Apache IoTDB User Guide (V0.10.x)

Apache IoTDB (Database for Internet of Things) is an IoT native database wit h high performance for data management and analysis, deployable on the edge and the cloud. Due to its light-weight architecture, high performance and rich feature…

目　录

- 致谢
- Get Started
  - QuickStart
  - Frequently asked questions
  - Research Papers
- Overview
  - What is IoTDB
  - Architecture
  - Scenario
  - Features
- Concept
  - Data Model and Terminology
  - Data Type
  - Encoding
  - Compression
- Server
  - Download
  - Single Node Setup
  - Cluster Setup
  - Config Manual
  - Docker Image
- Client
  - Command Line Interface
  - Native API
  - JDBC
  - Other Languages
  - TsFile API
  - MQTT
  - Status Codes
- Operation Manual
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
  - Administration
  - SQL Reference
- System Tools
  - Sync Tool

JMX Tool
Watermark Tool
Query History Visualization Tool
Monitor and Log Tools
Load External Tsfile
Performance Tracing Tool
Ecosystem Integration
Grafana
MapReduce TsFile
Spark TsFile
Spark IoTDB
Hive TsFile
Architecture
Files
Writing Data on HDFS
Shared Nothing Cluster
Comparison with TSDBs
Comparison

致谢

当前文档 《Apache IoTDB User Guide (V0.10.x)》 由 进击的皇虫 使用 书栈网(BookStack.CN) 进行构建，生成于 2020-12-19。

书栈网仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈网难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到书栈网，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到书栈网获取最新的文档，以跟上知识更新换代的步伐。

内容来源：Apache
https://iotdb.apache.org/UserGuide/V0.11.x/Get%20Started/QuickStart.html

文档地址：http://www.bookstack.cn/books/iotdb-0.11.x-en

书栈官网：https://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

- QuickStart
- Frequently asked questions
- Research Papers

Quick Start

Outline

- Quick Start
- Prerequisites
- Installation
    - Build from source
        - Configurations
- Start
    - Start IoTDB
    - Use IoTDB
        - Use Cli
        - Basic commands for IoTDB
    - Stop IoTDB
- Only build server
- Only build cli

This short guide will walk you through the basic process of using IoTDB. For a more-complete guide, please visit our website's User Guide.

Prerequisites

To use IoTDB, you need to have:

1. Java >= 1.8 (Please make sure the environment path has been set)
2. Set the max open files num as 65535 to avoid "too many open files" problem.

Installation

IoTDB provides you three installation methods, you can refer to the following suggestions, choose one of them:

- Installation from source code. If you need to modify the code yourself, you can use this method.
- Installation from binary files. Download the binary files from the official website. This is the recommended method, in which you will get a binary released package which is out-of-the-box.(Coming Soon…)
- Using Docker: The path to the dockerfile is https://github.com/apache/iotdb/blob/master/docker/src/main

Here in the Quick Start, we give a brief introduction to install IoTDB. For further information, please refer to Chapter 3 of the User Guide.

Download

You can download the binary file from: Here

Configurations

configuration files are under “conf” folder

- environment config module (`iotdb-env.bat` , `iotdb-env.sh`),
- system config module (`iotdb-engine.properties`)
- log config module (`logback.xml`).

For more, see Chapter3: Server in detail.

Start

You can go through the following step to test the installation, if there is no error after execution, the installation is completed.

Start IoTDB

Users can start IoTDB by the start-server script under the sbin folder.

```
# Unix/OS X
> nohup sbin/start-server.sh >/dev/null 2>&1 &
or
> nohup sbin/start-server.sh -c <conf_path> -rpc_port <rpc_port> >/dev/null 2>&1 &

# Windows
> sbin\start-server.bat -c <conf_path> -rpc_port <rpc_port>
```

- “-c” and “-rpc_port” are optional.
- option “-c” specifies the system configuration file directory.
- option “-rpc_port” specifies the rpc port.
- if both option specified, the *rpc_port* will overrides the rpc_port in *conf_path*.

if you want to use JMX to connect IOTDB, you may need to add

```
-Dcom.sun.management.jmxremote.rmi.port=PORT -Djava.rmi.server.hostname=IP
```

to $IOTDB_JMX_OPTS in iotdb-env.sh. or iotdb-env.bat

Use IoTDB

Use Cli

IoTDB offers different ways to interact with server, here we introduce basic steps of using Cli tool to insert and query data.

After installing IoTDB, there is a default user ‘root’, its default password is also ‘root’. Users can use this default user to login Cli to use IoTDB. The startup script of Cli is the start-cli script in the folder sbin. When executing the script, user should assign IP, PORT, USER_NAME and PASSWORD. The default parameters are “-h 127.0.0.1 -p 6667 -u root -pw -root”.

Here is the command for starting the Cli:

```
# Unix/OS X
> sbin/start-cli.sh -h 127.0.0.1 -p 6667 -u root -pw root

# Windows
> sbin\start-cli.bat -h 127.0.0.1 -p 6667 -u root -pw root
```

The command line client is interactive so if everything is ready you should see the welcome logo and statements:

```
 _____       _________  ______   ______
|_   _|     |  _   _  ||_   _ `.|_   _ \
  | |   .--.|_/ | | \_|  | | `. \ | |_) |
  | | / .'`\ \  | |      | |  | | |  __'.
 _| |_| \__. | _| |_    _| |_.' /_| |__) |
|_____|'.__.' |_____|  |______.'|_______/  version x.x.x


IoTDB> login successfully
IoTDB>
```

Basic commands for IoTDB

Now, let us introduce the way of creating timeseries, inserting data and querying data.

The data in IoTDB is organized as timeseries, in each timeseries there are some data-time pairs, and every timeseries is owned by a storage group. Before defining a timeseries, we should define a storage group using SET STORAGE GROUP, and here is an example:

```
IoTDB> SET STORAGE GROUP TO root.ln
```

We can also use SHOW STORAGE GROUP to check created storage group:

```
IoTDB> SHOW STORAGE GROUP
+-----------------------------------+
|                      Storage Group|
```

```
+---------------------------------+
|                         root.ln|
+---------------------------------+
storage group number = 1
```

After the storage group is set, we can use CREATE TIMESERIES to create new timeseries. When we create a timeseries, we should define its data type and the encoding scheme. We create two timeseries as follow:

```
IoTDB> CREATE TIMESERIES root.ln.wf01.wt01.status WITH DATATYPE=BOOLEAN, ENCODING=PLAIN
IoTDB> CREATE TIMESERIES root.ln.wf01.wt01.temperature WITH DATATYPE=FLOAT, ENCODING=RLE
```

To query the specific timeseries, use SHOW TIMESERIES <Path>. <Path> represents the path of the timeseries. Its default value is null, which means querying all the timeseries in the system(the same as using “SHOW TIMESERIES root”). Here are the examples:

1. Query all timeseries in the system:

```
IoTDB> SHOW TIMESERIES
+-------------------------------+---------------+--------+--------+
|                     Timeseries|  Storage Group|DataType|Encoding|
+-------------------------------+---------------+--------+--------+
|       root.ln.wf01.wt01.status|        root.ln| BOOLEAN|   PLAIN|
|  root.ln.wf01.wt01.temperature|        root.ln|   FLOAT|     RLE|
+-------------------------------+---------------+--------+--------+
Total timeseries number = 2
```

1. Query a specific timeseries(root.ln.wf01.wt01.status):

```
IoTDB> SHOW TIMESERIES root.ln.wf01.wt01.status
+-------------------------------+--------------+--------+--------+
|                     Timeseries| Storage Group|DataType|Encoding|
+-------------------------------+--------------+--------+--------+
|      root.ln.wf01.wt01.status|       root.ln| BOOLEAN|   PLAIN|
+-------------------------------+--------------+--------+--------+
Total timeseries number = 1
```

Insert timeseries data is the basic operation of IoTDB, you can use ‘INSERT’ command to finish this. Before insert you should assign the timestamp and the suffix path name:

```
IoTDB> INSERT INTO root.ln.wf01.wt01(timestamp,status) values(100,true);
IoTDB> INSERT INTO root.ln.wf01.wt01(timestamp,status,temperature) values(200,false,20.71)
```

The data we’ve just inserted displays like this:

```
IoTDB> SELECT status FROM root.ln.wf01.wt01
```

```
+-----------------------+-----------------------+
|                   Time|root.ln.wf01.wt01.status|
+-----------------------+-----------------------+
|1970-01-01T08:00:00.100|                   true|
|1970-01-01T08:00:00.200|                  false|
+-----------------------+-----------------------+
Total line number = 2
```

We can also query several timeseries data at once like this:

```
IoTDB> SELECT * FROM root.ln.wf01.wt01
+-----------------------+-------------------------+-----------------------------+
|                   Time|  root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------+-------------------------+-----------------------------+
|1970-01-01T08:00:00.100|                     true|                         null|
|1970-01-01T08:00:00.200|                    false|                        20.71|
+-----------------------+-------------------------+-----------------------------+
Total line number = 2
```

The commands to exit the Cli are:

```
IoTDB> quit
or
IoTDB> exit
```

For more on what commands are supported by IoTDB SQL, see SQL Reference.

Stop IoTDB

The server can be stopped with ctrl-C or the following script:

```
# Unix/OS X
> sbin/stop-server.sh

# Windows
> sbin\stop-server.bat
```

Only build cli

Under the root path of iotdb:

```
> mvn clean package -pl cli -am -DskipTests
```

After build, the IoTDB cli will be in the folder “cli/target/iotdb-cli-{project.version}”.

Frequently Asked Questions

Outline

- Frequently Asked Questions
  - How can I identify my version of IoTDB?
  - Where can I find IoTDB logs?
  - Where can I find IoTDB data files?
  - How do I know how many time series are stored in IoTDB?
  - Can I use Hadoop and Spark to read TsFile in IoTDB?
  - How does IoTDB handle duplicate points?
  - How can I tell what type of the specific timeseries?
  - How can I change IoTDB’s Cli time display format?

How can I identify my version of IoTDB?

There are several ways to identify the version of IoTDB that you are using:

- Launch IoTDB’s Command Line Interface:

```
> ./start-cli.sh -p 6667 -pw root -u root -h localhost
 _____       _________  ______   ______
|_   _|     |  _   _  ||_   _ `.|_   _ \
  | |   .--.|_/ | | \_|  | | `. \ | |_) |
  | | / .'`\ \  | |      | |  | | |  __'.
 _| |_| \__. | _| |_    _| |_.' /_| |__) |
|_____|'.__.' |_____|  |_____.'|_______/  version x.x.x
```

- Check pom.xml file:

```
<version>x.x.x</version>
```

- Use JDBC API:

```
String iotdbVersion = tsfileDatabaseMetadata.getDatabaseProductVersion();
```

- Use Command Line Interface:

```
IoTDB> show version
show version
+---------------+
|version        |
+---------------+
|x.x.x          |
```

```
+---------------+
Total line number = 1
It costs 0.241s
```

Where can I find IoTDB logs?

Suppose your root directory is:

```
$ pwd
/workspace/iotdb

$ ls -l
server/
cli/
pom.xml
Readme.md
...
```

Let `$IOTDB_HOME = /workspace/iotdb/server/target/iotdb-server-{project.version}`

Let `$IOTDB_CLI_HOME = /workspace/iotdb/cli/target/iotdb-cli-{project.version}`

By default settings, the logs are stored under `IOTDB_HOME/logs` . You can change log level and storage path by configuring `logback.xml` under `IOTDB_HOME/conf` .

Where can I find IoTDB data files?

By default settings, the data files (including tsfile, metadata, and WAL files) are stored under `IOTDB_HOME/data` .

How do I know how many time series are stored in IoTDB?

Use IoTDB’s Command Line Interface:

```
IoTDB> show timeseries root
```

In the result, there is a statement shows `Total timeseries number` , this number is the timeseries number in IoTDB.

In the current version, IoTDB supports querying the number of time series. Use IoTDB’s Command Line Interface:

```
IoTDB> count timeseries root
```

If you are using Linux, you can use the following shell command:

```
> grep "0,root" $IOTDB_HOME/data/system/schema/mlog.txt |  wc -l
>   6
```

Can I use Hadoop and Spark to read TsFile in IoTDB?

Yes. IoTDB has intense integration with Open Source Ecosystem. IoTDB supports Hadoop (opens new window), Spark (opens new window) and Grafana (opens new window) visualization tool.

How does IoTDB handle duplicate points?

A data point is uniquely identified by a full time series path (e.g. `root.vehicle.d0.s0`) and timestamp. If you submit a new point with the same path and timestamp as an existing point, IoTDB updates the value of this point instead of inserting a new point.

How can I tell what type of the specific timeseries?

Use `SHOW TIMESERIES <timeseries path>` SQL in IoTDB's Command Line Interface:

For example, if you want to know the type of all timeseries, the <timeseries path> should be `root`. The statement will be:

```
IoTDB> show timeseries root
```

If you want to query specific sensor, you can replace the <timeseries path> with the sensor name. For example:

```
IoTDB> show timeseries root.fit.d1.s1
```

Otherwise, you can also use wildcard in timeseries path:

```
IoTDB> show timeseries root.fit.d1.*
```

How can I change IoTDB's Cli time display format?

The default IoTDB’s Cli time display format is readable (e.g. `1970-01-01T08:00:00.001`), if you want to display time in timestamp type or other readable format, add parameter `-disableISO8601` in start command:

```
> $IOTDB_CLI_HOME/sbin/start-cli.sh -h 127.0.0.1 -p 6667 -u root -pw root -disableISO8601
```

Research Papers

Apache IoTDB starts at Tsinghua University, School of Software. IoTDB is a database for managing large amount of time series data with columnar storage, data encoding, pre-computation, and index techniques. It has SQL-like interface to write millions of data points per second per node and is optimized to get query results in few seconds over trillions of data points. It can also be easily integrated with Apache Hadoop MapReduce and Apache Spark for analytics.

The research papers related are as follows:

- Apache IoTDB: time-series database for internet of things (opens new window), Chen Wang, Xiangdong Huang, Jialin Qiao, Tian Jiang, Lei Rui, Jinrui Zhang, Rong Kang, Julian Feinauer, Kevin A. McGrail, Peng Wang, Jun Yuan, Jianmin Wang, Jiaguang Sun. VLDB 2020
- PISA: An Index for Aggregating Big Time Series Data (opens new window), Xiangdong Huang and Jianmin Wang and Raymond K. Wong and Jinrui Zhang and Chen Wang. CIKM 2016.
- Matching Consecutive Subpatterns over Streaming Time Series (opens new window), Rong Kang and Chen Wang and Peng Wang and Yuting Ding and Jianmin Wang. APWeb/WAIM 2018.
- KV-match: A Subsequence Matching Approach Supporting Normalization and Time Warping (opens new window), Jiaye Wu and Peng Wang and Chen Wang and Wei Wang and Jianmin Wang. ICDE 2019.
- The Design of Apache IoTDB distributed framework (opens new window), Tianan Li, Jianmin Wang, Xiangdong Huang, Yi Xu, Dongfang Mao, Jun Yuan. NDBC 2019
- Dual-PISA: An index for aggregation operations on time series data (opens new window), Jialin Qiao, Xiangdong Huang, Jianmin Wang, Raymond K Wong. IS 2020

Benchmark tools

We also developed Benchmark tools for time series databases

https://github.com/thulab/iotdb-benchmark

- What is IoTDB
- Architecture
- Scenario
- Features

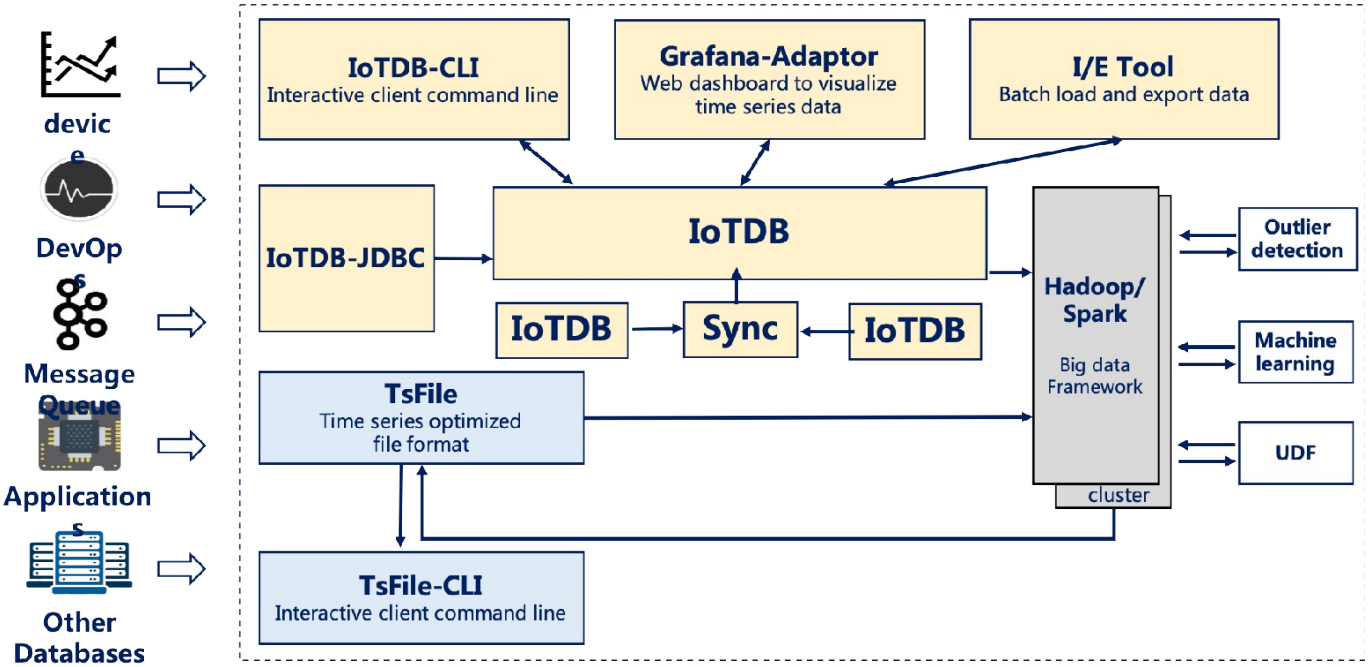
What is IoTDB

IoTDB(Internet of Things Database) is an integrated data management engine designed for timeseries data, which can provide users specific services for data collection, storage and analysis. Due to its light weight structure, high performance and usable features together with its intense integration with Hadoop and Spark ecology, IoTDB meets the requirements of massive dataset storage, high-speed data input and complex data analysis in the IoT industrial field.

Architecture

Besides IoTDB engine, we also developed several components to provide better IoT service. All components are referred to below as the IoTDB suite, and IoTDB refers specifically to the IoTDB engine.

IoTDB suite can provide a series of functions in the real situation such as data collection, data writing, data storage, data query, data visualization and data analysis. Figure 1.1 shows the overall application architecture brought by all the components of the IoTDB suite.



As shown in Figure 1.1, users can use JDBC to import timeseries data collected by sensor on the device to local/remote IoTDB. These timeseries data may be system state data (such as server load and CPU memory, etc.), message queue data, timeseries data from applications, or other timeseries data in the database. Users can also write the data directly to the TsFile (local or on HDFS).

TsFile could be written to the HDFS, thereby implementing data processing tasks such as abnormality detection and machine learning on the Hadoop or Spark data processing platform.

For the data written to HDFS or local TsFile, users can use TsFile-Hadoop-Connector or TsFile-Spark-Connector to allow Hadoop or Spark to process data.

The results of the analysis can be write back to TsFile in the same way.

Also, IoTDB and TsFile provide client tools to meet the various needs of users in writing and viewing data in SQL form, script form and graphical form.

Scenario

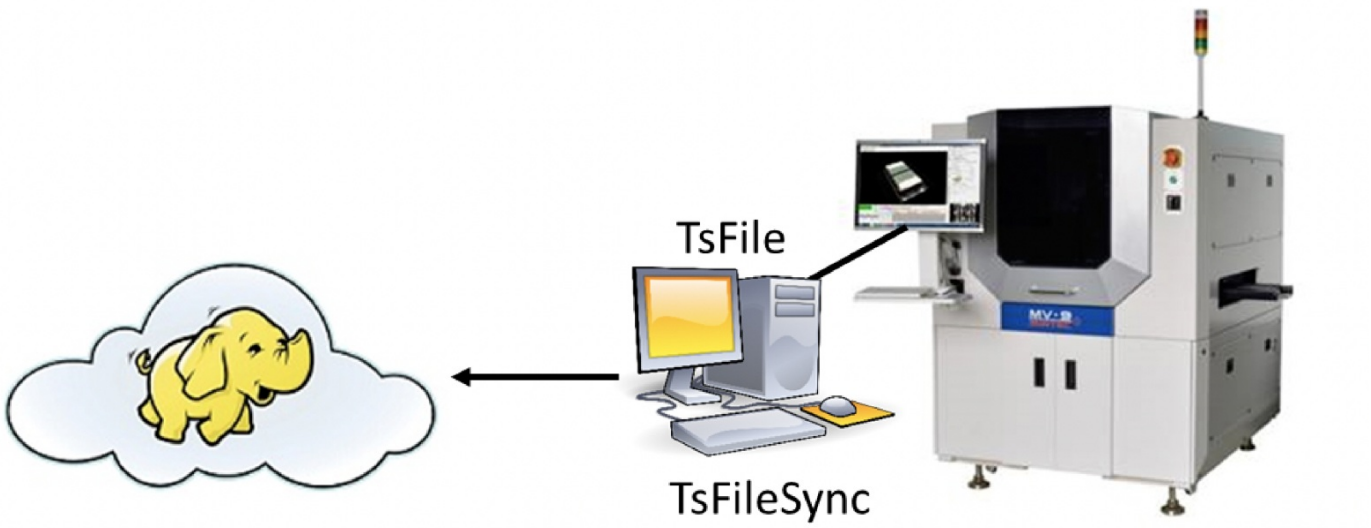
Scenario 1

A company uses surface mount technology (SMT) to produce chips: it is necessary to first print solder paste on the joints of the chip, then place the components on the solder paste, and then melt the solder paste by heating and cool it. Finally, the components are soldered to the chip.

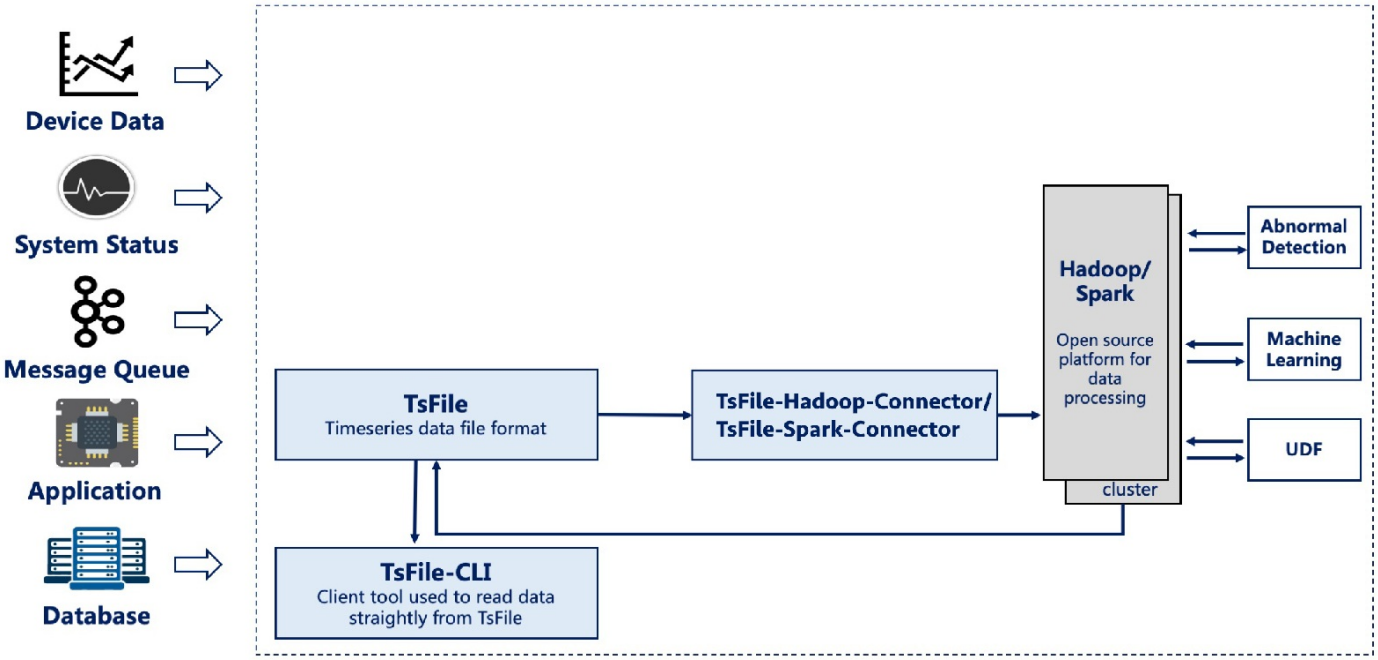
The above process uses an automated production line. In order to ensure the quality of the product, after printing the solder paste, the quality of the solder paste printing needs to be evaluated by optical equipment. The volume (v), height (h), area (a), horizontal offset (px), and vertical offset (py) of the solder paste on each joint are measured by a three-dimensional solder paste printing (SPI) device.

