1 time series database

1.1 Time Series

In mathematics, a time series is a series of data points indexed (or listed or plotted) in chronological order.

Most commonly, a time series is a sequence taken at successive equally spaced time points.

Therefore, it is a series of discrete time data.

Time series are used in statistics, signal processing, pattern recognition, econometrics, mathematical finance, weather forecasting, earthquake prediction, electroencephalography, control engineering, astronomy, communication engineering, and mainly in any applied science and engineering field involving time measurement.

1.2 Time series analysis

Time series analysis includes methods used to analyze time series data to extract meaningful statistics and other features of the data.

Time series prediction is the use of models to predict future values based on previously observed values.

Although regression analysis is commonly used to test the relationship between one or more different time series, this type of analysis is not usually referred to as "time series analysis", it specifically refers to the series of relationships between different time points within a single time point.

Interrupted time series analysis is used to detect changes in time series evolution before and after intervention that may affect underlying variables.

Time series analysis can be applied to real value, continuous data, discrete numerical data or discrete symbolic data.

In the context of statistics, econometrics, quantitative finance, seismology, meteorology, and geophysics, the primary goal of time series analysis is prediction.

In the context of signal processing, control engineering and communication engineering, it is used for signal detection.

Other applications include data mining, pattern recognition and machine learning, where time series analysis can be used for clustering, classification, query by content, anomaly detection and prediction.

1.3 Time series database

A Time Series database (TSDB) is a software system optimized to store and service time series through associated time and value pairs.

In some fields, time series may be referred to as profiles, curves, tracks, or trends. Some early time series databases were associated with industrial applications. These databases could efficiently store measurements from sensing devices (also known as data history databases), but are now used to support a wider range of applications.

In many cases, repositories of time series data will utilize compression algorithms to efficiently manage the data.

Although time series data can be stored in many different database types, the design of these time-critical indexed systems is markedly different from that of relational databases that reduce discrete relationships through reference models.

Compared to other datasets, time series datasets are relatively large and uniform -- typically consisting of timestamps and related data.

Time series datasets can also have fewer relationships between data entries in different tables and do not need to store entries indefinitely.

The unique properties of time series datasets mean that time series databases can provide significant improvements in storage space and performance over general purpose databases.

For example, because of the uniformity of time series data, specialized compression algorithms can provide improvements over conventional compression algorithms designed to deal with less uniform data.

Time series databases can also be configured to periodically delete old data, unlike regular databases that are designed to store data indefinitely.

Special database indexes can also improve query performance.

2NoSQL

2.1 NoSQL

NoSQL databases provide a mechanism for storing and retrieving data that is modeled in ways other than the table relationships used in relational databases.

NoSQL databases are increasingly used for big data and real-time Web applications.

NoSQL systems are also sometimes called NotonlySQL to emphasize that they may support SQL-like query languages or juxtapose with SQL databases in a multilanguage persistence schema.

NoSQL databases use data structures (such as key-value pairs, wide columns, graphs, or documents) that are different from those used by default in relational databases, which makes some operations in NoSQL faster.

Many NoSQL stores sacrifice consistency for availability, partition fault tolerance, and speed.

Barriers to more NoSQL storage include the use of low-level query languages, lack of ability to perform AD hoc joins across tables, lack of standardized interfaces, and previous heavy investments in existing relational databases.

Most NoSQL stores lack true ACID transactions, although a few databases have made them central to their design.

Most NoSQL databases offer the concept of "final consistency", where database changes propagate "eventually" (usually within milliseconds) to all nodes, so data queries may not return updated data immediately, or may result in data being read that is inaccurate, known as a stale read problem.

In addition, some NoSQL systems may experience write losses and other forms of data loss.

Some NoSQL systems provide concepts such as write-ahead logging to avoid data loss.

For distributed transaction processing across multiple databases, data consistency is a greater challenge and is difficult for both NoSQL and relational databases.

