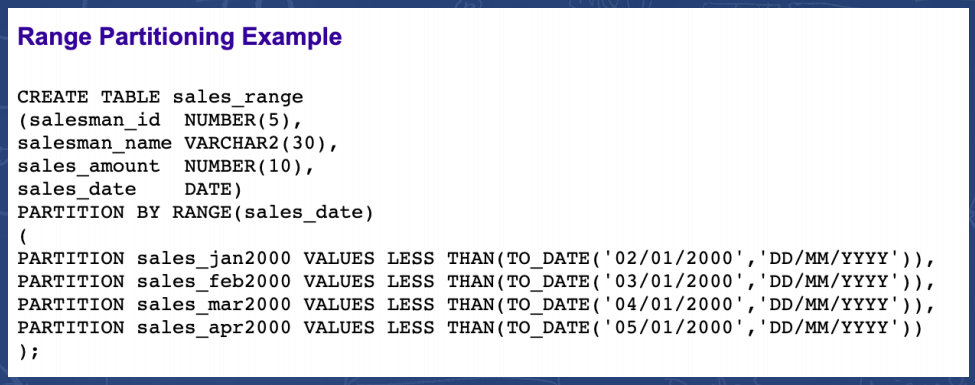
Why do we need a time-serise database?

With the development and progress of science and technology, we have generated new needs for previous systems, to do data visualization, information management, and process control. Now the business is not only satisfied with this simple management and control. Data visualization analysis, big data information mining, statistical prediction, modeling and simulation, and intelligent control have become the pursuit of various businesses. We used to use the Internet to solve real problems. Now we are not satisfied with the reality, the data will be connected into a time series, you can view its history, reveal its regularity, in the future can grasp its trend, predict its trend. Therefore, we began to store a large number of time-related data (such as logs, user behavior, etc.), and summarized the structural characteristics and common usage scenarios of these data, constantly improved and optimized, and created a new type of database classification —— time series database.

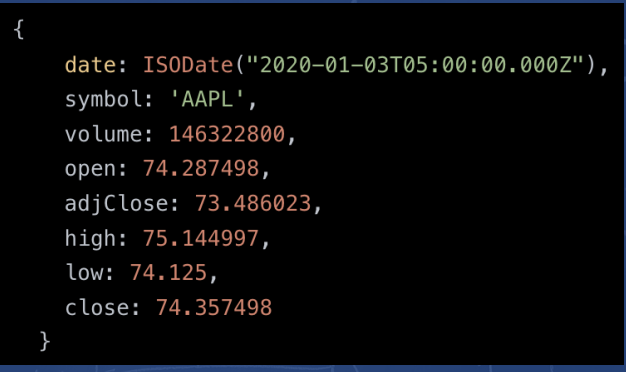
First of all, it needs to be clear what is a time series database, and what is the timing data that it manages.Timing data is a series of data generated over time, namely time series data, recorded and indexed in the order of time dimensions. Simply put, it is the data with time stamp data. A timing database is a database optimized for the uptake, processing, and storage of timestamp data, i. e., a database used specifically for processing timing data.

Timing data has some characteristics: 1. With timestamped data 2. structured data 3. data source like a stream 4. data rate is quite stable 5. 6. write more than read 7. data rarely deleted or updated 8. There are retention policy 9. real-time data calculation is ideal, query is always in time and space range, the most significant difference from other data is, timing data is usually written in chronological order, data state changes with time. The main data properties of the timing data are as follows: Each data point contains timestamps for indexing, aggregation, and sampling. This data can also be multidimensional and related; writing more and read less, requiring supporting second and millisecond or even nanosecond-level high-frequency writing; queries are usually multidimensional aggregated queries requiring higher latency; and summary views of data (e. g., subsampled or aggregated views, trend lines) may provide more insight than a single data point. For example, given network unreliability or abnormal sensor readings, we may set an alert when an average exceeds a threshold, rather than at a single data point; analyzing data often requires access over time (e. g., give me click-through data for the past week);

Why do we need timing databases? At present, most of the databases on the market are relational databases, most of which can not handle well and change quickly, and the amount of data greatly the time sequence data. Most current temporal databases have abandoned the trap of traditional relational databases and adopted models commonly referred to as NoSQL models. A recent survey showed that developers prefer NoSQL to relational databases for timing data, at more than 2:1. In traditional relational databases, the main way to processing temporal data is partitioning. As shown in Figure 1, a range partition maps data to a partition based on the range of partition key values you establish for each partition. It is the most common type of partition and is often used with dates. For example, you may want to divide sales data into once a month partition, usually use VALUES LESS THAN command, using this way for timing data management is the advantage of relational database development is more mature, team members are more familiar with it, but the disadvantages of this treatment is also obvious, with the increase of data, processing becomes more complex, reading, writing and management are more difficult. NoSQL can also be used to solve the processing of timing data. In Figure 2, take MongoDB as an example, with the same advantages as using a relational database to process timing data, but there are also some disadvantages, such as complex processing, and the disadvantages of significantly increasing the storage cost due to retaining redundant copies of the data.