In order to improve the quality of the printing, it is necessary for the company to store the metrics of the solder joints on each chip for subsequent analysis based on these data.

At this point, the data can be stored using TsFile component, TsFileSync tool, and Hadoop/Spark integration component in the IoTDB suite.That is, each time a new chip is printed, a data is written on the SPI device using the SDK, which ultimately forms a TsFile. Through the TsFileSync tool, the generated TsFile will be synchronized to the data center according to certain rules (such as daily) and analyzed by data analysts tools.



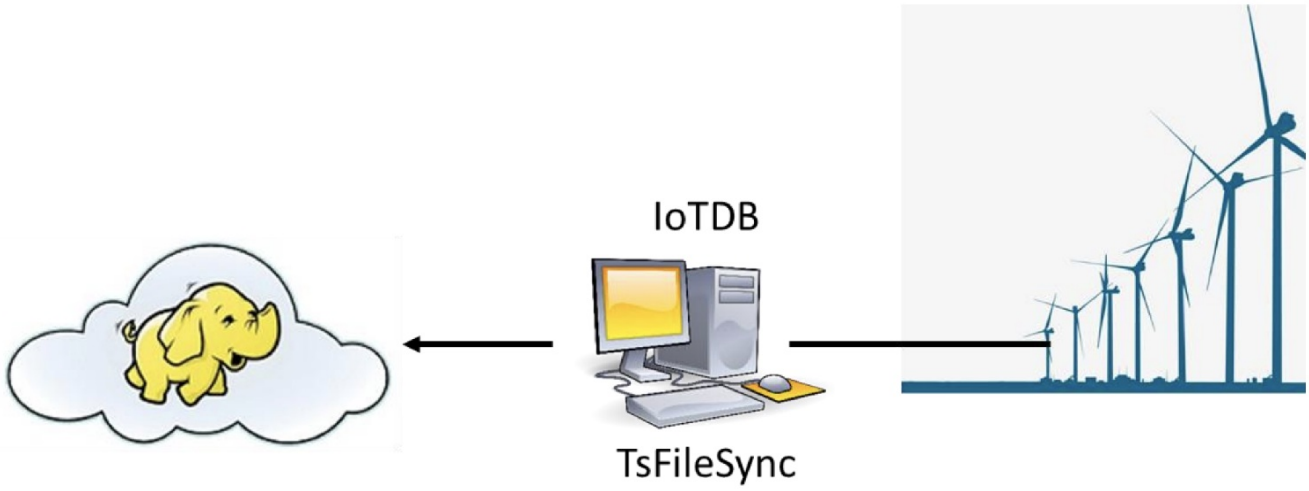
In this scenario, only TsFile and TsFileSync are required to be deployed on a PC, and a Hadoop/Spark cluster is required. The schematic diagram is shown in Figure 1.2. Figure 1.3 shows the architecture at this time.



Scenario 2

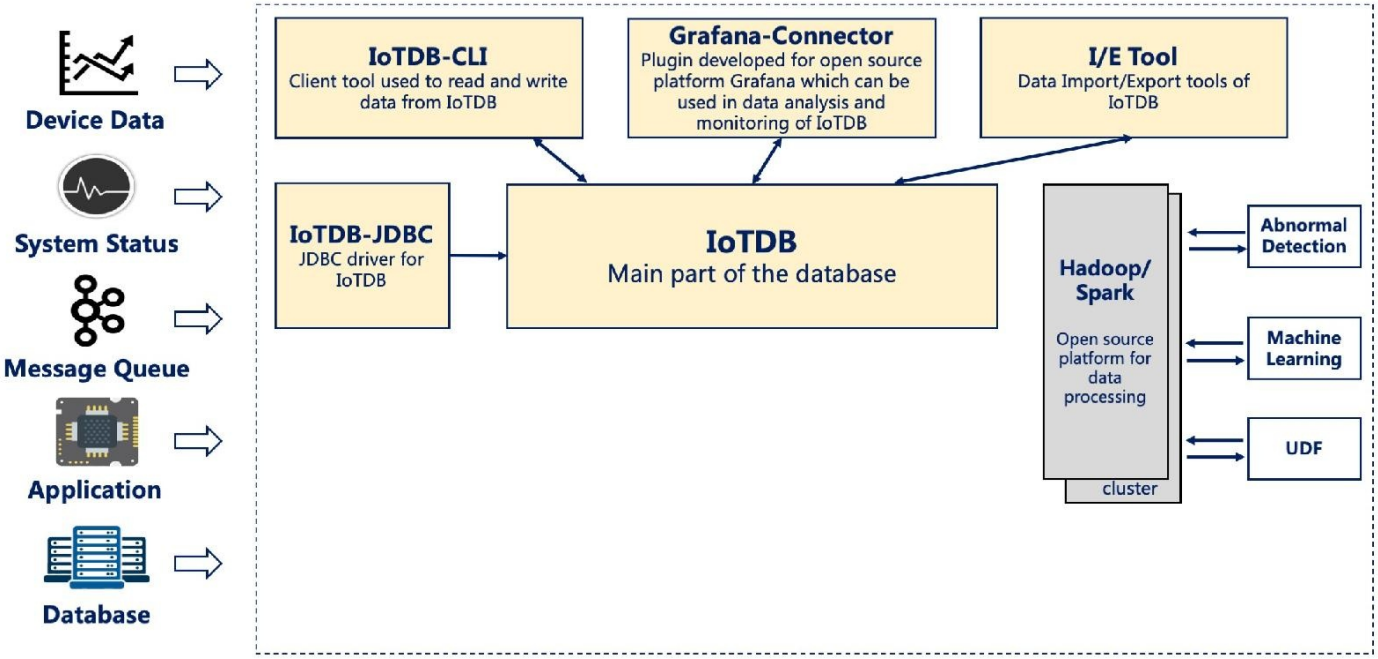
A company has several wind turbines which are installed hundreds of sensors on each generator to collect information such as the working status of the generator and the wind speed in the working environment.

In order to ensure the normal operation of the turbines and timely monitoring and analysis of the turbines, the company needs to collect these sensor data, perform partial calculation and analysis in the turbines working environment, and upload the original data collected to the data center.



In this situation, IoTDB, TsFileSync tools, and Hadoop/Spark integration components in the IoTDB suite can be used. A PC needs to be deployed with IoTDB and TsFileSync tools installed to support reading and writing data, local computing and analysis, and

uploading data to the data center. In addition, Hadoop/Spark clusters need to be deployed for data storage and analysis on the data center side. As shown in Figure 1.4. Figure 1.5 shows the architecture at this time.

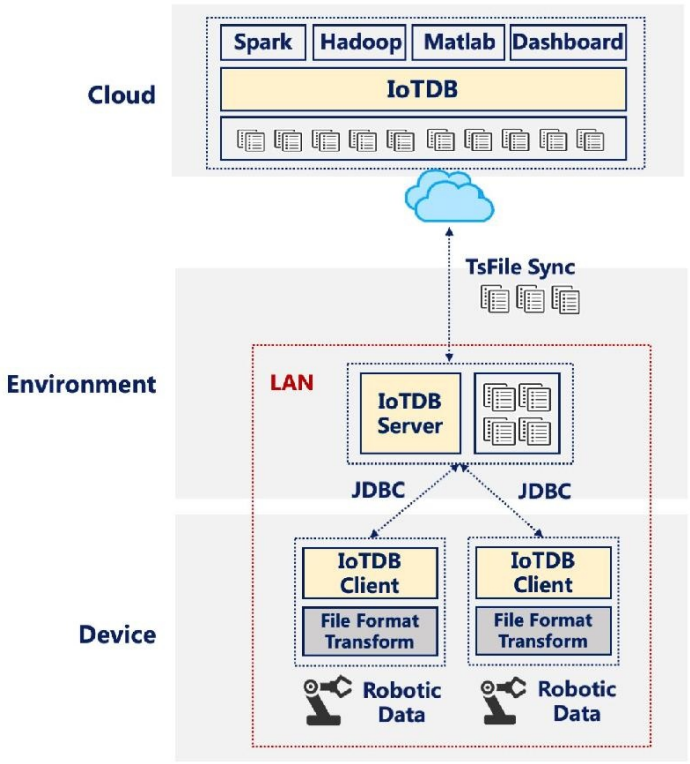


Scenario 3

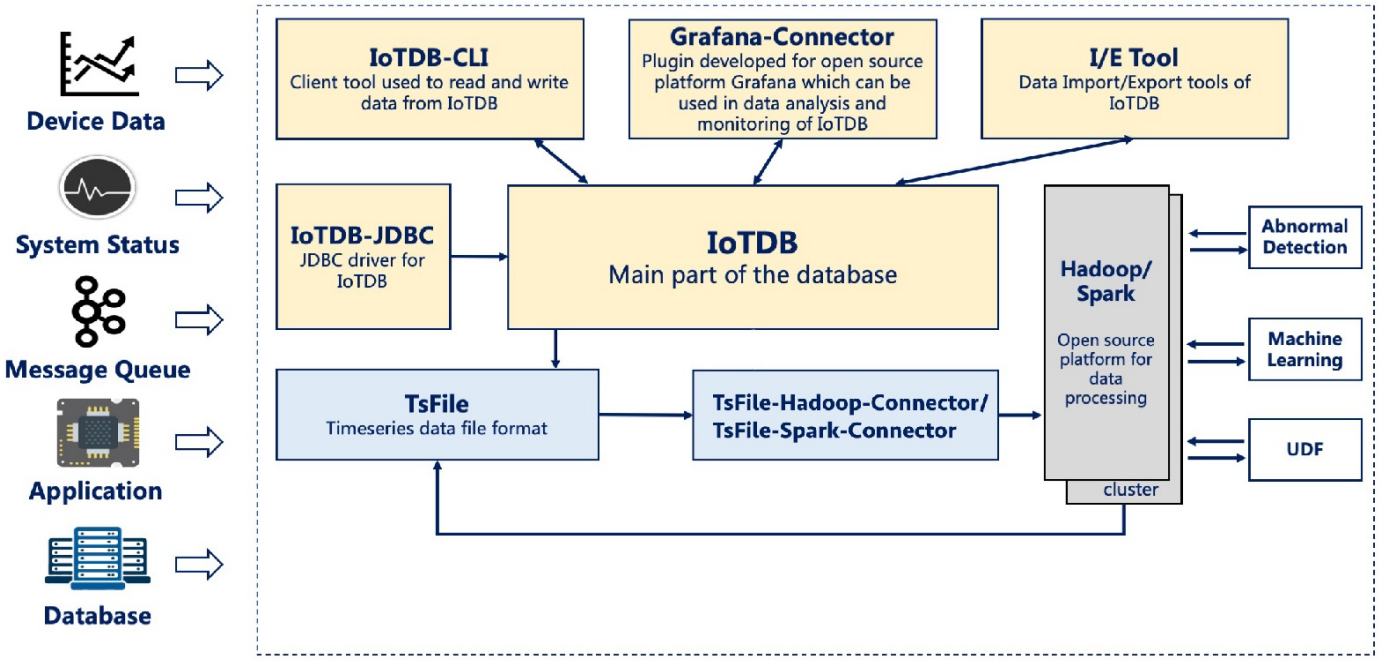
A factory has a variety of robotic equipment within the plant area. These robotic equipment have limited hardware and are difficult to carry complex applications.

A variety of sensors are installed on each robotic device to monitor the robot’s operating status, temperature, and other information. Due to the network environment of the factory, the robots inside the factory are all within the LAN of the factory and cannot connect to the external network. But there will be several servers in the factory that can connect directly to the external public network.

In order to ensure that the data of the robot can be monitored and analyzed in time, the company needs to collect the information of these robot sensors, send them to the server that can connect to the external network, and then upload the original data information to the data center for complex calculation and analysis.



At this point, IoTDB, IoTDB-Client tools, TsFileSync tools, and Hadoop/Spark integration components in the IoTDB suite can be used. IoTDB-Client tool is installed on the robot and each of them is connected to the LAN of the factory. When sensors generate real-time data, the data will be uploaded to the server in the factory. The IoTDB server and TsFileSync is installed on the server connected to the external network. Once triggered, the data on the server will be upload to the data center. In addition, Hadoop/Spark clusters need to be deployed for data storage and analysis on the data center side. As shown in Figure 1.6. Figure 1.7 shows the architecture at this time.



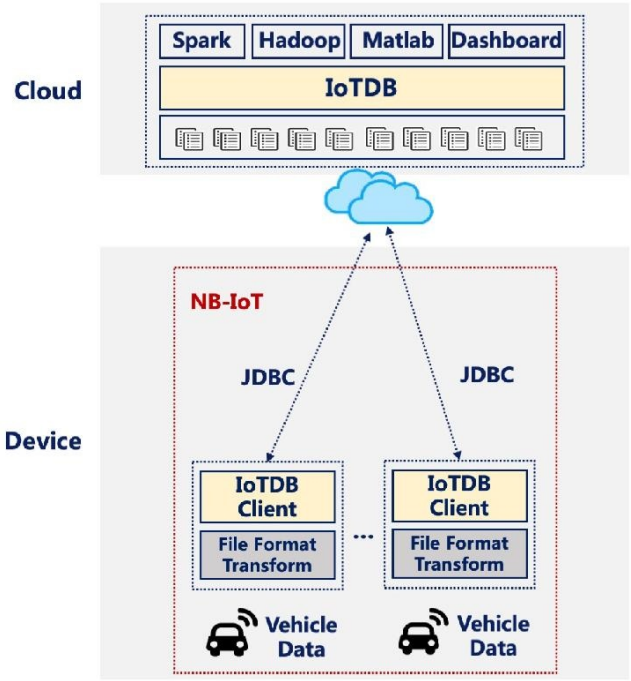
Scenario 4

A car company installed sensors on its cars to collect monitoring information such as the driving status of the vehicle. These automotive devices have limited hardware configurations and are difficult to carry complex applications. Cars with sensors can be connected to each other or send data via narrow-band IoT.

In order to receive the IoT data collected by the car sensor in real time, the company needs to send the sensor data to the data center in real time through the narrowband IoT while the vehicle is running. Thus, they can perform complex calculations and analysis on the server in the data center.

At this point, IoTDB, IoTDB-Client, and Hadoop/Spark integration components in the IoTDB suite can be used. IoTDB-Client tool is installed on each car and use IoTDB-JDBC tool to send data directly back to the server in the data center.

In addition, Hadoop/Spark clusters need to be deployed for data storage and analysis on the data center side. As shown in Figure 1.8.


Cloud
Spark
Hadoop
Matlab
Dashboard
IoTDB
NB-IoT
JDBC
JDBC
Device
IoTDB Client
File Format Transform
IoTDB Client
File Format Transform
Vehicle Data
Vehicle Data

Features

- Flexible deployment.

IoTDB provides users one-click installation tool on the cloud, once-decompressed-used terminal tool and the bridging tool between cloud platforms and terminal tools (Data Synchronization Tool).

- Low storage cost.

IoTDB can reach a high compression ratio of disk storage, which means IoTDB can store the same amount of data with less hardware disk cost.

- Efficient directory structure.

IoTDB supports efficient oganization for complex timeseries data structure from intelligent networking devices, oganization for timeseries data from devices of the same type, fuzzy searching strategy for massive and complex directory of timeseries data.

- High-throughput read and write.

IoTDB supports millions of low-power devices’ strong connection data access, high-speed data read and write for intelligent networking devices and mixed devices mentioned above.

- Rich query semantics.

IoTDB supports time alignment for timeseries data accross devices and sensors, computation in timeseries field (frequency domain transformation) and rich aggregation function support in time dimension.

- Easy to get started.

IoTDB supports SQL-Like language, JDBC standard API and import/export tools which are easy to use.

- Intense integration with Open Source Ecosystem.

IoTDB supports Hadoop, Spark, etc. analysis ecosystems and Grafana visualization tool.

- Data Model and Terminology
- Data Type
- Encoding
- Compression

Data Model and Terminology

In this section, a power scenario is taken as an example to illustrate how to creat a correct data model in IoTDB. For convenience, a sample data file is attached for you to practise IoTDB.

Download the attachment: IoTDB-SampleData.txt (opens new window).

According to the data attribute layers described in sample data (opens new window), it is expressed as an attribute hierarchy structure based on the coverage of attributes and the subordinate relationship between them, as shown in Figure 2.1 below. The hierarchical from top to bottom is: power group layer - power plant layer - device layer - sensor layer. ROOT is the root node, and each node of sensor layer is a leaf node. In the process of using IoTDB, the attributes on the path from ROOT node is directly connected to each leaf node with “.”, thus forming the name of a timeseries in IoTDB. For example, The left-most path in Figure 2.1 can generate a timeseries named `ROOT.ln.wf01.wt01.status` .

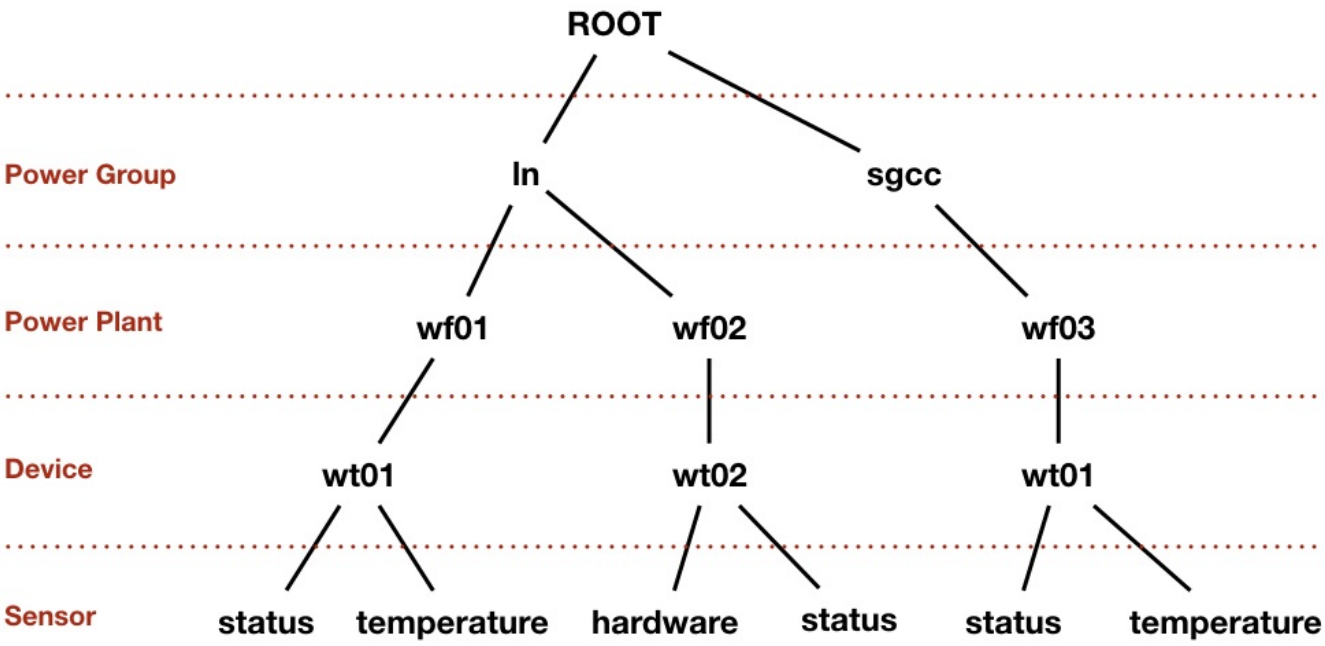


Figure 2.1 Attribute hierarchy structure

After getting the name of the timeseries, we need to set up the storage group according to the actual scenario and scale of the data. Because in the scenario of this chapter data is usually arrived in the unit of groups (i.e., data may be across electric fields and devices), in order to avoid frequent switch of IO when writing data, and meet the user’s requirement of physical isolation of data in the unit of groups, storage group is set at the group layer.

Here are the basic concepts of the model involved in IoTDB:

- Device

A device is an installation equipped with sensors in real scenarios. In IoTDB, all sensors should have their corresponding devices.

- Sensor

A sensor is a detection equipment in an actual scene, which can sense the information to be measured, and can transform the sensed information into an electrical signal or other desired form of information output and send it to IoTDB. In IoTDB, all data and paths stored are organized in units of sensors.

- Storage Group

Storage groups are used to let users define how to organize and isolate different time series data on disk. Time series belonging to the same storage group is continuously written to the same file in the corresponding folder. The file may be closed due to user commands or system policies, and hence the data coming next from these sensors will be stored in a new file in the same folder. Time series belonging to different storage groups are stored in different folders.

Users can set any prefix path as a storage group. Provided that there are four time series `root.vehicle.d1.s1` , `root.vehicle.d1.s2` , `root.vehicle.d2.s1` , `root.vehicle.d2.s2` , two devices `d1` and `d2` under the path `root.vehicle` may belong to the same owner or the same manufacturer, so d1 and d2 are closely related. At this point, the prefix path root.vehicle can be designated as a storage group, which will enable IoTDB to store all devices under it in the same folder. Newly added devices under `root.vehicle` will also belong to this storage group.

> Note: A full path (`root.vehicle.d1.s1` as in the above example) is not allowed to be set as a storage group.

Setting a reasonable number of storage groups can lead to performance gains: there is neither the slowdown of the system due to frequent switching of IO (which will also take up a lot of memory and result in frequent memory-file switching) caused by too many storage files (or folders), nor the block of write commands caused by too few storage files (or folders) (which reduces concurrency).

Users should balance the storage group settings of storage files according to their own data size and usage scenarios to achieve better system performance. (There will be officially provided storage group scale and performance test reports in the future).

> Note: The prefix of a time series must belong to a storage group. Before creating a time series, the user must set which storage group the series belongs to. Only the time series whose storage group is set can be persisted to disk.

Once a prefix path is set as a storage group, the storage group settings cannot be changed.

After a storage group is set, all parent and child layers of the corresponding prefix path are not allowed to be set up again (for example, after `root.ln` is set as the storage group, the root layer and `root.ln.wf01` are not allowed to be set as storage groups).

- Path

In IoTDB, a path is an expression that conforms to the following constraints:

```
path: LayerName (DOT LayerName)+
LayerName: Identifier | STAR
```

Among them, STAR is "*" and DOT is ".".

We call the middle part of a path between two "." as a layer, and thus `root.A.B.C` is a path with four layers.

It is worth noting that in the path, root is a reserved character, which is only allowed to appear at the beginning of the time series mentioned below. If root appears in other layers, it cannot be parsed and an error is reported.

Single quotes are not allowed in the path. If you want to use special characters such as "." in LayerName, use double quotes. For example, `root.sg."d.1"."s.1"` .

> Note: the LayerName of storage group can only be characters, numbers and underscores.

- Timeseries Path

The timeseries path is the core concept in IoTDB. A timeseries path can be thought of as the complete path of a sensor that produces the time series data. All timeseries paths in IoTDB must start with root and end with the sensor. A timeseries path can also be called a full path.

For example, if device1 of the vehicle type has a sensor named sensor1, its timeseries path can be expressed as: `root.vehicle.device1.sensor1` . Double quotes can be nested with escape characters, e.g. `root.sg.d1."s.\"t\"1"` .

> Note: The layer of timeseries paths supported by the current IoTDB must be greater than or equal to four (it will be changed to two in the future).

- Prefix Path

The prefix path refers to the path where the prefix of a timeseries path is located. A prefix path contains all timeseries paths prefixed by the path. For example, suppose that we have three sensors: `root.vehicle.device1.sensor1` , `root.vehicle.device1.sensor2` , `root.vehicle.device2.sensor1` , the prefix path `root.vehicle.device1` contains two timeseries paths `root.vehicle.device1.sensor1` and `root.vehicle.device1.sensor2` while `root.vehicle.device2.sensor1` is excluded.

- Path With Star

In order to make it easier and faster to express multiple timeseries paths or prefix paths, IoTDB provides users with the path pith star. `*` can appear in any layer of the path. According to the position where `*` appears, the path with star can be divided into two types:

`*` appears at the end of the path;

`*` appears in the middle of the path;

When `*` appears at the end of the path, it represents (`*`)+, which is one or more layers of `*` . For example, `root.vehicle.device1.*` represents all paths prefixed by `root.vehicle.device1` with layers greater than or equal to 4, like `root.vehicle.device1.*` , `root.vehicle.device1.*.*` , `root.vehicle.device1.*.*.*` , etc.

When `*` appears in the middle of the path, it represents `*` itself, i.e., a layer. For example, `root.vehicle.*.sensor1` represents a 4-layer path which is prefixed with `root.vehicle` and suffixed with `sensor1` .

> Note1: `*` cannot be placed at the beginning of the path.
>
> Note2: A path with `*` at the end has the same meaning as a prefix path, e.g., `root.vehicle.*` and `root.vehicle` is the same.