2.2 Characteristics of non-relational database

1)easy to scale: There are many types of NoSQL databases, but a common feature is to remove the relational characteristics of relational databases.

There is no relationship between the data, which makes it easy to scale.

Intangibles bring scalable capabilities at the architectural level.

2)large amount of data, high performance: NoSQL databases have very high read and write performance, especially in the large amount of data, also excellent performance.

This benefits from its irrelevance and simple database structure.

Generally, MySQL uses QueryCache.

NoSQL's Cache is record level, which is a fine-grained Cache, so NoSQL performs much better at this level.

3)flexible data model: NoSQL does not need to set up fields for the data to be stored, and can store custom data formats at any time.

4)High availability: NoSQL can easily implement a highly available architecture without much impact on performance.

For example, the Cassandra and HBase models can also be replicated to achieve high availability.

2.3 Comparison between relational and non-relational databases

1)Cost :Nosql databases are easy to deploy, are basically open source software, don't cost as much as Oracle, and are cheaper than relational databases.

2)Query speed :Nosql databases store data in a cache and do not need to analyze the SQL layer.

Relational databases store data on hard disks and are naturally much slower than Nosql databases.

3)Format of data storage: Nosql stores data in key,value, document, picture, etc., so it can store basic types, objects or collections and other formats, while databases only support basic types.

4)Scalability: The limitations of multi-table query mechanisms such as JOIN in relational databases make scaling difficult.

Nosql is based on key-value pairs and there is no coupling between data, so it is easy to scale horizontally.

5)Persistent storage: Nosql is not used for persistent storage, massive data persistent storage, still need a relational database

6)Data consistency: Non-relational databases usually emphasize the final consistency of data, rather than the strong consistency of data as relational databases do, and data read from non-relational databases may still be in an intermediate state, Nosql does not provide for transaction processing.

3. Selection and future direction of time series database

3.1 Introduction to the Application scenarios of the Sequential Database

Timing database is widely used in scenarios such as Internet of Things, Internet of vehicles, industrial Internet and smart city to realize data collection, storage, calculation and application of various devices.

3.2 Typical sequential database

PISystem: It is an industrial real-time data operation and management system developed by OSISoft. It is widely used in the process industry, such as petroleum, petrochemical, power, etc.

PSsystem includes DataArchive, AssetFramework and other different components, in which DataArchive realizes the collection, storage and management of temporal data, and AssetFramework realizes the data capitalization by combining temporal data with relational data.

It comes with an SQLServer relational database to store and process relational data, which gives meaning to sequential data.

OpenTSDB optimizes time series data based on the HBase storage model. The storage model is N-ARY storage. The format of each row is Metrics, Tags, Timestamp: datA1: datA2:...

.

To improve storage efficiency, OpenTSDB stores one hour of data per row by default and supports encoding, compression, and parallel query execution.

Essentially, OpenTSDB uses HBase as KV storage.

InfluxDB is an open-source temporal database created by InfluxData in 2013.

The core concept of InfluxDB is a Series, marked by measurement+tags.

It uses column storage and divides data into blocks according to time, and supports fast deletion of block data.

The InfluxDB data model consists of data points, each of which has four parts: Measurement, tag set (lexicalized KV pairs), field set (values of data points, types supporting float, int, string, and Boolean values), and timestamp.

The underlying storage uses time-StructuredMergeTree (TSM), where each TSM file contains compressed and sorted sequential data.

Druid uses wide tables instead of joins, which is why its completely flat schema provides significant performance gains, as well as low disk space overhead due to dictionary encoding + compression.

Druid data is stored in datasources, similar to tables in a relational database.

Each datasource is partitioned by time, and the data in each time segment is called chunk.

Within the chunk, the data is further partitioned into segments.

A segment file uses a column storage model and contains three types of fields: timestamp, dimension, and metric.

This structure enables fast aggregation and multidimensional queries.

