role in our world.

It turns out that using an existing relational database to handle this type of data is not a good way to go, so what is the rationale for using a time-series database?

First, time-series data is very large. Time series data accumulates very quickly (e.g., a connected car can collect 25GB of data per hour), and conventional databases are not designed to handle data of this scale; relational databases handle large data sets very poorly; NoSQL databases can handle scale data well, but are still no match for a database that has been fine-tuned for time series data. In contrast, a time-series database (which can be based on either a relational database or a NoSQL database) treats time as a first-class citizen and handles this kind of large-scale data by improving efficiency and bringing performance improvements, including: higher accommodation rates, faster large-scale queries (although some support more queries than others), and better data compression.

Second, time-series databases are specifically optimized for operations on time-series data. tSDBs also typically include some common features and operations for analyzing time-series data: data retention policies, continuous queries, flexible temporal aggregation, etc. Even if scale is not a concern at the moment (for example, you are just starting to collect data), these features can still provide a better user experience and make your life easier.

There are many companies that have developed excellent timing databases, and we can further understand why we need to use TSDB by learning the advantages of their databases, which, in the case of TDengine, are broadly as follows.

1. High performance: TDengine outperforms other time series databases in data ingestion and querying, while significantly reducing storage and computational costs through its innovative design and dedicated storage engine.
2. Scalability: TDengine provides out-of-the-box scalability and high availability through its natively distributed design. Nodes can be added with simple configuration for greater data processing capacity. In addition, this functionality is open source.
3. SQL support: TDengine uses SQL as the query language to reduce learning and migration costs, while adding SQL extensions to better handle time-series data and supporting easy and flexible schema-free data ingestion.
4. All in One: TDengine has built-in caching, stream processing and data subscription functions, and some scenarios no longer require integration with Kafka/Redis/HBase/Spark, etc. It makes the system architecture simpler and easier to maintain.
5. Seamless integration: TDengine provides seamless integration with third-party tools such as Telegraf, Grafana, EMQX, Prometheus, StatsD, collectd, etc. without a single line of code. More will be integrated.
6. Zero administration: installation and cluster setup can be done in seconds. Data partitioning and sharding is performed automatically. The operational status of TDengine can be monitored through Grafana or other DevOps tools.
7. Zero learning cost: Zero learning cost with SQL as the query language and support for ubiquitous tools such as Python, Java, C/C++, Go, Rust, Node.js connector, etc.
8. Interactive Console: TDengine provides convenient console access to the database to run live queries, maintain the database or manage clusters without any programming.

In a word, most of the data generated in the era of Big Data and IoT have the property of time series, and the common relational database cannot handle this type of data well, so we have developed a new form of data storage database, such as the time-series database, which has proved to be very advantageous in handling the data in the era of Big Data and IoT.