- Timestamp

The timestamp is the time point at which data is produced. It includes absolute timestamps and relative timestamps

- Absolute timestamp

Absolute timestamps in IoTDB are divided into two types: LONG and DATETIME (including DATETIME-INPUT and DATETIME-DISPLAY). When a user inputs a timestamp, he can use a LONG type timestamp or a DATETIME-INPUT type timestamp, and the supported formats of the DATETIME-INPUT type timestamp are shown in the table below:

Supported formats of DATETIME-INPUT type timestamp

Format
yyyy-MM-dd HH:mm:ss
yyyy/MM/dd HH:mm:ss
yyyy.MM.dd HH:mm:ss
yyyy-MM-dd’T’HH:mm:ss
yyyy/MM/dd’T’HH:mm:ss
yyyy.MM.dd’T’HH:mm:ss
yyyy-MM-dd HH:mm:ssZZ
yyyy/MM/dd HH:mm:ssZZ
yyyy.MM.dd HH:mm:ssZZ
yyyy-MM-dd’T’HH:mm:ssZZ

yyyy/MM/dd'T'HH:mm:ssZZ
yyyy.MM.dd'T'HH:mm:ssZZ
yyyy/MM/dd HH:mm:ss.SSS
yyyy-MM-dd HH:mm:ss.SSS
yyyy.MM.dd HH:mm:ss.SSS
yyyy/MM/dd'T'HH:mm:ss.SSS
yyyy-MM-dd'T'HH:mm:ss.SSS
yyyy.MM.dd'T'HH:mm:ss.SSS
yyyy-MM-dd HH:mm:ss.SSSZZ
yyyy/MM/dd HH:mm:ss.SSSZZ
yyyy.MM.dd HH:mm:ss.SSSZZ
yyyy-MM-dd'T'HH:mm:ss.SSSZZ
yyyy/MM/dd'T'HH:mm:ss.SSSZZ
yyyy.MM.dd'T'HH:mm:ss.SSSZZ
ISO8601 standard time format

IoTDB can support LONG types and DATETIME-DISPLAY types when displaying timestamps. The DATETIME-DISPLAY type can support user-defined time formats. The syntax of the custom time format is shown in the table below:

The syntax of the custom time format

Symbol	Meaning	Presentation	Examples
G	era	era	era
C	century of era (>=0)	number	20
Y	year of era (>=0)	year	1996
x	weekyear	year	1996
w	week of weekyear	number	27
e	day of week	number	2
E	day of week	text	Tuesday; Tue
y	year	year	1996
D	day of year	number	189
M	month of year	month	July; Jul; 07
d	day of month	number	10
a	halfday of day	text	PM
K	hour of halfday (0~11)	number	0
h	clockhour of halfday (1~12)	number	12

H	hour of day (0~23)	number	0
k	clockhour of day (1~24)	number	24
m	minute of hour	number	30
s	second of minute	number	55
S	fraction of second	millis	978
z	time zone	text	Pacific Standard Time; PST
Z	time zone offset/id	zone	-0800; -08:00; America/Los_Angeles
‘	escape for text	delimiter	
‘’	single quote	literal	‘

- Relative timestamp

Relative time refers to the time relative to the server time `now()` and `DATETIME` time.

Syntax:

```
1.  Duration = (Digit+ ('Y'|'MO'|'W'|'D'|'H'|'M'|'S'|'MS'|'US'|'NS'))+
2.  RelativeTime = (now() | DATETIME) ((+|-) Duration)+
```

The syntax of the duration unit

Symbol	Meaning	Presentation	Examples
y	year	1y=365 days	1y
mo	month	1mo=30 days	1mo
w	week	1w=7 days	1w
d	day	1d=1 day	1d
h	hour	1h=3600 seconds	1h
m	minute	1m=60 seconds	1m
s	second	1s=1 second	1s
ms	millisecond	1ms=1000_000 nanoseconds	1ms
us	microsecond	1us=1000 nanoseconds	1us
ns	nanosecond	1ns=1 nanosecond	1ns

eg:

```
1.  now() - 1d2h //1 day and 2 hours earlier than the current server time
2.  now() - 1w //1 week earlier than the current server time
```

> Note: There must be spaces on the left and right of ‘+’ and ‘-’.

Data Type

IoTDB supports six data types in total:

- BOOLEAN (Boolean)
- INT32 (Integer)
- INT64 (Long Integer)
- FLOAT (Single Precision Floating Point)
- DOUBLE (Double Precision Floating Point)
- TEXT (String).

The time series of **FLOAT** and **DOUBLE** type can specify (MAX_POINT_NUMBER, see this page for more information on how to specify), which is the number of digits after the decimal point of the floating point number, if the encoding method is RLE or TS_2DIFF (Refer to Create Timeseries Statement for more information on how to specify). If MAX_POINT_NUMBER is not specified, the system will use float_precision in the configuration file `iotdb-engine.properties` .

- For Float data value, The data range is (-Integer.MAX_VALUE, Integer.MAX_VALUE), rather than Float.MAX_VALUE, and the max_point_number is 19, caused by the limition of function Math.round(float) in Java.
- For Double data value, The data range is (-Long.MAX_VALUE, Long.MAX_VALUE), rather than Double.MAX_VALUE, and the max_point_number is 19, caused by the limition of function Math.round(double) in Java (Long.MAX_VALUE=9.22E18).

When the data type of data input by the user in the system does not correspond to the data type of the time series, the system will report type errors. As shown below, the second-order difference encoding does not support the Boolean type:

```
1. IoTDB> create timeseries root.ln.wf02.wt02.status WITH DATATYPE=BOOLEAN, ENCODING=TS_2DIFF
2. error: encoding TS_2DIFF does not support BOOLEAN
```

Encoding

To improve the efficiency of data storage, it is necessary to encode data during data writing, thereby reducing the amount of disk space used. In the process of writing and reading data, the amount of data involved in the I/O operations can be reduced to improve performance. IoTDB supports four encoding methods for different data types:

- PLAIN

PLAIN encoding, the default encoding mode, i.e, no encoding, supports multiple data types. It has high compression and decompression efficiency while suffering from low space storage efficiency.

- TS_2DIFF

Second-order differential encoding is more suitable for encoding monotonically increasing or decreasing sequence data, and is not recommended for sequence data with large fluctuations.

- RLE

Run-length encoding is suitable for storing sequence with continuous integer values, and is not recommended for sequence data with most of the time different values.

Run-length encoding can also be used to encode floating-point numbers, while it is necessary to specify reserved decimal digits (MAX_POINT_NUMBER, see this page for more information on how to specify) when creating time series. It is more suitable to store sequence data where floating-point values appear continuously, monotonously increasing or decreasing, and it is not suitable for storing sequence data with high precision requirements after the decimal point or with large fluctuations.

> TS_2DIFF and RLE have precision limit for data type of float and double. By default, two decimal places are reserved. GORILLA is recommended.

- GORILLA

GORILLA encoding is lossless. It is more suitable for numerical sequence with similar values and is not recommended for sequence data with large fluctuations.

Currently, there are two versions of GORILLA encoding implementation, it is recommended to use `GORILLA` instead of `GORILLA_V1` (deprecated).

Usage restrictions: When using GORILLA to encode INT32 data, you need to ensure that there is no data point with the value `Integer.MIN_VALUE` in the sequence. When using GORILLA to encode INT64 data, you need to ensure that there is no data point with the value `Long.MIN_VALUE` in the sequence.

- REGULAR

Regular data encoding is more suitable for encoding regular sequence increasing data (e.g. the timeseries with the same time elapsed between each data point), in which case it’s better than TS_2DIFF.

Regular data encoding method is not suitable for the data with fluctuations (irregular data), and TS_2DIFF is recommended to deal with it.

- Correspondence between data type and encoding

The four encodings described in the previous sections are applicable to different data types. If the correspondence is wrong, the time series cannot be created correctly. The correspondence between the data type and its supported encodings is summarized in Table 2-3.

Table 2-3 The correspondence between the data type and its supported encodings

Data Type	Supported Encoding
BOOLEAN	PLAIN, RLE
INT32	PLAIN, RLE, TS_2DIFF, REGULAR, GORILLA
INT64	PLAIN, RLE, TS_2DIFF, REGULAR, GORILLA
FLOAT	PLAIN, RLE, TS_2DIFF, GORILLA
DOUBLE	PLAIN, RLE, TS_2DIFF, GORILLA
TEXT	PLAIN

Compression

When the time series is written and encoded as binary data according to the specified type, IoTDB compresses the data using compression technology to further improve space storage efficiency. Although both encoding and compression are designed to improve storage efficiency, encoding techniques are usually available only for specific data types (e.g., second-order differential encoding is only suitable for INT32 or INT64 data type, and storing floating-point numbers requires multiplying them by 10m to convert to integers), after which the data is converted to a binary stream. The compression method (SNAPPY) compresses the binary stream, so the use of the compression method is no longer limited by the data type.

IoTDB allows you to specify the compression method of the column when creating a time series, and supports three compression methods:

- UNCOMPRESSED
- SNAPPY
- LZ4

The specified syntax for compression is detailed in Create Timeseries Statement.

- Download
- Single Node Setup
- Cluster Setup
- Config Manual
- Docker Image

Download

IoTDB provides you three installation methods, you can refer to the following suggestions, choose one of them:

- Installation from source code. If you need to modify the code yourself, you can use this method.
- Installation from binary files. Download the binary files from the official website. This is the recommended method, in which you will get a binary released package which is out-of-the-box.
- Using Docker: The path to the dockerfile is https://github.com/apache/iotdb/blob/master/docker/Dockerfile

Prerequisites

To use IoTDB, you need to have:

1. Java >= 1.8 (Please make sure the environment path has been set)
2. Maven >= 3.1 (Optional)
3. Set the max open files num as 65535 to avoid “too many open files” problem.

> Note: If you don’t have maven installed, you should replace ‘mvn’ in the following commands with ‘mvnw.sh’ or ‘mvnw.cmd’.
>
> ### Installation from binary files

You can download the binary file from: Here

Installation from source code

You can get the released source code from https://iotdb.apache.org/Download/, or from the git repository https://github.com/apache/iotdb/tree/master You can download the source code from:

```
1. git clone https://github.com/apache/iotdb.git
```

Under the root path of iotdb:

```
1. > mvn clean package -DskipTests
```

Then the binary version (including both server and client) can be found at **distribution/target/apache-iotdb-{project.version}-bin.zip**

> NOTE: Directories “thrift/target/generated-sources/thrift” and “antlr/target/generated-sources/antlr4” need to be added to sources roots to avoid compilation errors in IDE.

If you would like to build the IoTDB server, you can run the following command under the root path of iotdb:

```
> mvn clean package -pl server -am -DskipTests
```

After build, the IoTDB server will be at the folder “server/target/iotdb-server-{project.version}”.

Installation by Docker (Dockerfile)

You can build and run a IoTDB docker image by following the guide of Deployment by Docker

Single Node Setup

Users can start IoTDB by the start-server script under the sbin folder.

```
# Unix/OS X
> nohup sbin/start-server.sh >/dev/null 2>&1 &
or
> nohup sbin/start-server.sh -c <conf_path> -rpc_port <rpc_port> >/dev/null 2>&1 &

# Windows
> sbin\start-server.bat -c <conf_path> -rpc_port <rpc_port>
```

- “-c” and “-rpc_port” are optional.
- option “-c” specifies the system configuration file directory.
- option “-rpc_port” specifies the rpc port.
- if both option specified, the *rpc_port* will overrides the rpc_port in *conf_path*.

Cluster Setup

For installation prerequisites, please refer to Installation Prerequisites

Start Service

Users can build clusters in pseudo-distributed mode or distributed mode. The main difference between pseudo-distributed mode and distributed mode is the difference in `seed_nodes` in the configuration file. For detail descriptions, please refer to [Cluster Configuration Items](#Cluster Configuration Items).

To start the service of one of the nodes, you need to execute the following commands:

```
# Unix/OS X
> nohup sbin/start-node.sh >/dev/null 2>&1 &
or
> nohup sbin/start-node.sh -c <conf_path> -internal_meta_port 9003 >/dev/null 2>&1 &

# Windows
> nohup sbin/start-node.bat
or
> nohup sbin/start-node.bat -c <conf_path> -internal_meta_port 9003
```

`-c <conf_path>` use the configuration file in the `conf_path` folder to override the default configuration file; `-internal_meta_port 9003` overrides the specific configuration item `internal_meta_port`. The currently supported items to overwrite the original configurations when starting IoTDB are the followings : `internal_meta_port, internal_data_port, cluster_rpc_port, seed_nodes`. When both exist, the specified configuration item will overwrite the configurations in the configuration file.

Cluster Configuration Items

Before starting to use IoTDB, you need to config the configuration files first. For your convenience, we have already set the default config in the files.

In total, we provide users four kinds of configurations module:

- environment configuration file (`iotdb-env.bat` , `iotdb-env.sh`). The default configuration file for the environment configuration item. Users can configure the relevant system configuration items of JAVA-JVM in the file.

- system configuration file (`iotdb-engine.properties`). The default configuration file for the IoTDB engine layer configuration item. Users can configure the IoTDB engine related parameters in the file, such as the precision of timestamp(`timestamp_precision`), etc. What’s more, Users can configure settings of

TsFile (the data files), such as the data size written to the disk per time(`group_size_in_byte`).

- log configuration file (`logback.xml`). The default log configuration file, such as the log levels.

- `iotdb-cluster.properties` . Some configurations required by IoTDB cluster. Such as replication number(`default_replica_num`), etc.

For detailed description of the two configuration files `iotdb-engine.properties` , `cluster-env.sh` / `cluster-env.bat` , please refer to Configuration Manual. The configuration items described below are in the `iotdb-cluster.properties` file.

- internal_meta_port

Name	internal_meta_port
Description	IoTDB meta service port, for meta group’s communication, which involves all nodes and manages the cluster configuration and storage groups. **IoTDB will automatically create a heartbeat port for each meta service. The default meta service heartbeat port is `internal_meta_port+1` , please confirm that these two ports are not reserved by the system and are not occupied**
Type	Int32
Default	9003
Effective	After restart system, shall NOT change after cluster is up

- internal_data_port

Name	internal_data_port
Description	IoTDB data service port, for data groups’ communication, each consists of one node and its replicas, managing timeseries schemas and data. **IoTDB will automatically create a heartbeat port for each data service. The default data service heartbeat port is `internal_data_port+1` . Please confirm that these two ports are not reserved by the system and are not occupied**
Type	Int32
Default	40010
Effective	After restart system, shall NOT change after cluster is up

- cluster_rpc_port

Name	cluster_rpc_port
Description	The port used to communicate with the clients (JDBC, CLI, Session API…), please confirm that the port is not reserved by the system and is not occupied
Type	Int32
Default	55560
Effective	After restart system

- seed_nodes

Name	seed_nodes
Description	The address of the nodes in the cluster, `{IP/DOMAIN}:internal_meta_port:internal_data_port` format, separated by commas; for the pseudo-distributed mode, you can fill in `localhost`, or `127.0.0.1` or mixed, but the real ip address cannot appear; for the distributed mode, real ip or hostname is supported, but `localhost` or `127.0.0.1` cannot appear. When used by `start-node.sh(.bat)`, this configuration means the nodes that will form the initial cluster, so every node that use `start-node.sh(.bat)` should have the same `seed_nodes`, or the building of the initial cluster will fail. WARNING: if the initial cluster is built, this should not be changed before the environment is cleaned. When used by `add-node.sh(.bat)`, this means the nodes to which that the application of joining the cluster will be sent, as all nodes can respond to a request, this configuration can be any nodes that already in the cluster, unnecessary to be the nodes that were used to build the initial cluster by `start-node.sh(.bat)`. Several nodes will be picked randomly to send the request, the number of nodes picked depends on the number of retries.
Type	String
Default	127.0.0.1:9003:40010,127.0.0.1:9005:40012,127.0.0.1:9007:40014
Effective	After restart system

- rpc_thrift_compression_enable

Name	rpc_thrift_compression_enable
Description	Whether to enable thrift compression, **Note that this parameter should be consistent with each node and with the client, and also consistent with the `rpc_thrift_compression_enable` parameter in `iotdb-engine.properties`**
Type	Boolean
Default	false
Effective	After restart system, must be changed with all other nodes

- default_replica_num

Name	default_replica_num
Description	Number of cluster replicas of timeseries schema and data. Storage group info is always fully replicated in all nodes.
Type	Int32
Default	2
Effective	After restart system, shall NOT change after cluster is up

- cluster_name

Name	cluster_name
Description	Cluster name is used to identify different clusters, **`cluster_name` of all nodes in a cluster must be the same**
Type	String

Default	default
Effective	After restart system

- connection_timeout_ms

Name	connection_timeout_ms
Description	Heartbeat timeout time period between nodes in the same raft group, in milliseconds
Type	Int32
Default	20000
Effective	After restart system

- read_operation_timeout_ms

Name	read_operation_timeout_ms
Description	Read operation timeout time period, for internal communication only, not for the entire operation, in milliseconds
Type	Int32
Default	30000
Effective	After restart system

- write_operation_timeout_ms

Name	write_operation_timeout_ms
Description	The write operation timeout period, for internal communication only, not for the entire operation, in milliseconds
Type	Int32
Default	30000
Effective	After restart system

- min_num_of_logs_in_mem

Name	min_num_of_logs_in_mem
Description	The minimum number of committed logs in memory, after each log deletion, at most such number of logs will remain in memory. Increasing the number will reduce the chance to use snapshot in catch-ups, but will also increase the memory footprint
Type	Int32
Default	100
Effective	After restart system

- max_num_of_logs_in_mem

Name	max_num_of_logs_in_mem
	Maximum number of committed logs in memory, when reached, a log

Description	deletion will be triggered. Increasing the number will reduce the chance to use snapshot in catch-ups, but will also increase memory footprint
Type	Int32
Default	1000
Effective	After restart system

- log_deletion_check_interval_second

Name	log_deletion_check_interval_second
Description	The interval for checking and deleting the committed log task, which removes oldest in-memory committed logs when their size exceeds `min_num_of_logs_in_mem` , in seconds
Type	Int32
Default	60
Effective	After restart system

- enable_auto_create_schema

Name	enable_auto_create_schema
Description	Whether creating schema automatically is enabled, this will replace the one in `iotdb-engine.properties`
Type	BOOLEAN
Default	true
Effective	After restart system

- consistency_level

Name	consistency_level
Description	Consistency level, now three consistency levels are supported: strong, mid, and weak. Strong consistency means the server will first try to synchronize with the leader to get the newest data, if failed(timeout), directly report an error to the user; While mid consistency means the server will first try to synchronize with the leader, but if failed(timeout), it will give up and just use current data it has cached before; Weak consistency does not synchronize with the leader and simply use the local data
Type	strong, mid, weak
Default	mid
Effective	After restart system

- is_enable_raft_log_persistence

Name	is_enable_raft_log_persistence
Description	Whether to enable raft log persistence
Type	BOOLEAN
Default	true

Effective	After restart system

Enable GC log

GC log is off by default. For performance tuning, you may want to collect the GC info.

To enable GC log, just add a parameter `printgc` when you start the server.

```
1. nohup sbin/start-node.sh printgc >/dev/null 2>&1 &
```

Or

```
1. sbin\start-node.bat printgc
```

GC log is stored at `IOTDB_HOME/logs/gc.log`.

Config Manual

Before starting to use IoTDB, you need to config the configuration files first. For your convenience, we have already set the default config in the files.

In total, we provide users three kinds of configurations module:

- environment configuration file (`iotdb-env.bat` , `iotdb-env.sh`). The default configuration file for the environment configuration item. Users can configure the relevant system configuration items of JAVA-JVM in the file.
- system configuration file (`iotdb-engine.properties`).
    - `iotdb-engine.properties` : The default configuration file for the IoTDB engine layer configuration item. Users can configure the IoTDB engine related parameters in the file, such as JDBC service listening port (`rpc_port`), unsequence data storage directory (`unsequence_data_dir`), etc. What's more, Users can configure the information about the TsFile, such as the data size written to the disk per time(`group_size_in_byte`).
- log configuration file (`logback.xml`)

The configuration files of the three configuration items are located in the IoTDB installation directory: `$IOTDB_HOME/conf` folder.

Hot Modification Configuration

For the convenience of users, IoTDB server provides users with hot modification function, that is, modifying some configuration parameters in `iotdb engine. Properties` during the system operation and applying them to the system immediately. In the parameters described below, these parameters whose way of `Effective` is `trigger` support hot modification.

Trigger way: The client sends the command `load configuration` to the IoTDB server. See Chapter 4 for the usage of the client.

IoTDB Environment Configuration File

The environment configuration file is mainly used to configure the Java environment related parameters when IoTDB Server is running, such as JVM related configuration. This part of the configuration is passed to the JVM when the IoTDB Server starts. Users can view the contents of the environment configuration file by viewing the `iotdb-env.sh` (or `iotdb-env.bat`) file.

The detail of each variables are as follows:

- MAX_HEAP_SIZE

Name	MAX_HEAP_SIZE
Description	The maximum heap memory size that IoTDB can use at startup.
Type	String
Default	On Linux or MacOS, the default is one quarter of the memory. On Windows, the default value for 32-bit systems is 512M, and the default for 64-bit systems is 2G.
Effective	After restart system

- HEAP_NEWSIZE

Name	HEAP_NEWSIZE
Description	The minimum heap memory size that IoTDB can use at startup.
Type	String
Default	On Linux or MacOS, the default is min{cores * 100M, one quarter of MAX_HEAP_SIZE}. On Windows, the default value for 32-bit systems is 512M, and the default for 64-bit systems is 2G.
Effective	After restart system

- JMX_LOCAL

Name	JMX_LOCAL
Description	JMX monitoring mode, configured as yes to allow only local monitoring, no to allow remote monitoring
Type	Enum String: “true”, “false”
Default	true
Effective	After restart system

- JMX_PORT

Name	JMX_PORT
Description	JMX listening port. Please confirm that the port is not a system reserved port and is not occupied
Type	Short Int: [0,65535]
Default	31999
Effective	After restart system

- JMX_IP

Name	JMX_IP
Description	JMX listening address. Only take effect if JMX_LOCAL=false. 0.0.0.0 is never allowed
Type	String
Default	127.0.0.1
Effective	After restart system

JMX Authorization

We **STRONGLY RECOMMENDED** you CHANGE the PASSWORD for the JMX remote connection.

The user and passwords are in ${IOTDB_CONF}/conf/jmx.password.

The permission definitions are in ${IOTDB_CONF}/conf/jmx.access.