TimescaleDB is an open source relational timing database based on PostgreSQL. It uses hackPostgreSQL storage engine to support timing data scenarios.

MatrixDB is a postgresQl-based hyper-converged sequential database that implements a new storage engine within relational databases and is optimized for high-speed data inserts and diverse queries in sequential data scenarios.

MatrixDB, founded in 2020, is the only product that has passed the "Distributed Analytical Database Capability Evaluation" and "Timing Database Capability Evaluation" issued by the Ministry of Industry and Information Technology.

3.3 Selecting a sequential database

In the selection, we can consider the following selection factors:

Matching its own business scenarios: Timing business can be subdivided into multiple scenarios according to the scale of equipment, number of indicators and collection frequency.

At present, most time sequence databases can be applied to small and medium scale scenarios, but there are great challenges for large-scale scenarios, such as tens of millions of devices, hundreds of indicators in a single device, and second-level acquisition.

Graphical tools: Are there easy-to-use graphical tools, including graphical access tools and graphical monitoring o&M management tools?

Graphic tools can greatly reduce the threshold of use, improve the efficiency of development operation and maintenance.

Ecology and community: The database mainly solves the problem of data storage and calculation, and the end-to-end solution of business also needs to rely on the perfection of ecology.

In addition, database is complex software, especially distributed database, development operation and maintenance have a certain threshold, so community activity is an important factor to consider.

An active community can help us find solutions faster when we encounter problems.

Write performance, out-of-order write: More than 95% of operations in a sequential database are data writes and require smooth performance, so write performance is an important consideration.

In addition, whether to support out-of-order writing is also a factor in selection, because data errors often occur in sequential data and retransmission.

Update and Delete: In many service scenarios, recent data is often out of order or incorrect data. In this case, you need to update or delete the data.

Downsampling: The value density of time series data decays over time, so downsampling is often required for the collected index data. For example, the original data is collected in 10 seconds, and the data will also be downsampled to a point in an hour or even a point in a day.

In this scenario, you need to consider whether the database automatically supports downsampling.

Compression ratio: When the device has a large amount of data, a large number of indicators or a high collection frequency, the amount of time series data will become very large, requiring high storage space. The common method is to keep only recent data and delete historical data.

But the business wants as much data as possible to analyze and model as much data as possible, so compression ratio becomes an important metric.

In general, sequential databases support column encoding compression and block compression.

Query performance: sequence scene query is very diversified, including both simple index query and analytical query;

Both single device/multi-device latest value, clustered value class query, and multidimensional query;

Not only interpolation, but also threshold calculation, pattern recognition and other queries.

It is a good idea to actually measure the database based on the business scenario.

In addition, more common benchmarks such as TSBS benchmarks can also be used to comprehensively verify the query capability of the database.

Hot and cold tiered storage: The value density of sequential data decays over time, so it is a common requirement to use storage media and services at different prices for data in different time periods.

Hot and cold storage tiering is a good solution to this problem and is often used in conjunction with partitioning.

Partitioning support: Sequential scenarios are usually partitioned according to the time attribute, sometimes with secondary partitioning based on other attributes to better support inserts and queries.

Continuous aggregation: Sequential scenarios frequently use time window queries and frequently obtain aggregate values for data points within that time window, such as maximum CPU utilization in the last 10 seconds.

The traditional method is to send a query request to the database every 10 seconds. After the database receives the query, it calculates the maximum CPU indicator in the past 10 seconds. This method is feasible, but it costs a lot and has a high delay.

Continuous aggregation can continuously calculate the index aggregation value of the 10-second window, so that the result can be returned directly when the corresponding query is received.

A tie-in subscription mechanism further avoids polling queries and sends results directly to interested subscribers.

Timing functions: There are many specific functions in timing scenes, such as first, last, Gapfill, variance, standard deviation, ARIMA, etc. Whether these commonly used functions are supported natively is also a factor for selection consideration.

