Why do we need a time-series database

In 2017, the time series database suddenly became popular. In February of the same year, Facebook open-sourced the beringei time series database; In April, timescaleDB, a time series database based on PostgreSQL, was also open sourced, and as early as July 2016, Baidu Cloud released TSDB, the first multi-tenant distributed time series database product in China, on its Tiangong Internet of Things platform, which became a core product supporting its development in manufacturing, transportation, energy, smart cities and other industrial fields, and also became a landmark event for Baidu's strategic development of industrial Internet of Things. As a very important service in the direction of the Internet of Things, the frequent voices of the industry show that enterprises cannot wait to embrace the advent of the Internet of Things era.

This article will start from the basic concepts, usage scenarios, and the difference of the time-series database, and finally answer why we need time series databases

**Background**

Baidu unmanned vehicles need to monitor various states when running, including coordinates, speed, direction, temperature, humidity, etc., and need to record the data monitored every moment for big data analysis. Each vehicle collects nearly 8T of data per day. If you just store it and don't query it is fine (although it is already a big cost), but if you need to quickly query the multi-latitude grouping aggregation query such as "What are the unmanned vehicles with a speed of more than 60km/h at two o'clock in the afternoon at Hou Chang Village Road", then the time series database will be a good choice.

**What is a time series database**

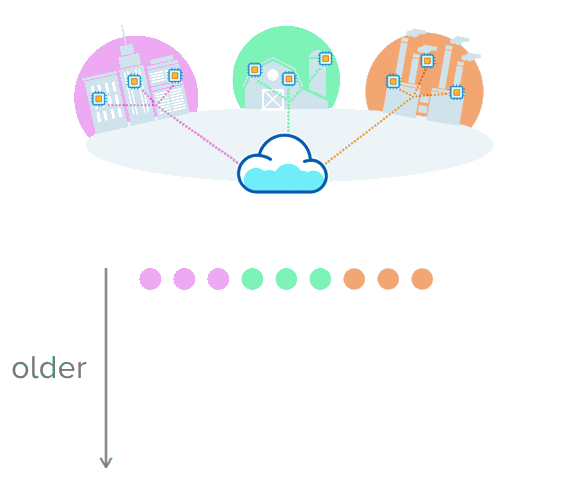
Let's start with what time series data is. Time-series data is a sequence of data points collected over time intervals, giving us the ability to track changes over time. Time-series data can track changes over milliseconds, days, or even years.

Some people think of "time series data" as a series of data points stored in chronological order that measure the same thing over time, which is true, but only describes shallow information.

Others might think of it as a series of numeric values paired with a timestamp defined by a name and a set of collation dimensions (or "labels"). This may be a way to model time series data, but it is not a definition of the data itself.

We continue to go deeper.

Here's a basic example of a sensor that collects data from three environments: cities, farms, and factories. Here, each data source periodically sends new readings, creating a series of measurements collected over time.



There are also many other types of time series data, such as: DevOps monitoring data, mobile/web application event streams, industrial machine data, scientific measurements.

These datasets have three main things in common:

●Arriving data is almost always recorded as a new entry

●Data usually arrives in chronological order

●Time is a primary axis (can be either regular or irregular)

In other words, the processing of time series data is usually accompanied by the arrival of the data. Although incorrect data needs to be corrected after the fact, or delayed or unordered data is processed, these are exceptions and do not fall within the scope of the standard.

**Scenarios for time series databases**

Suppose you are maintaining a Web application, and you can update the user's "last\_login" timestamp in a row of the "users" table each time a user logs on, but what if you treat each sign-in as a separate event and collect that data over time? You can then: Track historical sign-in activity, see how user usage increases or decreases over time, and differentiate users based on how often they visit your app or more metrics.

This example illustrates a key point: by preserving the time series nature inherent in the data, we are able to retain useful information about how the data changes over time.

(In fact, this example illustrates another point: event data is also time series data.)

Of course, storing data in this way poses an obvious problem: eventually you get a lot of data at a fairly fast rate, and time series data will quickly pile up.

Excessive data volumes can cause serious performance problems for logging and query operations. This is also the reason why people are gradually turning to time series databases.

All time series data is generated, and it is necessary to show its historical trends, cycle laws, anomalies, and further prediction and analy.sis of the future, which is a suitable scenario for time series databases.

In the direction of industrial Internet of Things environment monitoring, Baidu Tiangong's customers have encountered such a problem, due to the requirements of the industry, it is necessary to store working conditions data. The customer has 20,000 monitoring points per plant, a collection cycle of 500 milliseconds, and a total of 20 plants. That would result in a staggering 26 trillion data points a year. Assuming 50Bytes per point, the total amount of data will reach 1P (if each server has 10T hard disks, then a total of more than 100 servers are needed). This data is not just to be generated in real time, written to storage; It is also necessary to support fast queries, make visual displays, and help managers analyze decisions; And it can also be used to do big data analysis, find deep-seated problems, help enterprises save energy and reduce emissions, and increase efficiency. The final customer adopted Baidu Tiangong's time series database solution to help him solve the problem.