IoTDB System Configuration File

File Layer

- compressor

Name	compressor
Description	Data compression method
Type	Enum String : “UNCOMPRESSED”, “SNAPPY”
Default	UNCOMPRESSED
Effective	Trigger

- group_size_in_byte

Name	group_size_in_byte
Description	The data size written to the disk per time
Type	Int32
Default	134217728
Effective	Trigger

- page_size_in_byte

Name	page_size_in_byte
Description	The maximum size of a single page written in memory when each column in memory is written (in bytes)
Type	Int32
Default	65536
Effective	Trigger

- max_number_of_points_in_page

Name	max_number_of_points_in_page
Description	The maximum number of data points (timestamps - valued groups) contained in a page
Type	Int32

Default	1048576
Effective	Trigger

- max_degree_of_index_node

Name	max_degree_of_index_node
Description	The maximum degree of the metadata index tree (that is, the max number of each node’s children)
Type	Int32
Default	1024
Effective	Only allowed to be modified in first start up

- max_string_length

Name	max_string_length
Description	The maximum length of a single string (number of character)
Type	Int32
Default	128
Effective	Trigger

- time_series_data_type

Name	time_series_data_type
Description	Timestamp data type
Type	Enum String: “INT32”, “INT64”
Default	Int64
Effective	Trigger

- time_encoder

Name	time_encoder
Description	Encoding type of time column
Type	Enum String: “TS_2DIFF”,“PLAIN”,“RLE”
Default	TS_2DIFF
Effective	Trigger

- value_encoder

Name	value_encoder
Description	Encoding type of value column
Type	Enum String: “TS_2DIFF”,“PLAIN”,“RLE”
Default	PLAIN
Effective	Trigger

- float_precision

Name	float_precision
Description	The precision of the floating point number.(The number of digits after the decimal point)
Type	Int32
Default	The default is 2 digits. Note: The 32-bit floating point number has a decimal precision of 7 bits, and the 64-bit floating point number has a decimal precision of 15 bits. If the setting is out of the range, it will have no practical significance.
Effective	Trigger

- bloomFilterErrorRate

Name	bloomFilterErrorRate
Description	The false positive rate of bloom filter in each TsFile. Bloom filter checks whether a given time series is in the tsfile before loading metadata. This can improve the performance of loading metadata and skip the tsfile that doesn't contain specified time series. If you want to learn more about its mechanism, you can refer to: wiki page of bloom filter (opens new window).
Type	float, (0, 1)
Default	0.05
Effective	After restart system

Engine Layer

- rpc_address

Name	rpc_address
Description	The jdbc service listens on the address.
Type	String
Default	"0.0.0.0"
Effective	After restart system

- rpc_port

Name	rpc_port
Description	The jdbc service listens on the port. Please confirm that the port is not a system reserved port and is not occupied.
Type	Short Int : [0,65535]
Default	6667
Effective	After restart system

- time_zone

Name	time_zone
Description	The time zone in which the server is located, the default is Beijing time (+8)
Type	Time Zone String
Default	+08:00
Effective	Trigger

- base_dir

Name	base_dir
Description	The IoTDB system folder. It is recommended to use an absolute path.
Type	String
Default	data
Effective	After restart system

- data_dirs

Name	data_dirs
Description	The directories of data files. Multiple directories are separated by comma. The starting directory of the relative path is related to the operating system. It is recommended to use an absolute path. If the path does not exist, the system will automatically create it.
Type	String[]
Default	data/data
Effective	Trigger

- wal_dir

Name	wal_dir
Description	Write Ahead Log storage path. It is recommended to use an absolute path.
Type	String
Default	data/wal
Effective	After restart system

- enable_wal

Name	enable_wal
Description	Whether to enable the pre-write log. The default value is true(enabled), and false means closed.
Type	Bool
Default	true

Effective	Trigger

- enable_mem_control

Name	enable_mem_control
Description	enable memory control to avoid OOM
Type	Bool
Default	true
Effective	After restart system

- memtable_size_threshold

Name	memtable_size_threshold
Description	max memtable size
Type	Long
Default	1073741824
Effective	when enable_mem_control is false & After restart system

- avg_series_point_number_threshold

Name	avg_series_point_number_threshold
Description	max average number of point of each series in memtable
Type	Int32
Default	10000
Effective	After restart system

- tsfile_size_threshold

Name	tsfile_size_threshold
Description	max tsfile size
Type	Long
Default	536870912
Effective	After restart system

- enable_partition

Name	enable_partition
Description	Whether enable time partition for data, if disabled, all data belongs to partition 0
Type	Bool
Default	false
Effective	Only allowed to be modified in first start up

- partition_interval

Name	partition_interval
Description	Time range for dividing storage group, time series data will be divided into groups by this time range
Type	Int64
Default	604800
Effective	Only allowed to be modified in first start up

- concurrent_writing_time_partition

Name	concurrent_writing_time_partition
Description	This config decides how many time partitions in a storage group can be inserted concurrently For example, your partitionInterval is 86400 and you want to insert data in 5 different days,
Type	Int32
Default	1
Effective	After restart system

- multi_dir_strategy

Name	multi_dir_strategy
Description	IoTDB’s strategy for selecting directories for TsFile in tsfile_dir. You can use a simple class name or a full name of the class. The system provides the following three strategies: 1. SequenceStrategy: IoTDB selects the directory from tsfile_dir in order, traverses all the directories in tsfile_dir in turn, and keeps counting; 2. MaxDiskUsableSpaceFirstStrategy: IoTDB first selects the directory with the largest free disk space in tsfile_dir; 3. MinFolderOccupiedSpaceFirstStrategy: IoTDB prefers the directory with the least space used in tsfile_dir; 4. UserDfineStrategyPackage (user-defined policy) You can complete a user-defined policy in the following ways: 1. Inherit the cn.edu.tsinghua.iotdb.conf.directories.strategy.DirectoryStrategy class and implement its own Strategy method; 2. Fill in the configuration class with the full class name of the implemented class (package name plus class name, UserDfineStrategyPackage); 3. Add the jar file to the project.
Type	String
Default	MaxDiskUsableSpaceFirstStrategy
Effective	Trigger

- tsfile_size_threshold

Name	tsfile_size_threshold
Description	When a TsFile size on the disk exceeds this threshold, the TsFile is closed and open a new TsFile to accept data writes. The unit is byte

	and the default value is 2G.
Type	Int64
Default	536870912
Effective	After restart system

- tag_attribute_total_size

Name	tag_attribute_total_size
Description	The maximum persistence size of tags and attributes of each time series.
Type	Int32
Default	700
Effective	Only allowed to be modified in first start up

- enable_partial_insert

Name	enable_partial_insert
Description	Whether continue to write other measurements if some measurements are failed in one insertion.
Type	Bool
Default	true
Effective	After restart system

- mtree_snapshot_interval

Name	mtree_snapshot_interval
Description	The least interval line numbers of mlog.txt when creating a checkpoint and saving snapshot of MTree. Unit: line numbers
Type	Int32
Default	100000
Effective	After restart system

- flush_wal_threshold

Name	flush_wal_threshold
Description	After the WAL reaches this value, it is flushed to disk, and it is possible to lose at most flush_wal_threshold operations.
Type	Int32
Default	10000
Effective	Trigger

- force_wal_period_in_ms

Name	force_wal_period_in_ms

Description	The period during which the log is periodically forced to flush to disk(in milliseconds)
Type	Int32
Default	10
Effective	Trigger

- fetch_size

Name	fetch_size
Description	The amount of data read each time in batch (the number of data strips, that is, the number of different timestamps.)
Type	Int32
Default	10000
Effective	After restart system

- merge_concurrent_threads

Name	merge_concurrent_threads
Description	THe max threads which can be used when unsequence data is merged. The larger it is, the more IO and CPU cost. The smaller the value, the more the disk is occupied when the unsequence data is too large, the reading will be slower.
Type	Int32
Default	0
Effective	After restart system

- enable_stat_monitor

Name	enable_stat_monitor
Description	Whether to enable background statistics
Type	Boolean
Default	false
Effective	After restart system

- back_loop_period_in_second

Name	back_loop_period_in_second
Description	The frequency at which the system statistic module triggers(in seconds).
Type	Int32
Default	5
Effective	After restart system

- concurrent_flush_thread

Name	concurrent_flush_thread
Description	The thread number used to perform the operation when IoTDB writes data in memory to disk. If the value is less than or equal to 0, then the number of CPU cores installed on the machine is used. The default is 0.
Type	Int32
Default	0
Effective	After restart system

- stat_monitor_detect_freq_in_second

Name	stat_monitor_detect_freq_in_second
Description	The time interval which the system check whether the current record statistic time range exceeds stat_monitor_retain_interval every time (in seconds) and perform regular cleaning
Type	Int32
Default	600
Effective	After restart system

- stat_monitor_retain_interval_in_second

Name	stat_monitor_retain_interval_in_second
Description	The retention time of system statistics data(in seconds). Statistics data over the retention time range will be cleaned regularly.
Type	Int32
Default	600
Effective	After restart system

- tsfile_storage_fs

Name	tsfile_storage_fs
Description	The storage file system of Tsfile and related data files. Currently LOCAL file system and HDFS are supported.
Type	String
Default	LOCAL
Effective	Only allowed to be modified in first start up

- core_site_path

Name	core_site_path
Description	Absolute file path of core-site.xml if Tsfile and related data files are stored in HDFS.
Type	String
Default	/etc/hadoop/conf/core-site.xml
Effective	After restart system

- hdfs_site_path

Name	hdfs_site_path
Description	Absolute file path of hdfs-site.xml if Tsfile and related data files are stored in HDFS.
Type	String
Default	/etc/hadoop/conf/hdfs-site.xml
Effective	After restart system

- hdfs_ip

Name	hdfs_ip
Description	IP of HDFS if Tsfile and related data files are stored in HDFS. **If there are more than one hdfs_ip in configuration, Hadoop HA is used.**
Type	String
Default	localhost
Effective	After restart system

- hdfs_port

Name	hdfs_port
Description	Port of HDFS if Tsfile and related data files are stored in HDFS
Type	String
Default	9000
Effective	After restart system

- dfs_nameservices

Name	hdfs_nameservices
Description	Nameservices of HDFS HA if using Hadoop HA
Type	String
Default	hdfsnamespace
Effective	After restart system

- dfs_ha_namenodes

Name	hdfs_ha_namenodes
Description	Namenodes under DFS nameservices of HDFS HA if using Hadoop HA
Type	String
Default	nn1,nn2
Effective	After restart system

- dfs_ha_automatic_failover_enabled

Name	dfs_ha_automatic_failover_enabled
Description	Whether using automatic failover if using Hadoop HA
Type	Boolean
Default	true
Effective	After restart system

- dfs_client_failover_proxy_provider

Name	dfs_client_failover_proxy_provider
Description	Proxy provider if using Hadoop HA and enabling automatic failover
Type	String
Default	org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvic
Effective	After restart system

- hdfs_use_kerberos

Name	hdfs_use_kerberos
Description	Whether use kerberos to authenticate hdfs
Type	String
Default	false
Effective	After restart system

- kerberos_keytab_file_path

Name	kerberos_keytab_file_path
Description	Full path of kerberos keytab file
Type	String
Default	/path
Effective	After restart system

- kerberos_principal

Name	kerberos_principal
Description	Kerberos pricipal
Type	String
Default	your principal
Effective	After restart system

- authorizer_provider_class

Name	authorizer_provider_class

Description	the class name of the authorization service
Type	String
Default	org.apache.iotdb.db.auth.authorizer.LocalFileAuthorizer
Effective	After restart system
Other available values	org.apache.iotdb.db.auth.authorizer.OpenIdAuthorizer

- openID_url

Name	openID_url
Description	the openID server if OpenIdAuthorizer is enabled
Type	String (a http url)
Default	no
Effective	After restart system

Automatic Schema Creation and Type Inference

- enable_auto_create_schema

Name	enable_auto_create_schema
Description	whether auto create the time series when a non-existed time series data comes
Type	true or false
Default	true
Effective	After restart system

- default_storage_group_level

Name	default_storage_group_level
Description	Storage group level when creating schema automatically is enabled. For example, if we receives a data point from root.sg0.d1.s2, we will set root.sg0 as the storage group if storage group level is 1. (root is level 0)
Type	integer
Default	1
Effective	After restart system

- boolean_string_infer_type

Name	boolean_string_infer_type
Description	To which type the values “true” and “false” should be reslved
Type	BOOLEAN or TEXT
Default	BOOLEAN

Effective	After restart system

- integer_string_infer_type

Name	integer_string_infer_type
Description	To which type an integer string like “67” in a query should be resolved
Type	INT32, INT64, DOUBLE, FLOAT or TEXT
Default	DOUBLE
Effective	After restart system

- nan_string_infer_type

Name	nan_string_infer_type
Description	To which type the value NaN in a query should be resolved
Type	DOUBLE, FLOAT or TEXT
Default	FLOAT
Effective	After restart system

- floating_string_infer_type

Name	floating_string_infer_type
Description	To which type a floating number string like “6.7” in a query should be resolved
Type	DOUBLE, FLOAT or TEXT
Default	FLOAT
Effective	After restart system

Enable GC log

GC log is off by default. For performance tuning, you may want to collect the GC info.

To enable GC log, just add a parameter “printgc” when you start the server.

```
1. nohup sbin/start-server.sh printgc >/dev/null 2>&1 &
```

Or

```
1. sbin\start-server.bat printgc
```

GC log is stored at `IOTDB_HOME/logs/gc.log`. There will be at most 10 gc.log.* files and each one can reach to 10MB.

Docker Image

Now a Dockerfile has been written at docker/src/main/Dockerfile.

1. You can build a docker image by:

```
$ docker build -t iotdb:base git://github.com/apache/iotdb#master:docker
```

Or:

```
$ git clone https://github.com/apache/iotdb
$ cd iotdb
$ cd docker
$ docker build -t iotdb:base .
```

Once the docker image has been built locally (the tag is iotdb:base in this example), you are almost done!

1. create docker volume for data files and logs:

```
$ docker volume create mydata
$ docker volume create mylogs
```

1. run a docker container:

```
$ docker run -p 6667:6667 -v mydata:/iotdb/data -v mylogs:/iotdb/logs -d iotdb:base /iotdb/bin/start-server.sh
```

If success, you can run `docker ps`, and get something like the following:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
2a68b6944cb5        iotdb:base          "/iotdb/bin/start-se…"   4 minutes ago       Up 5 minutes
0.0.0.0:6667->6667/tcp              laughing_meitner
```

You can use the above command to get the container ID:

```
$ docker container ls
```

suppose the ID is <C_ID>.

And get the docker IP by:

```
$ docker inspect --format='{{.NetworkSettings.IPAddress}}' <C_ID>
```

suppose the IP is <C_IP>.

1. If you just want to have a try by using iotdb-cli, you can:

```
$ docker exec -it /bin/bash  <C_ID>
$ (now you have enter the container): /cli/sbin/start-cli.sh -h localhost -p 6667 -u root -pw root
```

Or, run a new docker container as the client:

```
$ docker run -it iotdb:base /cli/sbin/start-cli.sh -h <C_IP> -p 6667 -u root -pw root
```

Or, if you have a iotdb-cli locally (e.g., you have compiled the source code by `mvn package`), and suppose your work_dir is cli/bin, then you can just run:

```
$ start-cli.sh -h localhost -p 6667 -u root -pw root
```

1. If you want to write codes to insert data and query data, please add the following dependence:

```
        <dependency>
            <groupId>org.apache.iotdb</groupId>
            <artifactId>iotdb-jdbc</artifactId>
            <version>0.11.1</version>
        </dependency>
```

Some examples about how to use IoTDB with IoTDB-JDBC can be found at: https://github.com/apache/iotdb/tree/master/example/jdbc/src/main/java/org/apache/iotdb

1. Now enjoy it!

- Command Line Interface
- Native API
- JDBC
- Other Languages
- TsFile API
- MQTT
- Status Codes

Command Line Interface(CLI)

Outline

- Command Line Interface(CLI)
  - Running Cli/Shell
  - Cli/Shell Parameters
  - Cli/shell tool with -e parameter

IoTDB provides Cli/shell tools for users to interact with IoTDB server in command lines. This document shows how Cli/shell tool works and the meaning of its parameters.

> Note: In this document, $IOTDB_HOME represents the path of the IoTDB installation directory.

Build cli from source code

Under the root path of iotdb:

```
> mvn clean package -pl cli -am -DskipTests
```

After build, the IoTDB cli will be in the folder “cli/target/iotdb-cli-{project.version}”.

Running Cli/Shell

After installation, there is a default user in IoTDB: `root`, and the default password is `root`. Users can use this username to try IoTDB Cli/Shell tool. The cli startup script is the `start-cli` file under the $IOTDB_HOME/bin folder. When starting the script, you need to specify the IP and PORT. (Make sure the IoTDB server is running properly when you use Cli/Shell tool to connect it.)

Here is an example where the server is started locally and the user has not changed the running port. The default rpc port is 6667
If you need to connect to the remote server or changes the rpc port number of the server running, set the specific IP and RPC PORT at -h and -p.
You also can set your own environment variable at the front of the start script (“/sbin/start-cli.sh” for linux and “/sbin/start-cli.bat” for windows)

The Linux and MacOS system startup commands are as follows:

```
  Shell > sbin/start-cli.sh -h 127.0.0.1 -p 6667 -u root -pw root
```

The Windows system startup commands are as follows:

```
Shell > sbin\start-cli.bat -h 127.0.0.1 -p 6667 -u root -pw root
```

After using these commands, the cli can be started successfully. The successful status will be as follows:

```
 _____       _________  ______   ______
|_   _|     |  _   _  ||_   _ `.|_   _ \
  | |   .--.|_/ | | \_|  | | `. \ | |_) |
  | | / .'`\ \  | |      | |  | | |  __'.
 _| |_| \__. | _| |_    _| |_.' /_| |__) |
|_____|'.__.' |_____|  |______.'|_______/  version <version>


IoTDB> login successfully
IoTDB>
```

Enter `quit` or `exit` can exit Cli. The cli will shows `quit normally`

Cli/Shell Parameters

Parameter name	Parameter type	Required	Description	Example
-disableISO8601	No parameters	No	If this parameter is set, IoTDB will print the timestamp in digital form	-disableISO8601
-h < `host` >	string, no quotation marks	Yes	The IP address of the IoTDB server	-h 10.129.187.21
-help	No parameters	No	Print help information for IoTDB	-help
-p < `rpcPort` >	int	Yes	The rpc port number of the IoTDB server. IoTDB runs on rpc port 6667 by default	-p 6667
-pw < `password` >	string, no quotation marks	No	The password used for IoTDB to connect to the server. If no password is entered, IoTDB will ask for password in Cli command	-pw root
-u < `username` >	string, no quotation marks	Yes	User name used for IoTDB to connect the server	-u root
-maxPRC < `maxPrintRowCount` >	int	No	Set the maximum number of rows that IoTDB returns	-maxPRC 10
-e < `execute` >	string	No	manipulate IoTDB in batches without entering cli input	-e “show storage group”

			mode	

Following is a cli command which connects the host with IP 10.129.187.21, rpc port 6667, username “root”, password “root”, and prints the timestamp in digital form. The maximum number of lines displayed on the IoTDB command line is 10.

The Linux and MacOS system startup commands are as follows:

```
1.  Shell > sbin/start-cli.sh -h 10.129.187.21 -p 6667 -u root -pw root -disableISO8601 -maxPRC 10
```

The Windows system startup commands are as follows:

```
1.  Shell > sbin\start-cli.bat -h 10.129.187.21 -p 6667 -u root -pw root -disableISO8601 -maxPRC 10
```

Note on using the CLI with OpenID Connect Auth enabled on Server side

If OIDC is enabled on server side then no username / passwort is needed but a valid Access Token from the OIDC Provider. So as username you use the token and the password has to be empty, e.g.

```
1.  Shell > sbin/start-cli.sh -h 10.129.187.21 -p 6667 -u {my-access-token} -pw ""
```

How to get the token is dependent on your OpenID Connect setup and not covered here. In the simplest case you can get this via the command line with the `passwort-grant` . For example, if you use keycloack as OIDC and you have a realm with a client `iotdb` defined as public you could use the following `curl` command to fetch a token (replace all `{}` with appropriate values).

```
    curl -X POST "https://{your-keycloack-server}/auth/realms/{your-realm}/protocol/openid-connect/token" \
1.  Σ ✔ Σ 1613 Σ 11:09:38
2.   -H "Content-Type: application/x-www-form-urlencoded" \
3.   -d "username={username}" \
4.   -d "password={password}" \
5.   -d 'grant_type=password' \
6.   -d "client_id=iotdb"
```

The response looks something like

```
    {"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJxMS1XbTBvelE1TzBtUUg4LVNKYXAyWmNONE1tdWNXd25F
1.  gJpohy3JevOC44aJ4auzJR1RBj9LUbgcRinkBy0JLi6XXiYznSC2V485CSBHW3sseXn7pSXQADhnmGQrLfFGO5ZljmPO18eFJaimdjvgSChsrlSEm
```

The interesting part here is the access token with the key `access_token` . This has to be passed as username (with parameter `-u`) and empty password to the CLI.

Cli/shell tool with -e parameter

-e parameter is designed for the Cli/shell tool in the situation where you would like to manipulate IoTDB in batches through scripts. By using the -e parameter, you can operate IoTDB without entering the cli's input mode.

In order to avoid confusion between statements and other parameters, the current situation only supports the -e parameter as the last parameter.

The usage of -e parameter for Cli/shell is as follows:

The Linux and MacOS system commands:

```
Shell > sbin/start-cli.sh -h {host} -p {rpcPort} -u {user} -pw {password} -e {sql for iotdb}
```

The Windows system commands:

```
Shell > sbin\start-cli.bat -h {host} -p {rpcPort} -u {user} -pw {password} -e {sql for iotdb}
```

In the Windows environment, the SQL statement of the -e parameter needs to use `` to replace `" "`

In order to better explain the use of -e parameter, take following as an example(On linux system).

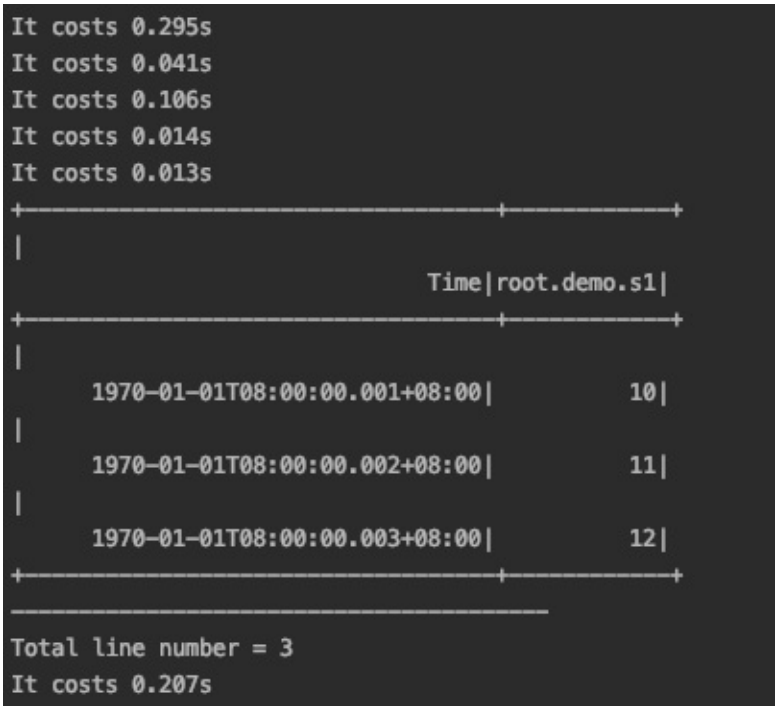
Suppose you want to create a storage group root.demo to a newly launched IoTDB, create a timeseries root.demo.s1 and insert three data points into it. With -e parameter, you could write a shell like this:

```
# !/bin/bash

host=127.0.0.1
rpcPort=6667
user=root
pass=root

./sbin/start-cli.sh -h ${host} -p ${rpcPort} -u ${user} -pw ${pass} -e "set storage group to root.demo"
./sbin/start-cli.sh -h ${host} -p ${rpcPort} -u ${user} -pw ${pass} -e "create timeseries root.demo.s1 WITH DATATYPE=INT32, ENCODING=RLE"
./sbin/start-cli.sh -h ${host} -p ${rpcPort} -u ${user} -pw ${pass} -e "insert into root.demo(timestamp,s1) values(1,10)"
./sbin/start-cli.sh -h ${host} -p ${rpcPort} -u ${user} -pw ${pass} -e "insert into root.demo(timestamp,s1) values(2,11)"
./sbin/start-cli.sh -h ${host} -p ${rpcPort} -u ${user} -pw ${pass} -e "insert into root.demo(timestamp,s1) values(3,12)"
./sbin/start-cli.sh -h ${host} -p ${rpcPort} -u ${user} -pw ${pass} -e "select s1 from root.demo"
```

The print results are shown in the figure, which are consistent with the cli and jdbc operations.



It should be noted that the use of the -e parameter in shell scripts requires attention to the escaping of special characters.

Programming - Native API

Dependencies

- JDK >= 1.8
- Maven >= 3.1

How to install in local maven repository

In root directory:

```
mvn clean install -pl session -am -DskipTests
```

Using IoTDB Native API with Maven

```
<dependencies>
    <dependency>
      <groupId>org.apache.iotdb</groupId>
      <artifactId>iotdb-session</artifactId>
      <version>0.11.1</version>
    </dependency>
</dependencies>
```

Native APIs

Here we show the commonly used interfaces and their parameters in the Native API:

- Initialize a Session

```
Session(String host, int rpcPort)

Session(String host, String rpcPort, String username, String password)

Session(String host, int rpcPort, String username, String password)
```

- Open a Session

```
Session.open()
```

- Close a Session

```
Session.close()
```

- Set storage group

```
void setStorageGroup(String storageGroupId)
```

- Delete one or several storage groups

```
void deleteStorageGroup(String storageGroup)
void deleteStorageGroups(List<String> storageGroups)
```

- Create one or multiple timeseries

```
void createTimeseries(String path, TSDataType dataType,
        TSEncoding encoding, CompressionType compressor, Map<String, String> props,
        Map<String, String> tags, Map<String, String> attributes, String measurementAlias)

void createMultiTimeseries(List<String> paths, List<TSDataType> dataTypes,
        List<TSEncoding> encodings, List<CompressionType> compressors,
        List<Map<String, String>> propsList, List<Map<String, String>> tagsList,
        List<Map<String, String>> attributesList, List<String> measurementAliasList)
```

- Delete one or several timeseries

```
void deleteTimeseries(String path)
void deleteTimeseries(List<String> paths)
```

- Delete data before or equal to a timestamp of one or several timeseries

```
void deleteData(String path, long time)
void deleteData(List<String> paths, long time)
```

- Insert a Record, which contains multiple measurement value of a device at a timestamp. Without type info the server has to do type inference, which may cost some time

```
void insertRecord(String deviceId, long time, List<String> measurements, List<String> values)
```

- Insert a Tablet, which is multiple rows of a device, each row has the same measurements

```
void insertTablet(Tablet tablet)
```

- Insert multiple Tablets

```
void insertTablets(Map<String, Tablet> tablet)
```

- Insert multiple Records. Without type info the server has to do type inference,

which may cost some time

```
void insertRecords(List<String> deviceIds, List<Long> times,
                   List<List<String>> measurementsList, List<List<String>> valuesList)
```

- Insert a Record, which contains multiple measurement value of a device at a timestamp. With type info the server has no need to do type inference, which leads a better performance

```
void insertRecord(String deviceId, long time, List<String> measurements,
     List<TSDataType> types, List<Object> values)
```

- Insert multiple Records. With type info the server has no need to do type inference, which leads a better performance

```
void insertRecords(List<String> deviceIds, List<Long> times,
      List<List<String>> measurementsList, List<List<TSDataType>> typesList,
      List<List<Object>> valuesList)
```

Native APIs for profiling network cost

- Test the network and client cost of insertRecords. This method NOT insert data into database and server just return after accept the request, this method should be used to test other time cost in client

```
void testInsertRecords(List<String> deviceIds, List<Long> times,
                List<List<String>> measurementsList, List<List<String>> valuesList)
```

or

```
void testInsertRecords(List<String> deviceIds, List<Long> times,
      List<List<String>> measurementsList, List<List<TSDataType>> typesList,
      List<List<Object>> valuesList)
```

- Test the network and client cost of insertRecord. This method NOT insert data into database and server just return after accept the request, this method should be used to test other time cost in client

```
void testInsertRecord(String deviceId, long time, List<String> measurements, List<String> values)
```

or

```
void testInsertRecord(String deviceId, long time, List<String> measurements,
      List<TSDataType> types, List<Object> values)
```

- Test the network and client cost of insertTablet. This method NOT insert data into database and server just return after accept the request, this method should be used to test other time cost in client

```
1. void testInsertTablet(Tablet tablet)
```

Sample code

To get more information of the following interfaces, please view session/src/main/java/org/apache/iotdb/session/Session.java

The sample code of using these interfaces is in example/session/src/main/java/org/apache/iotdb/SessionExample.java, which provides an example of how to open an IoTDB session, execute a batch insertion.

Session Pool for Native API

We provide a connection pool (`SessionPool) for Native API. Using the interface, you need to define the pool size.

If you can not get a session connection in 60 seconds, there is a warning log but the program will hang.

If a session has finished an operation, it will be put back to the pool automatically. If a session connection is broken, the session will be removed automatically and the pool will try to create a new session and redo the operation.

For query operations:

1. When using SessionPool to query data, the result set is `SessionDataSetWrapper`;
2. Given a `SessionDataSetWrapper`, if you have not scanned all the data in it and stop to use it, you have to call `SessionPool.closeResultSet(wrapper)` manually;
3. When you call `hasNext()` and `next()` of a `SessionDataSetWrapper` and there is an exception, then you have to call `SessionPool.closeResultSet(wrapper)` manually;
4. You can call `getColumnNames()` of `SessionDataSetWrapper` to get the column names of query result;

Examples: `session/src/test/java/org/apache/iotdb/session/pool/SessionPoolTest.java`

Or `example/session/src/main/java/org/apache/iotdb/SessionPoolExample.java`

0.9-0.10 Session Interface Updates

Significant chages are made in IoTDB session of version 0.10 compared to version 0.9. A large numbers of new interfaces are added, and some old interfaces have new names or parameters. Besides, all exceptions thrown by session interfaces are changed from

IoTDBSessionExeception to *IoTDBConnectionException* or *StatementExecutionExeception*. The detailed modifications are listed as follows.

Method name modifications

insert()

Insert a Record, which contains deviceId, timestamp of the record and multiple measurement values

```
1. void insert(String deviceId, long time, List<String> measurements, List<String> values)
```

The method name has been changed to insertRecord() in 0.10 version

```
1. void insertRecord(String deviceId, long time, List<String> measurements, List<String> values)
```

insertRowInBatch()

Insert multiple Records, which contains all the deviceIds, timestamps of the records and multiple measurement values

```
1. void insertRowInBatch(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList,
2.                       List<List<String>> valuesList)
```

The method name has been changed to insertRecords() in 0.10 version

```
1. void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList,
2.                    List<List<String>> valuesList)
```

insertBatch()

In 0.9, insertBatch is used for insertion in terms of “RowBatch” structure.