(1)



(2)

For MySQL in massive timing data scenarios: large storage cost: poor timing data compression, need to occupy a large amount of machine resources; high maintenance cost: PC system, requires upper manual library table, high maintenance cost; low write throughput: low single write throughput, difficult to meet the write pressure of timing data; poor query performance: suitable for transaction processing, poor aggregation and analysis performance of massive data. In addition, using Hadoop ecology (Hadoop, Spark, etc.) to store timing data will have the following problems: high data latency: offline batch processing system, data from generation to analysis, time consuming hours, even day; query performance is poor: not very good use of the index, rely on MapReduce tasks, query time is generally in the minute level.

You can see that the timing database needs to solve the following problems: The writing of the timing data: How to support the writing of tens of millions of millions of data points per second. Reading of temporal data: How to support the grouping and aggregation operation of hundreds of millions of data at the second level.Cost-sensitive: massive data storage brings cost issues. How to store these data at a lower cost will become the top priority of the timing database.

According to the characteristics of timing data can be made the following improvements: the first feature, the amount of data is large, the same dimension of repeated value, we can compress these same dimensions of storage (because it is repetitive), reduce the storage cost, such as repeated host and port only one storage. The second feature, high concurrency writing, and like hbase, we can adopt LSM instead of the B tree. The third feature is aggregation and aggregation of cold and hot data, which can reduce the precision storage of cold data, that is, aggregate historical data to save storage space.

So what are the essential differences between time sequence database and traditional big data storage solutions?The author thinks that the most important difference is the structured data.1. Stores structured data. We all know that the traditional big data scheme to store data contains structured, semi-structured, unstructured data, so that we can not decide which fields and define each field data type, like hbase is stored through byte type, that is to put into hbase data is byte array, from ordinary type to byte array needs to do ourselves, we do not know how to convert to byte, it will be more efficient storage. However, the data generated by the timing data is all structured data. We can define the fields and types of the data in advance, so that the database system can choose the optimal compression method according to different field types, and greatly improve the utilization rate of storage.2. Analyze and aggregate the structured data. Since analytic aggregation is structured data, So we don't need to use complex computing tools like mapreduce, and we generally don't need hive data warehouses, And just similar to sum, avg at the database storage level, You can even do some simple flow calculations, Provides the foundation for 'hyperfusion' (hyperfusion means integrating multiple components similar to previous big data processing solutions into one component, Mainly because the structured data is too simple, The collection and calculation are all relatively simple, This is also the development trend of the subsequent timing database, Reduce the system complexity).

Timing data accumulates very fast, and regular databases are designed to handle this scale (or at least not in an automated way). Relational databases perform poorly on very large datasets, while NoSQL databases are better in scale (although relational databases that are fine-tuned for timing data can actually perform better, as we show on benchmarks against InfluxDB, Cassandra, and MongoDB). In contrast, the benefits of introducing timing databases (whether relational, or NoSQL-based databases) can only be achieved if you consider time as your first consideration. These benefits enable them to provide large-scale performance improvements, including higher throughput and faster large-scale queries, and better data compression.

Compared with traditional databases, the timing database also has great advantages in processing large-scale data, but also has great advantages in terms of availability. Usually, it also includes the built-in functions and operations commonly used for temporal data analysis, such as data retention strategies, continuous queries, flexible time aggregation, etc. Even if you are just starting to collect this type of data and do not need to consider size right now, these features can still provide a better user experience, making data analysis tasks easier. Using built-in functions and features to analyze trends readily available in the data layer often finds unexpected value, regardless of whether the dataset is large or small.

Timing databases are indeed specialized in iot / monitoring, and have huge advantages in timing data writing / query / data compression to address many user pain points. The existing timing databases are still inadequate in storage, either stand-alone or difficult to maintain (opentsdb). There are many more places to be modified. But higher query performance, faster writing speed, more convenient and low-cost operation and maintenance, everyone want. Once the scale of the business increases, all needs should be considered, but they cannot be met. Doing engineering is essentially still constantly doing Trade Off. How to choose or in the actual production application to choose.