In the Internet scene, there are also a large amount of time series data generated. Baidu has a large number of services within it that use the time series database of Tiangong IoT Platform. For example, in order to protect the user's user experience, Baidu's internal service records every network lag and network delay of the user to Baidu Tiangong's time series database. Reports are directly generated by the time series database for technical product analysis, and problems are found and solved as soon as possible to ensure the user's experience.

**How is time-series data different?**

You may ask: How is this different than just having a time-field in a dataset? Well, it depends: how does your dataset track changes? By updating the current entry, or by inserting a new one?

When you collect a new reading for sensor\_x, do you overwrite your previous reading, or do you create a brand new reading in a separate row? While both methods will provide the current state of the system,**you can only analyze the changes in state over time if you insert a new reading each time.**

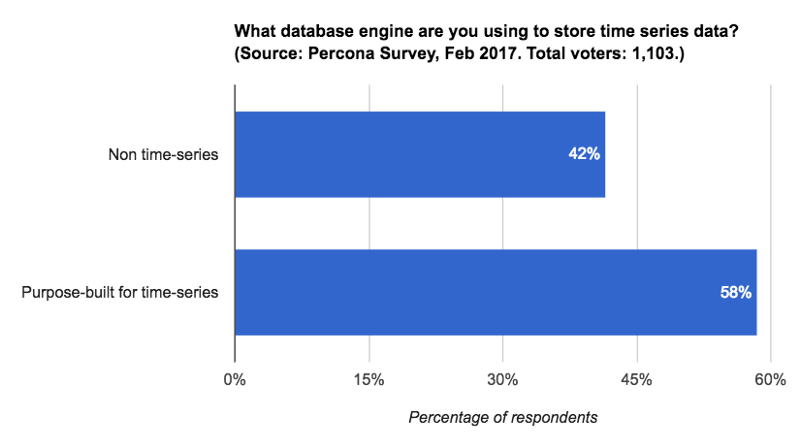
Simply put: time-series datasets track changes to the overall system as INSERTs , not UPDATEs.

This practice of recording each and every change to the system as a new, different row is what makes time-series data so powerful. It allows us to measure and analyze change: what has changed in the past, what is changing in the present, and what can we forecast changes may look like in the future.

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

You might ask: Why can’t I just use a “normal” (i.e., non-time-series) database?

The truth is that you can, and some people do.

 But, there are at least two reasons why TSDBs are the fastest-growing category of databases today: scale and usability.

**Scale:** Time-series data accumulates very quickly, and normal databases are not designed to handle that scale (at least not in an automated way). Traditionally, relational databases fare poorly with very large datasets, while NoSQL databases are better at scale (although a relational database fine-tuned for time-series data can actually perform better, as we’ve shown in benchmarks versus InfluxDB, versus Cassandra, and versus MongoDB). In contrast, time-series databases - whether they’re relational or NoSQL-based - introduce efficiencies that are only possible when you treat time as a first-class citizen. These efficiencies allow them to offer massive scale, from performance improvements, including higher ingest rates and faster queries at scale (although some support more queries than others) to better data compression.

**Usability:** TSDBs also typically include built-in functions and operations common to time-series data analysis, such as data retention policies, continuous queries, flexible time aggregations, etc. Even if you’re just starting to collect this type of data and scale is not a concern at the moment, these features can still provide a better user experience and make data analysis tasks easier. Having built-in functions and features to analyze trends readily available at the data-layer often leads you to discover opportunities you didn’t know existed, no matter how big or small your dataset

This is why developers are increasingly adopting time-series databases and using them for a variety of use cases:

**●**Monitoring software systems: Virtual machines, containers, services, applications

Monitoring physical systems: Equipment, machinery, connected devices, the environment, our homes, our bodies

**●**Asset tracking applications: Vehicles, trucks, physical containers, pallets

**●**Financial trading systems: Classic securities, newer cryptocurrencies

**●**Eventing applications: Tracking user/customer interaction data

**●**Business intelligence tools: Tracking key metrics and the overall health of the business

(and more)

Once you begin to see more of the information your applications store as time-series data, you still have to pick a time-series database that best fits your data model, write/read pattern, and developer skillsets. Although NoSQL time-series database options have prevailed for the past decade as the storage medium of choice, more and more developers are seeing the downside to storing time-series data separately from business data (most time-series databases don’t provide good support for relational data). In fact, this poor developer experience was one of the driving factors in why we started Timescale. Keeping all of your data in one system can drastically reduce application development time – and the speed at which you can make key decisions.

Nowhere is this more evident than with the rise of numerous self-service business intelligence tools like Tableau, Power BI, and yes, even Excel. When precious time-series data is kept separate from business data, users struggle to make timely, business-critical observations. Instead, users find that they need to rely on these third-party tools to mash up data into something meaningful. There are many valid and good reasons to use these powerful tools, but being able to quickly query your time-series data alongside meaningful metadata information shouldn’t be one of them. SQL has been built and honed over decades to provide efficient ways of generating these valuable aggregations and analyses.