```
1. void insertBatch(RowBatch rowBatch)
```

As “Tablet” replaced “RowBatch” in 0.10, the name has been changed to insertTablet()

```
1. void insertTablet(Tablet tablet)
```

testInsertRow()

To test the responsiveness of insertRow()

```
1. void testInsertRow(String deviceId, long time, List<String> measurements, List<String> values)
```

The method name has been changed to testInsertRecord() in 0.10 version

```
void testInsertRecord(String deviceId, long time, List<String> measurements, List<String> values)
```

testInsertRowInBatch()

To test the responsiveness of insertRowInBatch()

```
void testInsertRowInBatch(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList,
                          List<List<String>> valuesList)
```

The method name has been changed to testInsertRecords() in 0.10 version

```
void testInsertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList,
                       List<List<String>> valuesList)
```

testInsertBatch

To test the responsiveness of insertBatch()

```
void testInsertBatch(RowBatch rowBatch)
```

The method name has been changed to testInsertTablet() in 0.10 version

```
void testInsertTablet(Tablet tablet)
```

New Interfaces

```
void open(boolean enableRPCCompression)
```

Open a session, with a parameter to specify whether to enable RPC compression. Please pay attention that this RPC compression status of client must comply with the status of IoTDB server

```
void insertRecord(String deviceId, long time, List<String> measurements,
      List<TSDataType> types, List<Object> values)
```

Insert one record, in a way that user has to provide the type information of each measurement, which is different from the original insertRecord() interface. The values should be provided in their primitive types. This interface is more proficient than the one without type parameters.

```
void insertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList,
                   List<List<TSDataType>> typesList, List<List<Object>> valuesList)
```

Insert multiple records with type parameters. This interface is more proficient than the one without type parameters.

```
1. void insertTablet(Tablet tablet, boolean sorted)
```

An additional insertTablet() interface that providing a “sorted” parameter indicating if the tablet is in order. A sorted tablet may accelerate the insertion process.

```
1. void insertTablets(Map<String, Tablet> tablets)
```

A new insertTablets() for inserting multiple tablets.

```
1. void insertTablets(Map<String, Tablet> tablets, boolean sorted)
```

insertTablets() with an additional “sorted” parameter.

```
1. void testInsertRecord(String deviceId, long time, List<String> measurements, List<TSDataType> types,
2.                       List<Object> values)
3. void testInsertRecords(List<String> deviceIds, List<Long> times, List<List<String>> measurementsList,
4.                       List<List<TSDataType>> typesList, List<List<Object>> valuesList)
5. void testInsertTablet(Tablet tablet, boolean sorted)
6. void testInsertTablets(Map<String, Tablet> tablets)
7. void testInsertTablets(Map<String, Tablet> tablets, boolean sorted)
```

The above interfaces are newly added to test responsiveness of new insert interfaces.

```
1. void createTimeseries(String path, TSDataType dataType, TSEncoding encoding, CompressionType compressor,
2.                       Map<String, String> props, Map<String, String> tags, Map<String, String> attributes,
3.                       String measurementAlias)
```

Create a timeseries with path, datatype, encoding and compression. Additionally, users can provide props, tags, attributes and measurementAlias。

```
1. void createMultiTimeseries(List<String> paths, List<TSDataType> dataTypes, List<TSEncoding> encodings,
2.                            List<CompressionType> compressors, List<Map<String, String>> propsList,
3.                            List<Map<String, String>> tagsList, List<Map<String, String>> attributesList,
4.                            List<String> measurementAliasList)
```

Create multiple timeseries with a single method. Users can provide props, tags, attributes and measurementAlias as well for detailed timeseries information.

```
1. boolean checkTimeseriesExists(String path)
```

Add a method to check whether the specific timeseries exists.

Programming - JDBC

Usage

Dependencies

- JDK >= 1.8
- Maven >= 3.1

Package only JDBC projects

Execute the following command in the root directory:

```
mvn clean package -pl jdbc -am -DskipTests
```

Install in local maven repository

In root directory:

```
mvn clean install -pl jdbc -am -Dmaven.test.skip=true
```

Use IoTDB JDBC with Maven

```
<dependencies>
    <dependency>
      <groupId>org.apache.iotdb</groupId>
      <artifactId>iotdb-jdbc</artifactId>
      <version>0.11.1</version>
    </dependency>
</dependencies>
```

Examples

This chapter provides an example of how to open a database connection, execute a SQL query, and display the results.

It requires including the packages containing the JDBC classes needed for database programming.

NOTE: For faster insertion, the insertTablet() in Session is recommended.

```java
import java.sql.*;
import org.apache.iotdb.jdbc.IoTDBSQLException;

public class JDBCExample {
  /**
   * Before executing a SQL statement with a Statement object, you need to create a Statement object using the createStatement() method of the Connection object.
   * After creating a Statement object, you can use its execute() method to execute a SQL statement
   * Finally, remember to close the 'statement' and 'connection' objects by using their close() method
   * For statements with query results, we can use the getResultSet() method of the Statement object to get the result set.
   */
  public static void main(String[] args) throws SQLException {
    Connection connection = getConnection();
    if (connection == null) {
      System.out.println("get connection defeat");
      return;
    }
    Statement statement = connection.createStatement();
    //Create storage group
    try {
      statement.execute("SET STORAGE GROUP TO root.demo");
    }catch (IoTDBSQLException e){
      System.out.println(e.getMessage());
    }


    //Show storage group
    statement.execute("SHOW STORAGE GROUP");
    outputResult(statement.getResultSet());

    //Create time series
    //Different data type has different encoding methods. Here use INT32 as an example
    try {
      statement.execute("CREATE TIMESERIES root.demo.s0 WITH DATATYPE=INT32,ENCODING=RLE;");
    }catch (IoTDBSQLException e){
      System.out.println(e.getMessage());
    }
    //Show time series
    statement.execute("SHOW TIMESERIES root.demo");
    outputResult(statement.getResultSet());
    //Show devices
    statement.execute("SHOW DEVICES");
    outputResult(statement.getResultSet());
    //Count time series
    statement.execute("COUNT TIMESERIES root");
    outputResult(statement.getResultSet());
    //Count nodes at the given level
    statement.execute("COUNT NODES root LEVEL=3");
    outputResult(statement.getResultSet());
    //Count timeseries group by each node at the given level
    statement.execute("COUNT TIMESERIES root GROUP BY LEVEL=3");
    outputResult(statement.getResultSet());
```

```
52.
53.
54.     //Execute insert statements in batch
55.     statement.addBatch("insert into root.demo(timestamp,s0) values(1,1);");
56.     statement.addBatch("insert into root.demo(timestamp,s0) values(1,1);");
57.     statement.addBatch("insert into root.demo(timestamp,s0) values(2,15);");
58.     statement.addBatch("insert into root.demo(timestamp,s0) values(2,17);");
59.     statement.addBatch("insert into root.demo(timestamp,s0) values(4,12);");
60.     statement.executeBatch();
61.     statement.clearBatch();
62.
63.     //Full query statement
64.     String sql = "select * from root.demo";
65.     ResultSet resultSet = statement.executeQuery(sql);
66.     System.out.println("sql: " + sql);
67.     outputResult(resultSet);
68.
69.     //Exact query statement
70.     sql = "select s0 from root.demo where time = 4;";
71.     resultSet= statement.executeQuery(sql);
72.     System.out.println("sql: " + sql);
73.     outputResult(resultSet);
74.
75.     //Time range query
76.     sql = "select s0 from root.demo where time >= 2 and time < 5;";
77.     resultSet = statement.executeQuery(sql);
78.     System.out.println("sql: " + sql);
79.     outputResult(resultSet);
80.
81.     //Aggregate query
82.     sql = "select count(s0) from root.demo;";
83.     resultSet = statement.executeQuery(sql);
84.     System.out.println("sql: " + sql);
85.     outputResult(resultSet);
86.
87.     //Delete time series
88.     statement.execute("delete timeseries root.demo.s0");
89.
90.     //close connection
91.     statement.close();
92.     connection.close();
93.   }
94.
95.   public static Connection getConnection() {
96.     // JDBC driver name and database URL
97.     String driver = "org.apache.iotdb.jdbc.IoTDBDriver";
98.     String url = "jdbc:iotdb://127.0.0.1:6667/";
99.
100.    // Database credentials
101.    String username = "root";
102.    String password = "root";
103.
104.    Connection connection = null;
```

```
105.     try {
106.       Class.forName(driver);
107.       connection = DriverManager.getConnection(url, username, password);
108.     } catch (ClassNotFoundException e) {
109.       e.printStackTrace();
110.     } catch (SQLException e) {
111.       e.printStackTrace();
112.     }
113.     return connection;
114.   }
115.
116.   /**
117.    * This is an example of outputting the results in the ResultSet
118.    */
119.   private static void outputResult(ResultSet resultSet) throws SQLException {
120.     if (resultSet != null) {
121.       System.out.println("--------------------------");
122.       final ResultSetMetaData metaData = resultSet.getMetaData();
123.       final int columnCount = metaData.getColumnCount();
124.       for (int i = 0; i < columnCount; i++) {
125.         System.out.print(metaData.getColumnLabel(i + 1) + " ");
126.       }
127.       System.out.println();
128.       while (resultSet.next()) {
129.         for (int i = 1; ; i++) {
130.           System.out.print(resultSet.getString(i));
131.           if (i < columnCount) {
132.             System.out.print(", ");
133.           } else {
134.             System.out.println();
135.             break;
136.           }
137.         }
138.       }
139.       System.out.println("--------------------------\n");
140.     }
141.   }
142. }
```

Programming - Other Languages

Python API

1. Introduction

This is an example of how to connect to IoTDB with python, using the thrift rpc interfaces. Things will be a bit different on Linux or Windows, we will introduce how to operate on the two systems separately.

2. Prerequisites

python3.7 or later is preferred.

You have to install Thrift (0.11.1 or later) to compile our thrift file into python code. Below is the official tutorial of installation:

```
1. http://thrift.apache.org/docs/install/
```

3. How to Get the Python Library

Option 1: pip install

You can find the Apache IoTDB Python Client API package on https://pypi.org/project/apache-iotdb/.

The download command is:

```
1. pip install apache-iotdb
```

Option 2: use the compile script we provided

If you have added Thrift executable into your path, you may just run `client-py/compile.sh` or `client-py\compile.bat`, otherwise, modify it to set variable `THRIFT_EXE` to point to your executable. This will generate thrift sources under folder `target`, you can add it to your `PYTHONPATH` so that you will be able to use the library in your code. Note that the scripts locate the thrift source file by relative path, so if you move the scripts else where, they are no longer valid.

Option 3: basic usage of thrift

Optionally, if you know the basic usage of thrift, download the thrift source file in `thrift\src\main\thrift\rpc.thrift`, and simply use `thrift -gen py -out ./target/iotdb rpc.thrift` to

generate the python library.

4. Use Example

We provided an example of how to use the thrift library to connect to IoTDB in `client-py/src /client_example.py` , please read it carefully before you write your own code.

Programming - TsFile API

TsFile is a file format of Time Series used in IoTDB. This session introduces the usage of this file format.

TsFile library Installation

There are two ways to use TsFile in your own project.

- Use as jars:
    - Compile the source codes and build to jars

```
1. git clone https://github.com/apache/iotdb.git
2. cd tsfile/
3. mvn clean package -Dmaven.test.skip=true
```

    Then, all the jars are in folder named `target/` . Import `target/tsfile-0.11.1-jar-with-dependencies.jar` to your project.

- Use as a maven dependency:

  Compile source codes and deploy to your local repository in three steps:

    - Get the source codes

```
1. git clone https://github.com/apache/iotdb.git
```

    - Compile the source codes and deploy

```
1. cd tsfile/
2. mvn clean install -Dmaven.test.skip=true
```

    - add dependencies into your project:

```
1.  <dependency>
2.    <groupId>org.apache.iotdb</groupId>
3.    <artifactId>tsfile</artifactId>
4.    <version>0.11.1</version>
5.  </dependency>
```

  Or, you can download the dependencies from official Maven repository:

    - First, find your maven `settings.xml` on path: `${username}\.m2\settings.xml` , add this `<profile>` to `<profiles>` :

```
<profile>
      <id>allow-snapshots</id>
         <activation><activeByDefault>true</activeByDefault></activation>
      <repositories>
        <repository>
           <id>apache.snapshots</id>
           <name>Apache Development Snapshot Repository</name>
           <url>https://repository.apache.org/content/repositories/snapshots/</url>
           <releases>
               <enabled>false</enabled>
           </releases>
           <snapshots>
               <enabled>true</enabled>
           </snapshots>
         </repository>
      </repositories>
    </profile>
```

- Then add dependencies into your project:

```
<dependency>
  <groupId>org.apache.iotdb</groupId>
  <artifactId>tsfile</artifactId>
  <version>0.11.1</version>
</dependency>
```

TSFile Usage

This section demonstrates the detailed usages of TsFile.

Time-series Data

Time-series data is considered as a sequence of quadruples. A quadruple is defined as (device, measurement, time, value).

- **measurement**: A physical or formal measurement that a time-series data takes, e.g., the temperature of a city, the sales number of some goods or the speed of a train at different times. As a traditional sensor (like a thermometer) also takes a single measurement and produce a time-series, we will use measurement and sensor interchangeably below.
- **device**: A device refers to an entity that takes several measurements (producing multiple time-series), e.g., a running train monitors its speed, oil meter, miles it has run, current passengers each is conveyed to a time-series dataset.

Table 1 illustrates a set of time-series data. The set showed in the following table contains one device named “device_1” with three measurements named “sensor_1”, “sensor_2” and “sensor_3”.

device_1					
sensor_1		sensor_2		sensor_3	
time	value	time	value	time	value
1	1.2	1	20	2	50
3	1.4	2	20	4	51
5	1.1	3	21	6	52
7	1.8	4	20	8	53

A set of time-series data

One Line of Data: In many industrial applications, a device normally contains more than one sensor and these sensors may have values at the same timestamp, which is called one line of data.

Formally, one line of data consists of a `device_id`, a timestamp which indicates the milliseconds since January 1, 1970, 00:00:00, and several data pairs composed of `measurement_id` and corresponding `value`. All data pairs in one line belong to this `device_id` and have the same timestamp. If one of the `measurements` does not have a `value` in the `timestamp`, use a space instead(Actually, TsFile does not store null values). Its format is shown as follow:

```
1. device_id, timestamp, <measurement_id, value>...
```

An example is illustrated as follow. In this example, the data type of two measurements are `INT32`, `FLOAT` respectively.

```
1. device_1, 1490860659000, m1, 10, m2, 12.12
```

Write TsFile

Generate a TsFile File.

A TsFile is generated by the following three steps and the complete code is given in the section “Example for writing TsFile”.

- First, construct a `TsFileWriter` instance.

  Here are the available constructors:

  - Without pre-defined schema

  ```
  1. public TsFileWriter(File file) throws IOException
  ```

  - With pre-defined schema

  ```
  1. public TsFileWriter(File file, Schema schema) throws IOException
  ```

This one is for using the HDFS file system. `TsFileOutput` can be an instance of class `HDFSOutput` .

```
1. public TsFileWriter(TsFileOutput output, Schema schema) throws IOException
```

If you want to set some TSFile configuration on your own, you could use param `config` . For example:

```
1. TSFileConfig conf = new TSFileConfig();
2. conf.setTSFileStorageFs("HDFS");
3. TsFileWriter tsFileWriter = new TsFileWriter(file, schema, conf);
```

In this example, data files will be stored in HDFS, instead of local file system. If you'd like to store data files in local file system, you can use `conf.setTSFileStorageFs("LOCAL")` , which is also the default config.

You can also config the ip and rpc port of your HDFS by `config.setHdfsIp(...)` and `config.setHdfsPort(...)` . The default ip is `localhost` and default rpc port is `9000` .

Parameters:

- file : The TsFile to write
- schema : The file schemas, will be introduced in next part.
- config : The config of TsFile.

- Second, add measurements

Or you can make an instance of class `Schema` first and pass this to the constructor of class `TsFileWriter`

The class `Schema` contains a map whose key is the name of one measurement schema, and the value is the schema itself.

Here are the interfaces:

```
1. // Create an empty Schema or from an existing map
2. public Schema()
3. public Schema(Map<String, MeasurementSchema> measurements)
4. // Use this two interfaces to add measurements
5. public void registerMeasurement(MeasurementSchema descriptor)
6. public void registerMeasurements(Map<String, MeasurementSchema> measurements)
7. // Some useful getter and checker
8. public TSDataType getMeasurementDataType(String measurementId)
9. public MeasurementSchema getMeasurementSchema(String measurementId)
10. public Map<String, MeasurementSchema> getAllMeasurementSchema()
11. public boolean hasMeasurement(String measurementId)
```

You can always use the following interface in `TsFileWriter` class to add additional

measurements:

```
1. public void addMeasurement(MeasurementSchema measurementSchema) throws WriteProcessException
```

The class `MeasurementSchema` contains the information of one measurement, there are several constructors:

```
1. public MeasurementSchema(String measurementId, TSDataType type, TSEncoding encoding)
   public MeasurementSchema(String measurementId, TSDataType type, TSEncoding encoding, CompressionType
2. compressionType)
   public MeasurementSchema(String measurementId, TSDataType type, TSEncoding encoding, CompressionType
3. compressionType,
4. Map<String, String> props)
```

Parameters:

- measurementID: The name of this measurement, typically the name of the sensor.
- type: The data type, now support six types: `BOOLEAN`, `INT32`, `INT64`, `FLOAT`, `DOUBLE`, `TEXT`;
- encoding: The data encoding. See Chapter 2-3.
- compression: The data compression. Now supports `UNCOMPRESSED` and `SNAPPY`.
- props: Properties for special data types.Such as `max_point_number` for `FLOAT` and `DOUBLE`, `max_string_length` for `TEXT`. Use as string pairs into a map such as ("max_point_number", "3").

> **Notice:** Although one measurement name can be used in multiple deltaObjects, the properties cannot be changed. I.e. it's not allowed to add one measurement name for multiple times with different type or encoding. Here is a bad example:

```
1.   // The measurement "sensor_1" is float type
2.   addMeasurement(new MeasurementSchema("sensor_1", TSDataType.FLOAT, TSEncoding.RLE));
3.
4.   // This call will throw a WriteProcessException exception
5.   addMeasurement(new MeasurementSchema("sensor_1", TSDataType.INT32, TSEncoding.RLE));
```

- Third, insert and write data continually.

Use this interface to create a new `TSRecord` (a timestamp and device pair).

```
1. public TSRecord(long timestamp, String deviceId)
```

Then create a `DataPoint` (a measurement and value pair), and use the addTuple method to add the DataPoint to the correct TsRecord.

Use this method to write

```
1. public void write(TSRecord record) throws IOException, WriteProcessException
```

- Finally, call `close` to finish this writing process.

```
1. public void close() throws IOException
```

We are also able to write data into a closed TsFile.

- Use `ForceAppendTsFileWriter` to open a closed file.

```
1. public ForceAppendTsFileWriter(File file) throws IOException
```

- call `doTruncate` truncate the part of Metadata
- Then use `ForceAppendTsFileWriter` to construct a new `TsFileWriter`

```
1. public TsFileWriter(TsFileIOWriter fileWriter) throws IOException
```

Please note, we should redo the step of adding measurements before writing new data to the TsFile.

Example for writing a TsFile

You should install TsFile to your local maven repository.

```
1. mvn clean install -pl tsfile -am -DskipTests
```

You could write a TsFile by constructing **TSRecord** if you have the **non-aligned** (e.g. not all sensors contain values) time series data.

A more thorough example can be found at
`/example/tsfile/src/main/java/org/apache/iotdb/tsfile/TsFileWriteWithTSRecord.java`

You could write a TsFile by constructing **Tablet** if you have the **aligned** time series data.

A more thorough example can be found at
`/example/tsfile/src/main/java/org/apache/iotdb/tsfile/TsFileWriteWithTablet.java`

You could write data into a closed TsFile by using **ForceAppendTsFileWriter**.

A more thorough example can be found at
`/example/tsfile/src/main/java/org/apache/iotdb/tsfile/TsFileForceAppendWrite.java`

Interface for Reading TsFile

Before the Start

The set of time-series data in section “Time-series Data” is used here for a concrete introduction in this section. The set showed in the following table contains one deltaObject named “device_1” with three measurements named “sensor_1”, “sensor_2” and “sensor_3”. And the measurements has been simplified to do a simple illustration, which contains only 4 time-value pairs each.

device_1					
sensor_1		sensor_2		sensor_3	
time	value	time	value	time	value
1	1.2	1	20	2	50
3	1.4	2	20	4	51
5	1.1	3	21	6	52
7	1.8	4	20	8	53

A set of time-series data

Definition of Path

A path is a dot-separated string which uniquely identifies a time-series in TsFile, e.g., “root.area_1.device_1.sensor_1”. The last section “sensor_1” is called “measurementId” while the remaining parts “root.area_1.device_1” is called deviceId. As mentioned above, the same measurement in different devices has the same data type and encoding, and devices are also unique.

In read interfaces, The parameter `paths` indicates the measurements to be selected.

Path instance can be easily constructed through the class `Path`. For example:

```
1. Path p = new Path("device_1.sensor_1");
```

We will pass an ArrayList of paths for final query call to support multiple paths.