Cloud one: because of the large amount of time-series data, high frequency, thus often use edge computing architecture, on the edge side small cluster deployment of single node temporal database or database, at the same time, the edge to the data through the preliminary processing of backwardness to the cloud/data center of large database cluster, this product can support the cloud side will need to consider.

Safety mechanism: Safety mechanism is a factor often ignored in selection, because at the beginning, it is mainly to run through the business, so it pays little attention to safety.

However, many temporal scenarios, such as energy, electricity, industrial Internet, etc., attach great importance to security.

Whether there is perfect security control, including authentication, access control, encryption and audit, is an important factor to judge whether a sequential database is secure or not.

Embedded scripting capability: In addition to standard SQL, application developers often perform more complex data processing. One way is to use JDBC to read data into memory and use the data processing functions provided by the programming language to process the data. This method is suitable for small data scenarios.

If there is a large amount of data, it will be inefficient to read the data into memory for transformation.

Therefore, the ability to process data in the database using common programming languages (Python, R, etc.) is an important consideration.

Operation and maintenance management: This consideration includes installation and deployment, monitoring, alarm, fault recovery, backup and restoration, capacity expansion, and upgrade.

3.4 Future direction of sequential database

Whatever the future of dedicated sequential databases, sequential databases (let's continue to call them sequential databases) will continue to evolve and will become increasingly important as the Internet of Things, the Internet of vehicles, the industrial Internet, and smart cities evolve.

There are several directions that deserve our attention.

1)Hyperfusion timing sequence database

Fusion is one of the main themes of database development in the next few years. The boundary of database is becoming more and more fuzzy. Like the evolution from simple to complex in the biological world, database will appear a "new species" with more complex organization and more powerful function but simpler use: hyper-fusion database.

With the rapid development of the Internet, the amount of data increases at a super fast speed every year, but the speed of database technology iteration has not caught up with the growth of data.

In order to solve the performance problem of application processing massive data, various special database applications were born.

These special time sequence databases solve the pain points of the current business with excellent performance and scalability, but also bring the problem of data isolation, and the coupling of data processing logic and business logic.

In order to solve the problem of data silos and tight coupling of data processing logic/business processing logic, query engines like Presto emerged, which in turn led to the emergence of data middleware.

However, because query engines such as Presto do not manage the data themselves, query performance is poor and features such as ACID are not supported.

In addition, the technology stack is complex and the development operation and maintenance efficiency is low.

So rather than on multiple independent database encapsulates a query engine, might as well make the storage engine into the relational database, through the pluggable storage engine, support for multiple storage engines in a relational database, combining calculation engine, can be implemented in a database to support various data types and various business scenarios, this is super database.

Under the trend of hyperfusion database, hyperfusion timing database is an important development direction of timing database.

Because the implementation difficulty of hyperfusion timing sequence database is lower than that of general hyperfusion database, the hyperfusion timing sequence database first appeared and realized the product.

2) Cloud native timing database

Cloud native database is an important innovation of business model, which is exerting profound influence on database technology.

Foreign cloud native databases such as Snowflake have achieved great commercial success and played an important role in promoting database technology.

In such a big situation, how to implement cloud native timing database is an important research direction.

There are many differences between cloud native timing databases and current cloud native data warehouses such as Snowflake. For example, while a data warehouse is mainly about bulk loading and OLAP type queries, a timing database needs to support frequent high-throughput data writes, out-of-order data writes, updates and deletions, high-concurrency timing queries, and continuous clustering queries.

The specific issues of these timing scenarios need to be considered when designing and implementing cloud native timing databases.

3) Intelligent database

Database operation and maintenance management is a very challenging task. As the database cluster becomes larger, software and hardware failures will become normal, which will further increase the difficulty of distributed database operation and maintenance.

In this case, intelligent operation and maintenance is becoming a hot topic.

By collecting various index data during the operation of the database, the sequential database can be used to analyze the sequential database itself to improve the intelligence of the database and reduce the complexity of operation and maintenance.