```
1. List<Path> paths = new ArrayList<Path>();
2. paths.add(new Path("device_1.sensor_1"));
3. paths.add(new Path("device_1.sensor_3"));
```

> **Notice:** When constructing a Path, the format of the parameter should be a dot-separated string, the last part will be recognized as measurementId while the remaining parts will be recognized as deviceId.

Definition of Filter

Usage Scenario

Filter is used in TsFile reading process to select data satisfying one or more given condition(s).

IExpression

The `IExpression` is a filter expression interface and it will be passed to our final query call. We create one or more filter expressions and may use binary filter operators to link them to our final expression.

- **Create a Filter Expression**

  There are two types of filters.

  - TimeFilter: A filter for `time` in time-series data.

    ```
    1. IExpression timeFilterExpr = new GlobalTimeExpression(TimeFilter);
    ```

    Use the following relationships to get a `TimeFilter` object (value is a long int variable).

Relationship	Description
TimeFilter.eq(value)	Choose the time equal to the value
TimeFilter.lt(value)	Choose the time less than the value
TimeFilter.gt(value)	Choose the time greater than the value
TimeFilter.ltEq(value)	Choose the time less than or equal to the value
TimeFilter.gtEq(value)	Choose the time greater than or equal to the value
TimeFilter.notEq(value)	Choose the time not equal to the value
TimeFilter.not(TimeFilter)	Choose the time not satisfy another TimeFilter

  - ValueFilter: A filter for `value` in time-series data.

    ```
    1. IExpression valueFilterExpr = new SingleSeriesExpression(Path, ValueFilter);
    ```

    The usage of `ValueFilter` is the same as using `TimeFilter`, just to make sure that the type of the value equal to the measurement's(defined in the path).

- **Binary Filter Operators**

  Binary filter operators can be used to link two single expressions.

  - BinaryExpression.and(Expression, Expression): Choose the value satisfy for both expressions.
  - BinaryExpression.or(Expression, Expression): Choose the value satisfy for at least one expression.

Filter Expression Examples

- **TimeFilterExpression Examples**

```
IExpression timeFilterExpr = new GlobalTimeExpression(TimeFilter.eq(15)); // series time = 15
```

```
IExpression timeFilterExpr = new GlobalTimeExpression(TimeFilter.ltEq(15)); // series time <= 15
```

```
IExpression timeFilterExpr = new GlobalTimeExpression(TimeFilter.lt(15)); // series time < 15
```

```
IExpression timeFilterExpr = new GlobalTimeExpression(TimeFilter.gtEq(15)); // series time >= 15
```

```
IExpression timeFilterExpr = new GlobalTimeExpression(TimeFilter.notEq(15)); // series time != 15
```

```
IExpression timeFilterExpr = BinaryExpression.and(new GlobalTimeExpression(TimeFilter.gtEq(15L)),
                                        new GlobalTimeExpression(TimeFilter.lt(25L))); // 15 <= series
time < 25
```

```
IExpression timeFilterExpr = BinaryExpression.or(new GlobalTimeExpression(TimeFilter.gtEq(15L)),
                                        new GlobalTimeExpression(TimeFilter.lt(25L))); // series time
>= 15 or series time < 25
```

Read Interface

First, we open the TsFile and get a `ReadOnlyTsFile` instance from a file path string `path` .

```
TsFileSequenceReader reader = new TsFileSequenceReader(path);

ReadOnlyTsFile readTsFile = new ReadOnlyTsFile(reader);
```

Next, we prepare the path array and query expression, then get final `QueryExpression` object by this interface:

```
QueryExpression queryExpression = QueryExpression.create(paths, statement);
```

The ReadOnlyTsFile class has two `query` method to perform a query.

- **Method 1**

```
public QueryDataSet query(QueryExpression queryExpression) throws IOException
```

- **Method 2**

```
public QueryDataSet query(QueryExpression queryExpression, long partitionStartOffset, long
partitionEndOffset) throws IOException
```

This method is designed for advanced applications such as the TsFile-Spark Connector.

- **params** : For method 2, two additional parameters are added to support partial query:
  - `partitionStartOffset` : start offset for a TsFile
  - `partitionEndOffset` : end offset for a TsFile

> **What is Partial Query ?**
>
> In some distributed file systems(e.g. HDFS), a file is split into severval parts which are called "Blocks" and stored in different nodes. Executing a query paralleled in each nodes involved makes better efficiency. Thus Partial Query is needed. Paritial Query only selects the results stored in the part split by `QueryConstant.PARTITION_START_OFFSET` and `QueryConstant.PARTITION_END_OFFSET` for a TsFile.

QueryDataset Interface

The query performed above will return a `QueryDataset` object.

Here's the useful interfaces for user.

- `bool hasNext();`

  Return true if this dataset still has elements.

- `List<Path> getPaths()`

  Get the paths in this data set.

- `List<TSDataType> getDataTypes();`

  Get the data types. The class TSDataType is an enum class, the value will be one of the following:

```
1. BOOLEAN,
2. INT32,
3. INT64,
4. FLOAT,
5. DOUBLE,
6. TEXT;
```

- `RowRecord next() throws IOException;`

  Get the next record.

  The class `RowRecord` consists of a `long` timestamp and a `List<Field>` for data in different sensors, we can use two getter methods to get them.

```
1. long getTimestamp();
2. List<Field> getFields();
```

To get data from one Field, use these methods:

```
1. TSDataType getDataType();
2. Object getObjectValue();
```

Example for reading an existing TsFile

You should install TsFile to your local maven repository.

A more thorough example with query statement can be found at `/tsfile/example/src/main/java/org/apache/iotdb/tsfile/TsFileRead.java`

```
1. package org.apache.iotdb.tsfile;
2. import java.io.IOException;
3. import java.util.ArrayList;
4. import org.apache.iotdb.tsfile.read.ReadOnlyTsFile;
5. import org.apache.iotdb.tsfile.read.TsFileSequenceReader;
6. import org.apache.iotdb.tsfile.read.common.Path;
7. import org.apache.iotdb.tsfile.read.expression.IExpression;
8. import org.apache.iotdb.tsfile.read.expression.QueryExpression;
9. import org.apache.iotdb.tsfile.read.expression.impl.BinaryExpression;
10. import org.apache.iotdb.tsfile.read.expression.impl.GlobalTimeExpression;
11. import org.apache.iotdb.tsfile.read.expression.impl.SingleSeriesExpression;
12. import org.apache.iotdb.tsfile.read.filter.TimeFilter;
13. import org.apache.iotdb.tsfile.read.filter.ValueFilter;
14. import org.apache.iotdb.tsfile.read.query.dataset.QueryDataSet;
15.
16. /**
17.  * The class is to show how to read TsFile file named "test.tsfile".
18.  * The TsFile file "test.tsfile" is generated from class TsFileWrite.
19.  * Run TsFileWrite to generate the test.tsfile first
20.  */
21. public class TsFileRead {
22.   private static void queryAndPrint(ArrayList<Path> paths, ReadOnlyTsFile readTsFile, IExpression statement)
23.           throws IOException {
24.     QueryExpression queryExpression = QueryExpression.create(paths, statement);
25.     QueryDataSet queryDataSet = readTsFile.query(queryExpression);
26.     while (queryDataSet.hasNext()) {
27.       System.out.println(queryDataSet.next());
28.     }
29.     System.out.println("------------");
30.   }
31.
32.   public static void main(String[] args) throws IOException {
33.
34.     // file path
35.     String path = "test.tsfile";
36.
37.     // create reader and get the readTsFile interface
38.     TsFileSequenceReader reader = new TsFileSequenceReader(path);
```

```
    ReadOnlyTsFile readTsFile = new ReadOnlyTsFile(reader);
    // use these paths(all sensors) for all the queries
    ArrayList<Path> paths = new ArrayList<>();
    paths.add(new Path("device_1.sensor_1"));
    paths.add(new Path("device_1.sensor_2"));
    paths.add(new Path("device_1.sensor_3"));

    // no query statement
    queryAndPrint(paths, readTsFile, null);

    //close the reader when you left
    reader.close();
  }
}
```

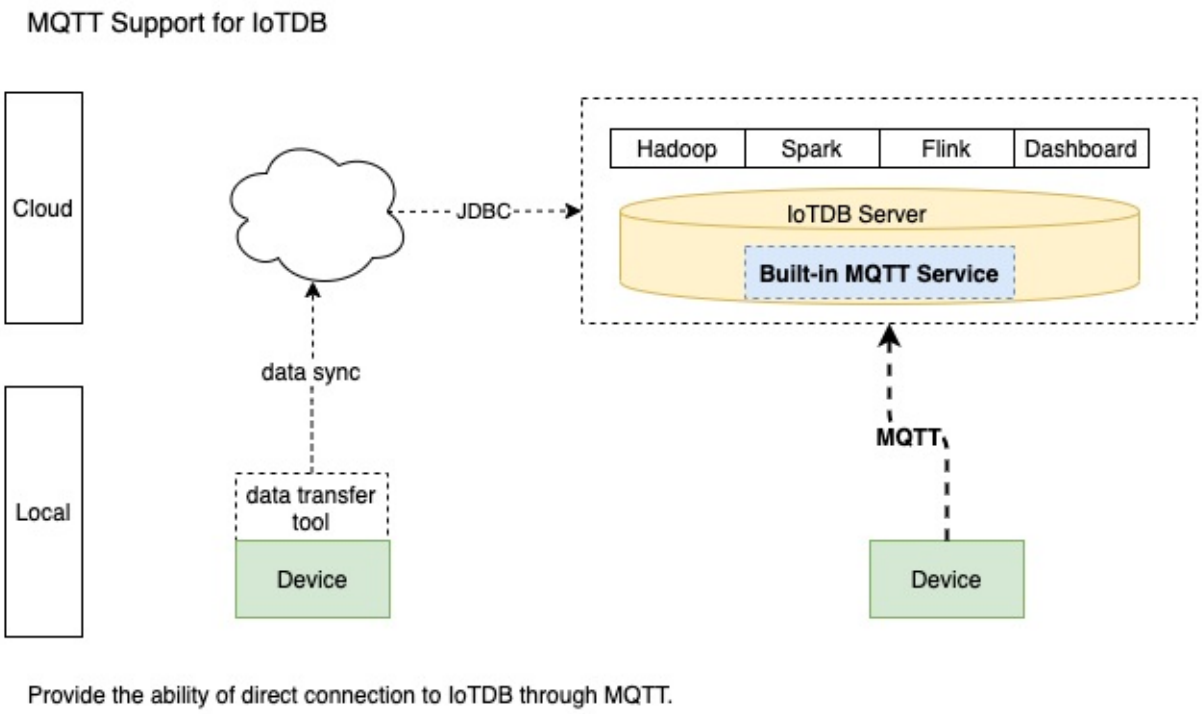
Change TsFile Configuration

```
TSFileConfig config = TSFileDescriptor.getInstance().getConfig();
config.setXXX();
```

MQTT Protocol

MQTT (opens new window) is a machine-to-machine (M2M)/“Internet of Things” connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.

IoTDB supports the MQTT v3.1(an OASIS Standard) protocol. IoTDB server includes a built-in MQTT service that allows remote devices send messages into IoTDB server directly.



Provide the ability of direct connection to IoTDB through MQTT.

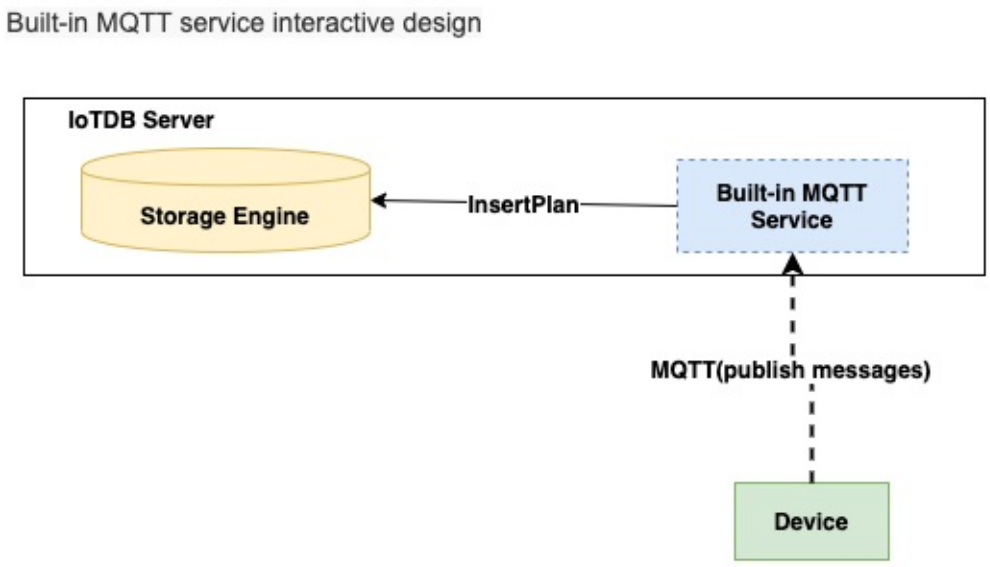
Built-in MQTT Service

The Built-in MQTT Service provide the ability of direct connection to IoTDB through MQTT. It listen the publish messages from MQTT clients and then write the data into storage immediately. The MQTT topic corresponds to IoTDB timeseries. The messages payload can be format to events by `PayloadFormatter` which loaded by java SPI, and the default implementation is `JSONPayloadFormatter` . The default `json` formatter support two json format, and the following is an MQTT message payload example:

```
{
    "device":"root.sg.d1",
    "timestamp":1586076045524,
    "measurements":["s1","s2"],
    "values":[0.530635,0.530635]
}
```

or

```
{
      "device":"root.sg.d1",
      "timestamps":[1586076045524,1586076065526],
      "measurements":["s1","s2"],
      "values":[[0.530635,0.530635], [0.530655,0.530695]]
  }
```



MQTT Configurations

The IoTDB MQTT service load configurations from `${IOTDB_HOME}/${IOTDB_CONF}/iotdb-engine.properties` by default.

Configurations are as follows:

NAME	DESCRIPTION	DEFAULT
enable_mqtt_service	whether to enable the mqtt service	false
mqtt_host	the mqtt service binding host	0.0.0.0
mqtt_port	the mqtt service binding port	1883
mqtt_handler_pool_size	the handler pool size for handing the mqtt messages	1
mqtt_payload_formatter	the mqtt message payload formatter	json
mqtt_max_message_size	the max mqtt message size in byte	1048576

Examples

The following is an example which a mqtt client send messages to IoTDB server.

```
        MQTT mqtt = new MQTT();
        mqtt.setHost("127.0.0.1", 1883);
        mqtt.setUserName("root");
        mqtt.setPassword("root");

        BlockingConnection connection = mqtt.blockingConnection();
        connection.connect();

        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            String payload = String.format("{\n" +
                    "\"device\":\"root.sg.d1\",\n" +
                    "\"timestamp\":%d,\n" +
                    "\"measurements\":[\"s1\"],\n" +
                    "\"values\":[%f]\n" +
                    "}", System.currentTimeMillis(), random.nextDouble());

            connection.publish("root.sg.d1.s1", payload.getBytes(), QoS.AT_LEAST_ONCE, false);
        }

        connection.disconnect();
    }
```

Status Codes

Status Code is introduced in the latest version. A sample solution as IoTDB requires registering the time series first before writing data is:

```
try {
    writeData();
} catch (SQLException e) {
  // the most case is that the time series does not exist
  if (e.getMessage().contains("exist")) {
      //However, using the content of the error message is not so efficient
      registerTimeSeries();
      //write data once again
      writeData();
  }
}
```

With Status Code, instead of writing codes like `if (e.getErrorMessage().contains("exist"))`, we can simply use `e.getErrorCode() == TSStatusCode.TIME_SERIES_NOT_EXIST_ERROR.getStatusCode()`.

Here is a list of Status Code and related message:

Status Code	Status Type	Meanings
200	SUCCESS_STATUS	
201	STILL_EXECUTING_STATUS	
202	INVALID_HANDLE_STATUS	
203	INCOMPATIBLE_VERSION	Incompatible version
298	NODE_DELETE_FAILED_ERROR	Failed while deleting node
299	ALIAS_ALREADY_EXIST_ERROR	Alias already exists
300	PATH_ALREADY_EXIST_ERROR	Path already exists
301	PATH_NOT_EXIST_ERROR	Path does not exist
302	UNSUPPORTED_FETCH_METADATA_OPERATION_ERROR	Unsupported fetch metadata operation
303	METADATA_ERROR	Meet error when dealing with metadata
305	OUT_OF_TTL_ERROR	Insertion time is less than TTL time bound
306	CONFIG_ADJUSTER	IoTDB system load is too large
307	MERGE_ERROR	Meet error while merging
308	SYSTEM_CHECK_ERROR	Meet error while system checking
309	SYNC_DEVICE_OWNER_CONFLICT_ERROR	Sync device owners conflict

310	SYNC_CONNECTION_EXCEPTION	Meet error while sync connecting
311	STORAGE_GROUP_PROCESSOR_ERROR	Storage group processor related error
312	STORAGE_GROUP_ERROR	Storage group related error
313	STORAGE_ENGINE_ERROR	Storage engine related error
314	TSFILE_PROCESSOR_ERROR	TsFile processor related error
315	PATH_ILLEGAL	Illegal path
316	LOAD_FILE_ERROR	Meet error while loading file
317	STORAGE_GROUP_NOT_READY	The storage group is in recovery mode, not ready fore accepting read/write operation
400	EXECUTE_STATEMENT_ERROR	Execute statement error
401	SQL_PARSE_ERROR	Meet error while parsing SQL
402	GENERATE_TIME_ZONE_ERROR	Meet error while generating time zone
403	SET_TIME_ZONE_ERROR	Meet error while setting time zone
404	NOT_STORAGE_GROUP_ERROR	Operating object is not a storage group
405	QUERY_NOT_ALLOWED	Query statements are not allowed error
406	AST_FORMAT_ERROR	AST format related error
407	LOGICAL_OPERATOR_ERROR	Logical operator related error
408	LOGICAL_OPTIMIZE_ERROR	Logical optimize related error
409	UNSUPPORTED_FILL_TYPE_ERROR	Unsupported fill type related error
410	PATH_ERROR	Path related error
411	QUERY_PROCESS_ERROR	Query process related error
412	WRITE_PROCESS_ERROR	Writing data related error
500	INTERNAL_SERVER_ERROR	Internal server error
501	CLOSE_OPERATION_ERROR	Meet error in close operation
502	READ_ONLY_SYSTEM_ERROR	Operating system is read only
503	DISK_SPACE_INSUFFICIENT_ERROR	Disk space is insufficient
504	START_UP_ERROR	Meet error while starting up
505	SHUT_DOWN_ERROR	Meet error while shutdown
506	MULTIPLE_ERROR	Meet error when executing

506	MULTIPLE_ERROR	multiple statements
600	WRONG_LOGIN_PASSWORD_ERROR	Username or password is wrong
601	NOT_LOGIN_ERROR	Has not logged in
602	NO_PERMISSION_ERROR	No permissions for this operation
603	UNINITIALIZED_AUTH_ERROR	Uninitialized authorizer

> All exceptions are refactored in latest version by extracting uniform message into exception classes. Different error codes are added to all exceptions. When an exception is caught and a higher-level exception is thrown, the error code will keep and pass so that users will know the detailed error reason. A base exception class “ProcessException” is also added to be extended by all exceptions.

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- Administration
- SQL Reference

DDL (Data Definition Language)

Create Storage Group

According to the storage model we can set up the corresponding storage group. The SQL statements for creating storage groups are as follows:

```
1. IoTDB > set storage group to root.ln
2. IoTDB > set storage group to root.sgcc
```

We can thus create two storage groups using the above two SQL statements.

It is worth noting that when the path itself or the parent/child layer of the path is already set as a storage group, the path is then not allowed to be set as a storage group. For example, it is not feasible to set `root.ln.wf01` as a storage group when two storage groups `root.ln` and `root.sgcc` exist. The system gives the corresponding error prompt as shown below:

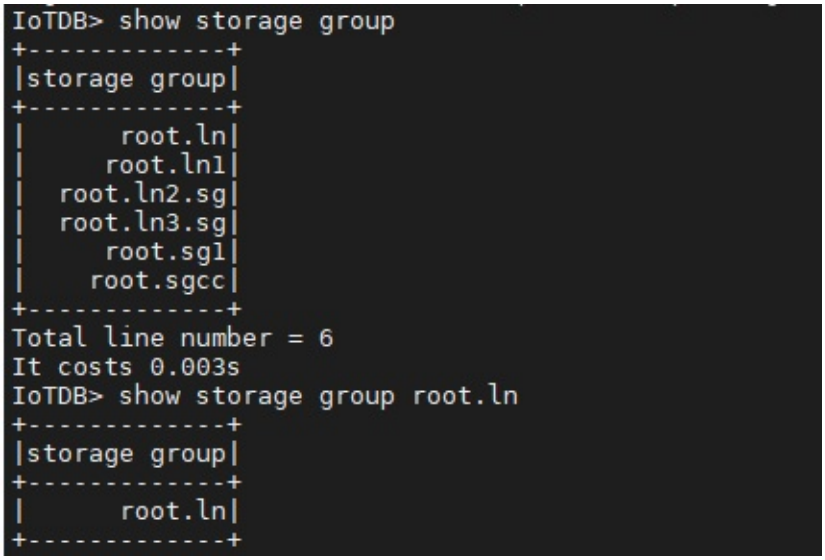
```
1. IoTDB> set storage group to root.ln.wf01
   Msg: org.apache.iotdb.exception.MetadataException: org.apache.iotdb.exception.MetadataException: The prefix of
2. root.ln.wf01 has been set to the storage group.
```

Show Storage Group

After the storage group is created, we can use the SHOW STORAGE GROUP statement and SHOW STORAGE GROUP <PrefixPath> to view the storage groups. The SQL statements are as follows:

```
1. IoTDB> show storage group
2. IoTDB> show storage group root.ln
```

The result is as follows:



Delete Storage Group

User can use the `DELETE STORAGE GROUP <PrefixPath>` statement to delete all storage groups under the prefixPath. Please note the data in the storage group will also be deleted.

```
IoTDB > DELETE STORAGE GROUP root.ln
IoTDB > DELETE STORAGE GROUP root.sgcc
// delete all data, all timeseries and all storage groups
IoTDB > DELETE STORAGE GROUP root.*
```

Create Timeseries

According to the storage model selected before, we can create corresponding timeseries in the two storage groups respectively. The SQL statements for creating timeseries are as follows:

```
IoTDB > create timeseries root.ln.wf01.wt01.status with datatype=BOOLEAN,encoding=PLAIN
IoTDB > create timeseries root.ln.wf01.wt01.temperature with datatype=FLOAT,encoding=RLE
IoTDB > create timeseries root.ln.wf02.wt02.hardware with datatype=TEXT,encoding=PLAIN
IoTDB > create timeseries root.ln.wf02.wt02.status with datatype=BOOLEAN,encoding=PLAIN
IoTDB > create timeseries root.sgcc.wf03.wt01.status with datatype=BOOLEAN,encoding=PLAIN
IoTDB > create timeseries root.sgcc.wf03.wt01.temperature with datatype=FLOAT,encoding=RLE
```

Notice that when in the CRATE TIMESERIES statement the encoding method conflicts with the data type, the system gives the corresponding error prompt as shown below:

```
IoTDB> create timeseries root.ln.wf02.wt02.status WITH DATATYPE=BOOLEAN, ENCODING=TS_2DIFF
error: encoding TS_2DIFF does not support BOOLEAN
```

Please refer to Encoding for correspondence between data type and encoding.

Tag and attribute management

We can also add an alias, extra tag and attribute information while creating one timeseries. The SQL statements for creating timeseries with extra tag and attribute information are extended as follows:

```
    create timeseries root.turbine.d1.s1(temprature) with datatype=FLOAT, encoding=RLE, compression=SNAPPY
1.  tags(tag1=v1, tag2=v2) attributes(attr1=v1, attr2=v2)
```

The `temprature` in the brackets is an alias for the sensor `s1`. So we can use `temprature` to replace `s1` anywhere.

> IoTDB also supports using AS function to set alias. The difference between the two is: the alias set by the AS function is used to replace the whole time series name, temporary and not bound with the time series; while the alias mentioned above is only used as the alias of the sensor, which is bound with it and can be used equivalent to the original sensor name.

The only difference between tag and attribute is that we will maintain an inverted index on the tag, so we can use tag property in the show timeseries where clause which you can see in the following `Show Timeseries` section.

> Notice that the size of the extra tag and attribute information shouldn't exceed the `tag_attribute_total_size`.

UPDATE TAG OPERATION

We can update the tag information after creating it as following:

- Rename the tag/attribute key

```
1.  ALTER timeseries root.turbine.d1.s1 RENAME tag1 TO newTag1
```

- reset the tag/attribute value

```
1.  ALTER timeseries root.turbine.d1.s1 SET tag1=newV1, attr1=newV1
```

- delete the existing tag/attribute

```
1.  ALTER timeseries root.turbine.d1.s1 DROP tag1, tag2
```

- add new tags

```
1.  ALTER timeseries root.turbine.d1.s1 ADD TAGS tag3=v3, tag4=v4
```

- add new attributes

```
1.  ALTER timeseries root.turbine.d1.s1 ADD ATTRIBUTES attr3=v3, attr4=v4
```

- upsert alias, tags and attributes

> add alias or a new key-value if the alias or key doesn't exist, otherwise, update the old one with new value.

```
ALTER timeseries root.turbine.d1.s1 UPSERT ALIAS=newAlias TAGS(tag3=v3, tag4=v4) ATTRIBUTES(attr3=v3,
attr4=v4)
```

Show Timeseries

- SHOW LATEST? TIMESERIES prefixPath? showWhereClause? limitClause?

  There are four optional clauses added in SHOW TIMESERIES, return information of time series

Timeseries information includes: timeseries path, alias of measurement, storage group it belongs to, data type, encoding type, compression type, tags and attributes.

Examples:

- SHOW TIMESERIES

  presents all timeseries information in JSON form

- SHOW TIMESERIES < `Path` >

  returns all timeseries information under the given < `Path` >. < `Path` > needs to be a prefix path or a path with star or a timeseries path. SQL statements are as follows:

```
IoTDB> show timeseries root
IoTDB> show timeseries root.ln
```

The results are shown below respectively:

```
IoTDB> show timeseries root
+-------------------------------+-------------+--------+--------+
|                     Timeseries|Storage Group|DataType|Encoding|
+-------------------------------+-------------+--------+--------+
|       root.ln.wf01.wt01.status|      root.ln| BOOLEAN|   PLAIN|
|  root.ln.wf01.wt01.temperature|      root.ln|   FLOAT|     RLE|
|     root.ln.wf02.wt02.hardware|      root.ln|    TEXT|   PLAIN|
|       root.ln.wf02.wt02.status|      root.ln| BOOLEAN|   PLAIN|
|     root.sgcc.wf03.wt01.status|    root.sgcc| BOOLEAN|   PLAIN|
|root.sgcc.wf03.wt01.temperature|    root.sgcc|   FLOAT|     RLE|
+-------------------------------+-------------+--------+--------+
---------------------------------------
timeseries number = 6
Execute successfully.
It costs 0.007s
```

```
IoTDB> show timeseries root.ln
+-------------------------------+-------------+--------+--------+
|                     Timeseries|Storage Group|DataType|Encoding|
+-------------------------------+-------------+--------+--------+
|      root.ln.wf01.wt01.status|      root.ln| BOOLEAN|   PLAIN|
|root.ln.wf01.wt01.temperature|      root.ln|   FLOAT|     RLE|
|    root.ln.wf02.wt02.hardware|      root.ln|    TEXT|   PLAIN|
|      root.ln.wf02.wt02.status|      root.ln| BOOLEAN|   PLAIN|
+-------------------------------+-------------+--------+--------+
-------------------------------------
timeseries number = 4
Execute successfully.
It costs 0.059s
```

- SHOW TIMESERIES (< `PrefixPath` >)? WhereClause

  returns all the timeseries information that satisfy the where condition and start with the prefixPath SQL statements are as follows:

```
1. show timeseries root.ln where unit=c
2. show timeseries root.ln where description contains 'test1'
```

The results are shown below respectly:

```
IoTDB> show timeseries root.ln where unit=c
+-------------+-----------+-------------+--------+--------+-----------+-------+-------+-----------------+----+
|   timeseries|      alias|storage group|dataType|encoding|compression|H_Alarm|M_Alarm|      description|unit|
+-------------+-----------+-------------+--------+--------+-----------+-------+-------+-----------------+----+
|root.ln.d0.s0|temperature|      root.ln|   FLOAT|     RLE|     SNAPPY|   1000|    500|ln this is a test1|   c|
+-------------+-----------+-------------+--------+--------+-----------+-------+-------+-----------------+----+
Total line number = 1
It costs 0.003s
IoTDB> show timeseries root.ln where description contains 'test1'
+-------------+-----------+-------------+--------+--------+-----------+-------+-------+-----------------+----+
|   timeseries|      alias|storage group|dataType|encoding|compression|H_Alarm|M_Alarm|      description|unit|
+-------------+-----------+-------------+--------+--------+-----------+-------+-------+-----------------+----+
|root.ln.d0.s0|temperature|      root.ln|   FLOAT|     RLE|     SNAPPY|   1000|    500|ln this is a test1|   c|
+-------------+-----------+-------------+--------+--------+-----------+-------+-------+-----------------+----+
Total line number = 1
It costs 0.003s
```

> Notice that, we only support one condition in the where clause. Either it's an equal filter or it is an `contains` filter. In both case, the property in the where condition must be a tag.

- SHOW TIMESERIES LIMIT INT OFFSET INT

  returns all the timeseries information start from the offset and limit the number of series returned

- SHOW LATEST TIMESERIES

  all the returned timeseries information should be sorted in descending order of the last timestamp of timeseries

It is worth noting that when the queried path does not exist, the system will return no timeseries.

Show Child Paths

```
SHOW CHILD PATHS prefixPath
```

Return all child paths of the prefixPath, the prefixPath could contains *.

Example:

- return the child paths of root.ln: show child paths root.ln

```
+------------+
| child paths|
+------------+
|root.ln.wf01|
|root.ln.wf02|
+------------+
```

- get all paths in form of root.xx.xx.xx: show child paths root.*.*

```
+---------------+
|    child paths|
+---------------+
|root.ln.wf01.s1|
|root.ln.wf02.s2|
+---------------+
```

Count Timeseries

IoTDB is able to use `COUNT TIMESERIES <Path>` to count the number of timeseries in the path. SQL statements are as follows:

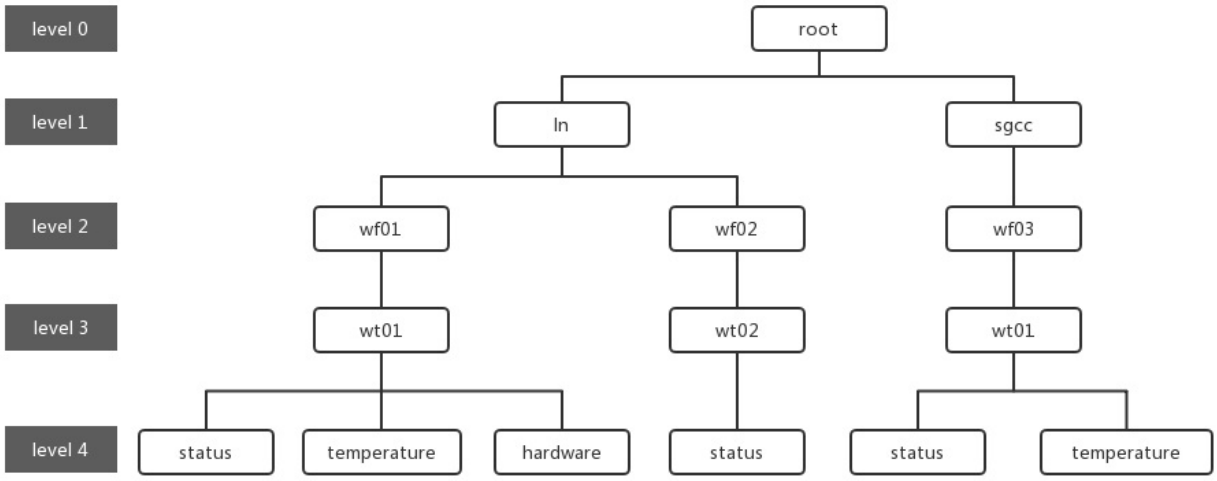
```
IoTDB > COUNT TIMESERIES root
IoTDB > COUNT TIMESERIES root.ln
IoTDB > COUNT TIMESERIES root.ln.*.*.status
IoTDB > COUNT TIMESERIES root.ln.wf01.wt01.status
```

Besides, `LEVEL` could be defined to show count the number of timeseries of each node at the given level in current Metadata Tree. This could be used to query the number of sensors under each device. The grammar is: `COUNT TIMESERIES <Path> GROUP BY LEVEL=<INTEGER>` .

For example, if there are several timeseries (use `show timeseries` to show all timeseries):

```
IoTDB> show timeseries
+-----------------------------------------------------------------------+-----------------------------------------+--------+--------+
|                                                             Timeseries|                        Storage Group|DataType|Encoding|
+-----------------------------------------------------------------------+-----------------------------------------+--------+--------+
|                                      root.ln.wf01.wt01.status|                  root.ln.wf01.wt01| BOOLEAN|   PLAIN|
|                                 root.ln.wf01.wt01.temperature|                  root.ln.wf01.wt01|   FLOAT|   PLAIN|
|                                    root.ln.wf01.wt01.hardware|                  root.ln.wf01.wt01|   INT32|   PLAIN|
|                                      root.ln.wf02.wt02.status|                       root.ln.wf02| BOOLEAN|   PLAIN|
|                                      root.sgcc.wf03.wt01.status|                     root.sgcc.wf03| BOOLEAN|   PLAIN|
|                                 root.sgcc.wf03.wt01.temperature|                     root.sgcc.wf03|   FLOAT|     RLE|
+-----------------------------------------------------------------------+-----------------------------------------+--------+--------+
Total timeseries number = 6
```

Then the Metadata Tree will be as below:



As can be seen, `root` is considered as `LEVEL=0`. So when you enter statements such as:

```
1. IoTDB > COUNT TIMESERIES root GROUP BY LEVEL=1
2. IoTDB > COUNT TIMESERIES root.ln GROUP BY LEVEL=2
3. IoTDB > COUNT TIMESERIES root.ln.wf01 GROUP BY LEVEL=2
```

You will get following results:

```
IoTDB> COUNT TIMESERIES root GROUP BY LEVEL=1
+------------------------------------------------------------------------------+-----------------------------------------------+
|                                                                        column|                                          count|
+------------------------------------------------------------------------------+-----------------------------------------------+
|                                                                      root.ln|                                              4|
|                                                                    root.sgcc|                                              2|
+------------------------------------------------------------------------------+-----------------------------------------------+
It costs 0.004s
IoTDB> COUNT TIMESERIES root.ln GROUP BY LEVEL=2
+------------------------------------------------------------------------------+-----------------------------------------------+
|                                                                        column|                                          count|
+------------------------------------------------------------------------------+-----------------------------------------------+
|                                                                 root.ln.wf02|                                              1|
|                                                                 root.ln.wf01|                                              3|
+------------------------------------------------------------------------------+-----------------------------------------------+
It costs 0.003s
IoTDB> COUNT TIMESERIES root.ln.wf01 GROUP BY LEVEL=2
+------------------------------------------------------------------------------+-----------------------------------------------+
|                                                                        column|                                          count|
+------------------------------------------------------------------------------+-----------------------------------------------+
|                                                                 root.ln.wf01|                                              3|
+------------------------------------------------------------------------------+-----------------------------------------------+
It costs 0.002s
```

> Note: The path of timeseries is just a filter condition, which has no relationship with the definition of level.

Count Nodes

IoTDB is able to use `COUNT NODES <PrefixPath> LEVEL=<INTEGER>` to count the number of nodes at the given level in current Metadata Tree. This could be used to query the number of devices. The usage are as follows:

```
1. IoTDB > COUNT NODES root LEVEL=2
2. IoTDB > COUNT NODES root.ln LEVEL=2
3. IoTDB > COUNT NODES root.ln.wf01 LEVEL=3
```

As for the above mentioned example and Metadata tree, you can get following results:

```
IoTDB> COUNT NODES root LEVEL=2
+------------------------------------------------------------------------+
|                                                                   count|
+------------------------------------------------------------------------+
|                                                                       3|
+------------------------------------------------------------------------+
It costs 0.002s
IoTDB> COUNT NODES root.ln LEVEL=2
+------------------------------------------------------------------------+
|                                                                   count|
+------------------------------------------------------------------------+
|                                                                       2|
+------------------------------------------------------------------------+
It costs 0.001s
IoTDB> COUNT NODES root.ln.wf01 LEVEL=3
+------------------------------------------------------------------------+
|                                                                   count|
+------------------------------------------------------------------------+
|                                                                       1|
+------------------------------------------------------------------------+
It costs 0.001s
```

> Note: The path of timeseries is just a filter condition, which has no relationship with the definition of level. `PrefixPath` could contains `*`, but all nodes after `*` would be ignored. Only the prefix path before `*` is valid.

Delete Timeseries

To delete the timeseries we created before, we are able to use `DELETE TimeSeries <PrefixPath>` statement.

The usage are as follows:

```
IoTDB> delete timeseries root.ln.wf01.wt01.status
IoTDB> delete timeseries root.ln.wf01.wt01.temperature, root.ln.wf02.wt02.hardware
IoTDB> delete timeseries root.ln.wf02.*
```

Show Devices

Similar to `Show Timeseries`, IoTDB also supports two ways of viewing devices:

- `SHOW DEVICES` statement presents all devices information, which is equal to `SHOW DEVICES root`.
- `SHOW DEVICES <PrefixPath>` statement specifies the `PrefixPath` and returns the devices information under the given level.

SQL statement is as follows:

```
IoTDB> show devices
IoTDB> show devices root.ln
```

TTL

IoTDB supports storage-level TTL settings, which means it is able to delete old data automatically and periodically. The benefit of using TTL is that hopefully you can control the total disk space usage and prevent the machine from running out of disks. Moreover, the query performance may downgrade as the total number of files goes up and the memory usage also increase as there are more files. Timely removing such files helps to keep at a high query performance level and reduce memory usage.

Set TTL

The SQL Statement for setting TTL is as follow:

```
IoTDB> set ttl to root.ln 3600000
```

This example means that for data in `root.ln`, only that of the latest 1 hour will remain, the older one is removed or made invisible.

Unset TTL

To unset TTL, we can use follwing SQL statement:

```
IoTDB> unset ttl to root.ln
```

After unset TTL, all data will be accepted in `root.ln`

Show TTL

To Show TTL, we can use follwing SQL statement:

```
IoTDB> SHOW ALL TTL
IoTDB> SHOW TTL ON StorageGroupNames
```

The SHOW ALL TTL example gives the TTL for all storage groups. The SHOW TTL ON root.group1 , root.group2 , root.group3 example shows the TTL for the three storage groups specified. Note: the TTL for storage groups that do not have a TTL set will display as null.

FLUSH

Persist all the data points in the memory table of the storage group to the disk, and seal the data file.

```
IoTDB> FLUSH
IoTDB> FLUSH root.ln
IoTDB> FLUSH root.sg1,root.sg2
```

MERGE

Merge sequence and unsequence data. Currently IoTDB supports the following two types of SQL to manually trigger the merge process of data files:

- `MERGE` Only rewrite overlapped Chunks, the merge speed is quick, while there will be redundant data on the disk eventually.
- `FULL MERGE` Rewrite all data in overlapped files, the merge speed is slow, but there will be no redundant data on the disk eventually.

```
IoTDB> MERGE
IoTDB> FULL MERGE
```

CLEAR CACHE

Clear the cache of chunk, chunk metadata and timeseries metadata to release the memory

footprint.

```
IoTDB> CLEAR CACHE
```

CREATE SNAPSHOT FOR SCHEMA

To speed up restarting of IoTDB, users can create snapshot of schema and avoid recovering schema from mlog file.

```
IoTDB> CREATE SNAPSHOT FOR SCHEMA
```

DML (Data Manipulation Language)

INSERT

Insert Real-time Data

IoTDB provides users with a variety of ways to insert real-time data, such as directly inputting INSERT SQL statement in Client/Shell tools, or using Java JDBC to perform single or batch execution of INSERT SQL statement.

This section mainly introduces the use of INSERT SQL statement for real-time data import in the scenario.

Use of INSERT Statements

The INSERT SQL statement statement is used to insert data into one or more specified timeseries created. For each point of data inserted, it consists of a timestamp and a sensor acquisition value (see Data Type).

In the scenario of this section, take two timeseries `root.ln.wf02.wt02.status` and `root.ln.wf02.wt02.hardware` as an example, and their data types are BOOLEAN and TEXT, respectively.

The sample code for single column data insertion is as follows:

```
1. IoTDB > insert into root.ln.wf02.wt02(timestamp,status) values(1,true)
2. IoTDB > insert into root.ln.wf02.wt02(timestamp,hardware) values(1, "v1")
```

The above example code inserts the long integer timestamp and the value “true” into the timeseries `root.ln.wf02.wt02.status` and inserts the long integer timestamp and the value “v1” into the timeseries `root.ln.wf02.wt02.hardware`. When the execution is successful, cost time is shown to indicate that the data insertion has been completed.

> Note: In IoTDB, TEXT type data can be represented by single and double quotation marks. The insertion statement above uses double quotation marks for TEXT type data. The following example will use single quotation marks for TEXT type data.

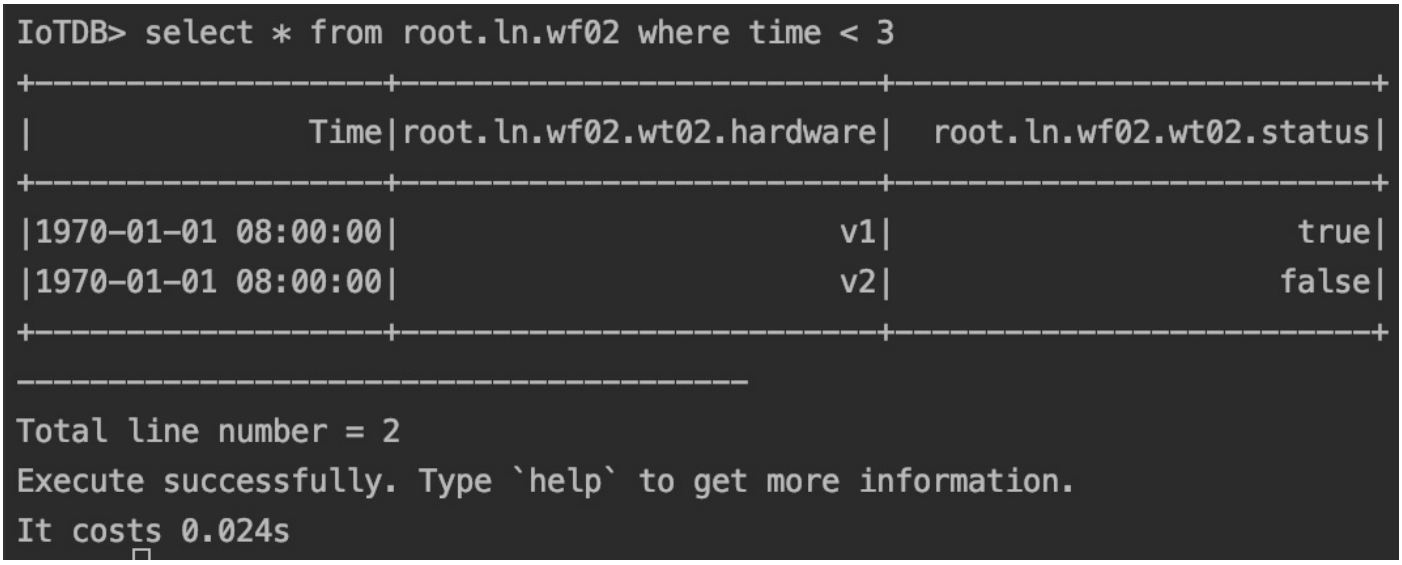
The INSERT statement can also support the insertion of multi-column data at the same time point. The sample code of inserting the values of the two timeseries at the same time point ‘2’ is as follows:

```
1. IoTDB > insert into root.ln.wf02.wt02(timestamp, status, hardware) VALUES (2, false, 'v2')
```

After inserting the data, we can simply query the inserted data using the SELECT statement:

```
1. IoTDB > select * from root.ln.wf02 where time < 3
```

The result is shown below. The query result shows that the insertion statements of single column and multi column data are performed correctly.



Error Handling of INSERT Statements

If the user inserts data into a non-existent timeseries, for example, execute the following commands:

```
1. IoTDB > insert into root.ln.wf02.wt02(timestamp, temperature) values(1,"v1")
```

Because `root.ln.wf02.wt02. temperature` does not exist, the system will return the following ERROR information:

```
   Msg: The resultDataType or encoding or compression of the last node temperature is conflicting in the storage
1. group root.ln
```

If the data type inserted by the user is inconsistent with the corresponding data type of the timeseries, for example, execute the following command:

```
1. IoTDB > insert into root.ln.wf02.wt02(timestamp,hardware) values(1,100)
```

The system will return the following ERROR information:

```
1. error: The TEXT data type should be covered by " or '
```

SELECT

Time Slice Query

This chapter mainly introduces the relevant examples of time slice query using IoTDB SELECT statements. Detailed SQL syntax and usage specifications can be found in SQL Documentation. You can also use the Java JDBC standard interface to execute related queries.

Select a Column of Data Based on a Time Interval

The SQL statement is:

```
1. select temperature from root.ln.wf01.wt01 where time < 2017-11-01T00:08:00.000
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is the temperature sensor (temperature). The SQL statement requires that all temperature sensor values before the time point of “2017-11-01T00:08:00.000” be selected.

The execution result of this SQL statement is as follows:

```
IoTDB> select temperature from root.ln.wf01.wt01 where time < 2017-11-01T00:08:00.000
+-----------------------------+------------------------------+
|                         Time|root.ln.wf01.wt01.temperature|
+-----------------------------+------------------------------+
|2017-11-01T00:00:00+08:00|                         25.96|
|2017-11-01T00:01:00+08:00|                         24.36|
|2017-11-01T00:02:00+08:00|                         20.09|
|2017-11-01T00:03:00+08:00|                         20.18|
|2017-11-01T00:04:00+08:00|                         21.13|
|2017-11-01T00:05:00+08:00|                         22.72|
|2017-11-01T00:06:00+08:00|                         20.71|
|2017-11-01T00:07:00+08:00|                         21.45|
+-----------------------------+------------------------------+
Total line number = 8
It costs 0.043s
IoTDB>
```

Select Multiple Columns of Data Based on a Time Interval

The SQL statement is:

```
1. select status, temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000;
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is “status” and “temperature”. The SQL statement requires that the status and temperature sensor values between the time point of “2017-11-01T00:05:00.000” and “2017-11-01T00:12:00.000” be selected.

The execution result of this SQL statement is as follows:

```
IoTDB> select status, temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000;
+-----------------------------+------------------------------+-----------------------------+
|                         Time|     root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------------+------------------------------+-----------------------------+
|2017-11-01T00:06:00+08:00|                         false|                        20.71|
|2017-11-01T00:07:00+08:00|                         false|                        21.45|
|2017-11-01T00:08:00+08:00|                         false|                        22.58|
|2017-11-01T00:09:00+08:00|                         false|                        20.98|
|2017-11-01T00:10:00+08:00|                          true|                        25.52|
|2017-11-01T00:11:00+08:00|                         false|                        22.91|
+-----------------------------+------------------------------+-----------------------------+
Total line number = 6
It costs 0.223s
IoTDB>
```

Select Multiple Columns of Data for the Same Device According to Multiple Time Intervals

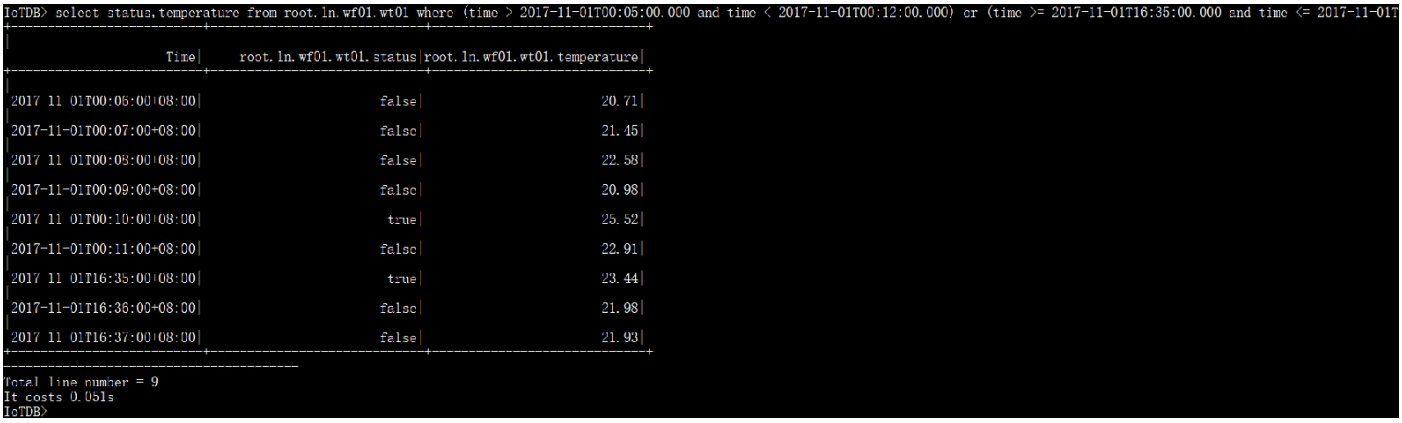
IoTDB supports specifying multiple time interval conditions in a query. Users can combine time interval conditions at will according to their needs. For example, the SQL statement is:

```
select status,temperature from root.ln.wf01.wt01 where (time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000) or (time >= 2017-11-01T16:35:00.000 and time <= 2017-11-01T16:37:00.000);
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is “status” and “temperature”; the statement specifies two different time intervals, namely “2017-11-01T00:05:00.000 to 2017-11-01T00:12:00.000” and “2017-11-01T16:35:00.000 to 2017-11-01T16:37:00.000”. The SQL statement requires that the values of selected timeseries satisfying any time interval be selected.

The execution result of this SQL statement is as follows:



Choose Multiple Columns of Data for Different Devices According to Multiple Time Intervals

The system supports the selection of data in any column in a query, i.e., the selected columns can come from different devices. For example, the SQL statement is:

```
select wf01.wt01.status,wf02.wt02.hardware from root.ln where (time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000) or (time >= 2017-11-01T16:35:00.000 and time <= 2017-11-01T16:37:00.000);
```

which means:

The selected timeseries are “the power supply status of ln group wf01 plant wt01 device” and “the hardware version of ln group wf02 plant wt02 device”; the statement specifies two different time intervals, namely “2017-11-01T00:05:00.000 to 2017-11-01T00:12:00.000” and “2017-11-01T16:35:00.000 to 2017-11-01T16:37:00.000”. The SQL statement requires that the values of selected timeseries satisfying any time interval be selected.

The execution result of this SQL statement is as follows:

```
IoTDB> select wf01.wt01.status,wf02.wt02.hardware from root.ln where (time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000) or (time >= 2017-11-01T16:35:00.000 and time <= 2017-11-01T16:37:00.000)
+-----------------------+--------------------------+--------------------------+
|                   Time|  root.ln.wf01.wt01.status|root.ln.wf02.wt02.hardware|
+-----------------------+--------------------------+--------------------------+
|2017-11-01T00:06:00.000|                     false|                        v1|
|2017-11-01T00:07:00.000|                     false|                        v1|
|2017-11-01T00:08:00.000|                     false|                        v1|
|2017-11-01T00:09:00.000|                     false|                        v1|
|2017-11-01T00:10:00.000|                      true|                        v2|
|2017-11-01T00:11:00.000|                     false|                        v1|
|2017-11-01T16:35:00.000|                      true|                        v2|
|2017-11-01T16:36:00.000|                     false|                        v1|
|2017-11-01T16:37:00.000|                     false|                        v1|
+-----------------------+--------------------------+--------------------------+
record number = 9
```

Order By Time Query

IoTDB supports the 'order by time' statement since 0.11, it's used to display results in descending order by time. For example, the SQL statement is:

```
1. select * from root.ln where time > 1 order by time desc limit 10;
```

Aggregate Query

This section mainly introduces the related examples of aggregate query.

Count Points

```
1. select count(status) from root.ln.wf01.wt01;
```

count(root.ln.wf01.wt01.status)
4

Count Points By Level

Level could be defined to show count the number of points of each node at the given level in current Metadata Tree.

This could be used to query the number of points under each device.

The SQL statement is:

```
1. select count(status) from root.ln.wf01.wt01 group by level=1;
```

Time	count(root.ln)
0	7

```
1. select count(status) from root.ln.wf01.wt01 group by level=2;
```

Time	count(root.ln.wf01)	count(root.ln.wf02)
0	4	3

Down-Frequency Aggregate Query

This section mainly introduces the related examples of down-frequency aggregation query, using the GROUP BY clause, which is used to partition the result set according to the user's given partitioning conditions and aggregate the partitioned result set. IoTDB supports partitioning result sets according to time interval and customized sliding step which should not be smaller than the time interval and defaults to equal the time interval if not set. And by default results are sorted by time in ascending order. You can also use the Java JDBC standard interface to execute related queries.

The GROUP BY statement provides users with three types of specified parameters:

- Parameter 1: The display window on the time axis
- Parameter 2: Time interval for dividing the time axis(should be positive)
- Parameter 3: Time sliding step (optional and should not be smaller than the time interval and defaults to equal the time interval if not set)

The actual meanings of the three types of parameters are shown in Figure 5.2 below. Among them, the parameter 3 is optional. There are three typical examples of frequency reduction aggregation: parameter 3 not specified, parameter 3 specified, and value filtering conditions specified.

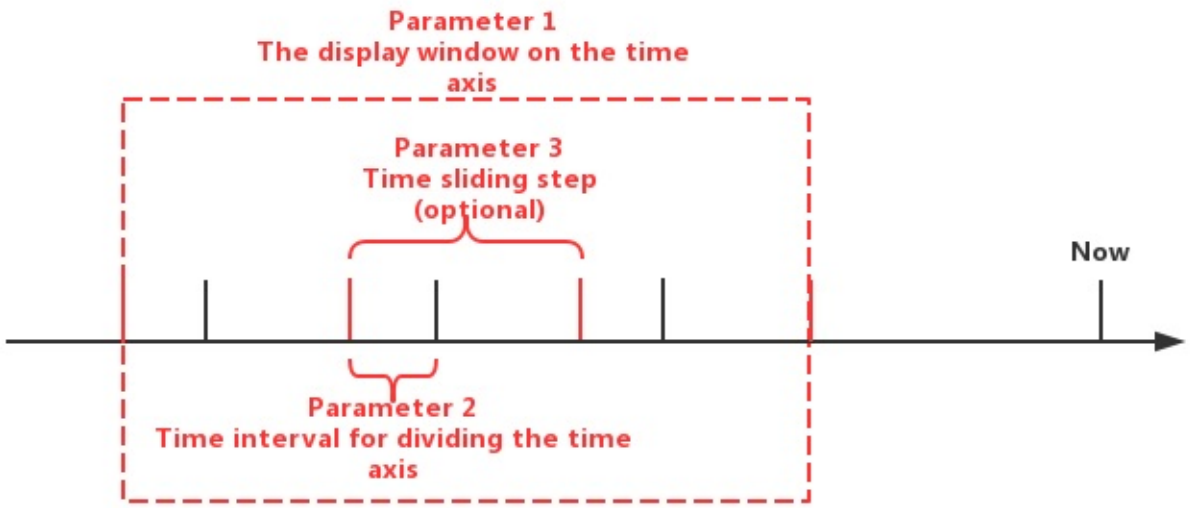


Figure 5.2 The actual meanings of the three types of parameters

Down-Frequency Aggregate Query without Specifying the Sliding Step Length

The SQL statement is:

```
    select count(status), max_value(temperature) from root.ln.wf01.wt01 group by ([2017-11-01T00:00:00, 2017-11-
1. 07T23:00:00),1d);
```

which means:

Since the sliding step length is not specified, the GROUP BY statement by default set the sliding step the same as the time interval which is `1d` .

The fist parameter of the GROUP BY statement above is the display window parameter, which determines the final display range is [2017-11-01T00:00:00, 2017-11-07T23:00:00).

The second parameter of the GROUP BY statement above is the time interval for dividing the time axis. Taking this parameter (1d) as time interval and startTime of the display window as the dividing origin, the time axis is divided into several

continuous intervals, which are [0,1d), [1d, 2d), [2d, 3d), etc.

Then the system will use the time and value filtering condition in the WHERE clause and the first parameter of the GROUP BY statement as the data filtering condition to obtain the data satisfying the filtering condition (which in this case is the data in the range of [2017-11-01T00:00:00, 2017-11-07 T23:00:00]), and map these data to the previously segmented time axis (in this case there are mapped data in every 1-day period from 2017-11-01T00:00:00 to 2017-11-07T23:00:00:00).

Since there is data for each time period in the result range to be displayed, the execution result of the SQL statement is shown below:

```
+-----------------------------+---------------------------------+-----------------------------------------+
|                         Time|count(root.ln.wf01.wt01.status)|max_value(root.ln.wf01.wt01.temperature)|
+-----------------------------+---------------------------------+-----------------------------------------+
|2017-11-01T00:00:00.000+08:00|                             1440|                                    26.0|
|2017-11-02T00:00:00.000+08:00|                             1440|                                    26.0|
|2017-11-03T00:00:00.000+08:00|                             1440|                                   25.99|
|2017-11-04T00:00:00.000+08:00|                             1440|                                    26.0|
|2017-11-05T00:00:00.000+08:00|                             1440|                                    26.0|
|2017-11-06T00:00:00.000+08:00|                             1440|                                   25.99|
|2017-11-07T00:00:00.000+08:00|                             1381|                                    26.0|
+-----------------------------+---------------------------------+-----------------------------------------+
```

Down-Frequency Aggregate Query Specifying the Sliding Step Length

The SQL statement is:

```
    select count(status), max_value(temperature) from root.ln.wf01.wt01 group by ([2017-11-01 00:00:00, 2017-11-07
1.  23:00:00), 3h, 1d);
```

which means:

Since the user specifies the sliding step parameter as 1d, the GROUP BY statement will move the time interval `1 day` long instead of `3 hours` as default.

That means we want to fetch all the data of 00:00:00 to 02:59:59 every day from 2017-11-01 to 2017-11-07.

The first parameter of the GROUP BY statement above is the display window parameter, which determines the final display range is [2017-11-01T00:00:00, 2017-11-07T23:00:00).

The second parameter of the GROUP BY statement above is the time interval for dividing the time axis. Taking this parameter (3h) as time interval and the startTime of the display window as the dividing origin, the time axis is divided into several continuous intervals, which are [2017-11-01T00:00:00, 2017-11-01T03:00:00), [2017-11-02T00:00:00, 2017-11-02T03:00:00), [2017-11-03T00:00:00, 2017-11-03T03:00:00), etc.

The third parameter of the GROUP BY statement above is the sliding step for each time interval moving.

Then the system will use the time and value filtering condition in the WHERE clause and the first parameter of the GROUP BY statement as the data filtering condition to obtain the data satisfying the filtering condition (which in this case is the data in the range of [2017-11-01T00:00:00, 2017-11-07T23:00:00]), and map these data to the previously segmented time axis (in this case there are mapped data in every 3-hour period for each day from 2017-11-01T00:00:00 to 2017-11-07T23:00:00:00).

Since there is data for each time period in the result range to be displayed, the execution result of the SQL statement is shown below:

```
+-----------------------------+---------------------------------+-----------------------------------------+
|                         Time|count(root.ln.wf01.wt01.status)|max_value(root.ln.wf01.wt01.temperature)|
+-----------------------------+---------------------------------+-----------------------------------------+
|2017-11-01T00:00:00.000+08:00|                              180|                                    25.98|
|2017-11-02T00:00:00.000+08:00|                              180|                                    25.98|
|2017-11-03T00:00:00.000+08:00|                              180|                                    25.96|
|2017-11-04T00:00:00.000+08:00|                              180|                                    25.96|
|2017-11-05T00:00:00.000+08:00|                              180|                                     26.0|
|2017-11-06T00:00:00.000+08:00|                              180|                                    25.85|
|2017-11-07T00:00:00.000+08:00|                              180|                                    25.99|
+-----------------------------+---------------------------------+-----------------------------------------+
```

Down-Frequency Aggregate Query Specifying the value Filtering Conditions

The SQL statement is:

```
    select count(status), max_value(temperature) from root.ln.wf01.wt01 where time > 2017-11-01T01:00:00 and
1.  temperature > 20 group by([2017-11-01T00:00:00, 2017-11-07T23:00:00), 3h, 1d);
```

which means:

Since the user specifies the sliding step parameter as 1d, the GROUP BY statement will move the time interval `1 day` long instead of `3 hours` as default.

The first parameter of the GROUP BY statement above is the display window parameter, which determines the final display range is [2017-11-01T00:00:00, 2017-11-07T23:00:00).

The second parameter of the GROUP BY statement above is the time interval for dividing the time axis. Taking this parameter (3h) as time interval and the startTime of the display window as the dividing origin, the time axis is divided into several continuous intervals, which are [2017-11-01T00:00:00, 2017-11-01T03:00:00), [2017-11-02T00:00:00, 2017-11-02T03:00:00), [2017-11-03T00:00:00, 2017-11-03T03:00:00), etc.

The third parameter of the GROUP BY statement above is the sliding step for each time interval moving.

Then the system will use the time and value filtering condition in the WHERE clause and the first parameter of the GROUP BY statement as the data filtering condition to obtain the data satisfying the filtering condition (which in this case is the data in the range of (2017-11-01T01:00:00, 2017-11-07T23:00:00] and satisfying

root.ln.wf01.wt01.temperature > 20), and map these data to the previously segmented time axis (in this case there are mapped data in every 3-hour period for each day from 2017-11-01T00:00:00 to 2017-11-07T23:00:00).

```
+-----------------------------+---------------------------------+-----------------------------------+
|                         Time|count(root.ln.wf01.wt01.status)|max_value(root.ln.wf01.wt01.temperature)|
+-----------------------------+---------------------------------+-----------------------------------+
|2017-11-01T00:00:00.000+08:00|                              119|                              25.98|
|2017-11-02T00:00:00.000+08:00|                              179|                              25.98|
|2017-11-03T00:00:00.000+08:00|                              180|                              25.96|
|2017-11-04T00:00:00.000+08:00|                              180|                              25.96|
|2017-11-05T00:00:00.000+08:00|                              180|                               26.0|
|2017-11-06T00:00:00.000+08:00|                              180|                              25.85|
|2017-11-07T00:00:00.000+08:00|                              180|                              25.99|
+-----------------------------+---------------------------------+-----------------------------------+
```

Left Open And Right Close Range

The SQL statement is:

```
1. select count(status) from root.ln.wf01.wt01 group by((5, 40], 5ms);
```

In this sql, the time interval is left open and right close, so we won't include the value of timestamp 5 and instead we will include the value of timestamp 40.

We will get the result like following:

Time	count(root.ln.wf01.wt01.status)
10	1
15	2
20	3
25	4
30	4
35	3
40	5

Down-Frequency Aggregate Query with Level Clause

Level could be defined to show count the number of points of each node at the given level in current Metadata Tree.

This could be used to query the number of points under each device.

The SQL statement is:

Get down-frequency aggregate query by level.

```
1. select count(status) from root.ln.wf01.wt01 group by ([0,20),3ms), level=1;
```

Time	count(root.ln)
0	1
3	0
6	0
9	1
12	3
15	0
18	0

Down-frequency aggregate query with sliding step and by level.

```
1. select count(status) from root.ln.wf01.wt01 group by ([0,20),2ms,3ms), level=1;
```

Time	count(root.ln)
0	1
3	0
6	0
9	0
12	2
15	0
18	0

Down-Frequency Aggregate Query with Fill Clause

In group by fill, sliding step is not supported in group by clause

Now, only last_value aggregation function is supported in group by fill.

Linear fill is not supported in group by fill.

Difference Between PREVIOUSUNTILLAST And PREVIOUS

- PREVIOUS will fill any null value as long as there exist value is not null before it.
- PREVIOUSUNTILLAST won’t fill the result whose time is after the last time of that time series.

The SQL statement is:

```
1. SELECT last_value(temperature) FROM root.ln.wf01.wt01 GROUP BY([8, 39), 5m) FILL (int32[PREVIOUSUNTILLAST])
   SELECT last_value(temperature) FROM root.ln.wf01.wt01 GROUP BY([8, 39), 5m) FILL (int32[PREVIOUSUNTILLAST,
2. 3m])
```

which means:

using PREVIOUSUNTILLAST Fill way to fill the origin down-frequency aggregate query result.

The path after SELECT in GROUP BY statement must be aggregate function, otherwise the system will give the corresponding error prompt, as shown below:

```
IoTDB> select count(status), temperature from root.ln.wf01.wt01 group by ([2017-11-01T00:00:00, 2017-11-07T23:00:00],1d);
Msg: Statement format is not right: Parsing error, statement [select count(status), temperature from root.ln.wf01.wt01 group
by ([2017-11-01T00:00:00, 2017-11-07T23:00:00],1d)] failed when parsing AST tree to generate logical operator. Detailed infor
mation: [line 1:34 mismatched input 'from' expecting ( near 'temperature'. Please refer to SQL document and check if there is
 any keyword conflict.]
It costs 0.007s
```

Last point Query

In scenarios when IoT devices updates data in a fast manner, users are more interested in the most recent point of IoT devices.

The Last point query is to return the most recent data point of the given timeseries in a three column format.

The SQL statement is defined as:

```
1. select last <Path> [COMMA <Path>]* from < PrefixPath > [COMMA < PrefixPath >]* <WhereClause>
```

which means: Query and return the last data points of timeseries prefixPath.path.

Only time filter with '>' or '>=' is supported in <WhereClause>. Any other filters given in the <WhereClause> will give an exception.

The result will be returned in a three column table format.

```
1. | Time | Path | Value |
```

Example 1: get the last point of root.ln.wf01.wt01.speed:

```
1. > select last speed from root.ln.wf01.wt01
2.
3. | Time | Path                   | Value |
4. | ---  | ---------------------- | ----- |
5. |  5   | root.ln.wf01.wt01.speed | 100   |
```

Example 2: get the last speed, status and temperature points of root.ln.wf01.wt01, whose timestamp larger or equal to 5.

```
1. > select last speed, status, temperature from root.ln.wf01.wt01 where time >= 5
2.
3. | Time | Path                         | Value |
4. | ---  | ---------------------------- | ----- |
5. |  5   | root.ln.wf01.wt01.speed      | 100   |
```

```
| 7   | root.ln.wf01.wt01.status     | true  |
| 9   | root.ln.wf01.wt01.temperature| 35.7  |
```

Automated Fill

In the actual use of IoTDB, when doing the query operation of timeseries, situations where the value is null at some time points may appear, which will obstruct the further analysis by users. In order to better reflect the degree of data change, users expect missing values to be automatically filled. Therefore, the IoTDB system introduces the function of Automated Fill.

Automated fill function refers to filling empty values according to the user's specified method and effective time range when performing timeseries queries for single or multiple columns. If the queried point's value is not null, the fill function will not work.

> Note: In the current version, IoTDB provides users with two methods: Previous and Linear. The previous method fills blanks with previous value. The linear method fills blanks through linear fitting. And the fill function can only be used when performing point-in-time queries.

Fill Function

- Previous Function

When the value of the queried timestamp is null, the value of the previous timestamp is used to fill the blank. The formalized previous method is as follows (see Section 7.1.3.6 for detailed syntax):

```
select <path> from <prefixPath> where time = <T> fill(<data_type>[previous, <before_range>], …)
```

Detailed descriptions of all parameters are given in Table 3-4.

Table 3-4 Previous fill paramter list

Parameter name (case insensitive)	Interpretation
path, prefixPath	query path; mandatory field
T	query timestamp (only one can be specified); mandatory field
data_type	the type of data used by the fill method. Optional values are int32, int64, float, double, boolean, text; optional field
before_range	represents the valid time range of the previous method. The previous method works when there are values in the [T-before_range, T] range. When before_range is not specified, before_range takes the default value default_fill_interval; -1 represents infinit; optional field

Here we give an example of filling null values using the previous method. The SQL

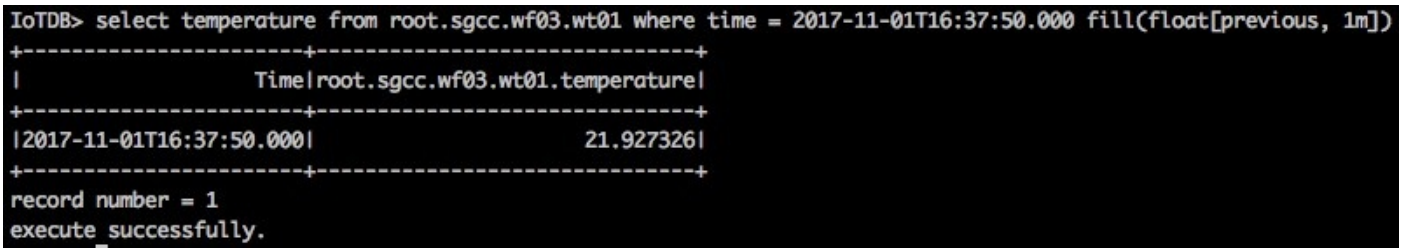
statement is as follows:

```
1. select temperature from root.sgcc.wf03.wt01 where time = 2017-11-01T16:37:50.000 fill(float[previous, 1m])
```

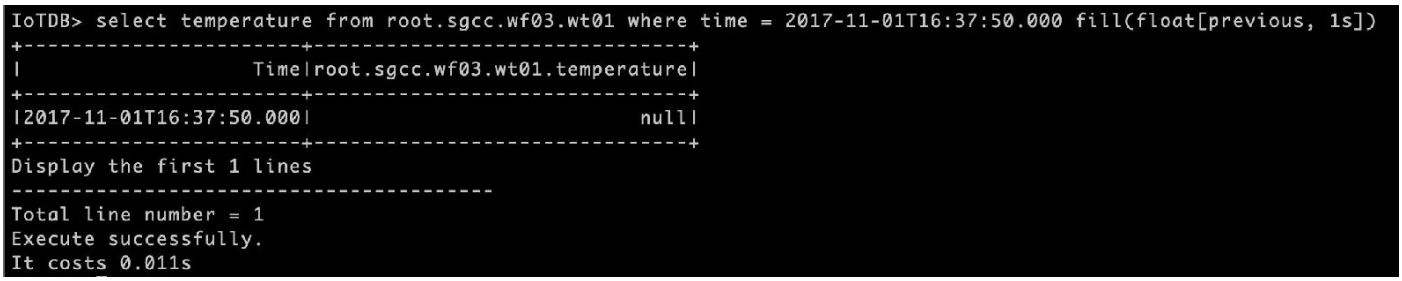
which means:

Because the timeseries root.sgcc.wf03.wt01.temperature is null at 2017-11-01T16:37:50.000, the system uses the previous timestamp 2017-11-01T16:37:00.000 (and the timestamp is in the [2017-11-01T16:36:50.000, 2017-11-01T16:37:50.000] time range) for fill and display.

On the sample data (opens new window), the execution result of this statement is shown below:



It is worth noting that if there is no value in the specified valid time range, the system will not fill the null value, as shown below:



- Linear Method

When the value of the queried timestamp is null, the value of the previous and the next timestamp is used to fill the blank. The formalized linear method is as follows:

```
1. select <path> from <prefixPath> where time = <T> fill(<data_type>[linear, <before_range>, <after_range>]…)
```

Detailed descriptions of all parameters are given in Table 3-5.

Table 3-5 Linear fill paramter list

Parameter name (case insensitive)	Interpretation
path, prefixPath	query path; mandatory field

T	query timestamp (only one can be specified); mandatory field
data_type	the type of data used by the fill method. Optional values are int32, int64, float, double, boolean, text; optional field
before_range, after_range	represents the valid time range of the linear method. The previous method works when there are values in the [T-before_range, T+after_range] range. When before_range and after_range are not explicitly specified, default_fill_interval is used. -1 represents infinity; optional field

Note if the timeseries has a valid value at query timestamp T, this value will be used as the linear fill value. Otherwise, if there is no valid fill value in either range [T-before_range, T] or [T, T + after_range], linear fill method will return null.

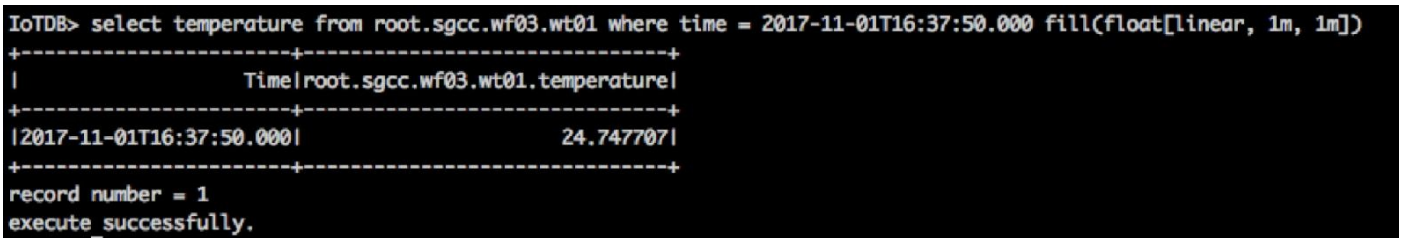
Here we give an example of filling null values using the linear method. The SQL statement is as follows:

```
1. select temperature from root.sgcc.wf03.wt01 where time = 2017-11-01T16:37:50.000 fill(float [linear, 1m, 1m])
```

which means:

Because the timeseries root.sgcc.wf03.wt01.temperature is null at 2017-11-01T16:37:50.000, the system uses the previous timestamp 2017-11-01T16:37:00.000 (and the timestamp is in the [2017-11-01T16:36:50.000, 2017-11-01T16:37:50.000] time range) and its value 21.927326, the next timestamp 2017-11-01T16:38:00.000 (and the timestamp is in the [2017-11-01T16:37:50.000, 2017-11-01T16:38:50.000] time range) and its value 25.311783 to perform linear fitting calculation: 21.927326 + (25.311783-21.927326)/60s * 50s = 24.747707

On the sample data (opens new window), the execution result of this statement is shown below:



Correspondence between Data Type and Fill Method

Data types and the supported fill methods are shown in Table 3-6.

Table 3-6 Data types and the supported fill methods

Data Type	Supported Fill Methods
boolean	previous
int32	previous, linear
int64	previous, linear

float	previous, linear
double	previous, linear
text	previous

It is worth noting that IoTDB will give error prompts for fill methods that are not supported by data types, as shown below:

```
IoTDB> select temperature from root.sgcc.wf03.wt01 where time = 2017-11-01T16:37:50.000 fill(boolean[linear, 1m, 1m])
error: type BOOLEAN cannot use TOK_LINEAR fill function
```

When the fill method is not specified, each data type bears its own default fill methods and parameters. The corresponding relationship is shown in Table 3-7.

Table 3-7 Default fill methods and parameters for various data types

Data Type	Default Fill Methods and Parameters
boolean	previous, 600000
int32	previous, 600000
int64	previous, 600000
float	previous, 600000
double	previous, 600000
text	previous, 600000

> Note: In version 0.7.0, at least one fill method should be specified in the Fill statement.

Row and Column Control over Query Results

IoTDB provides LIMIT/SLIMIT clause and OFFSET/SOFFSET clause in order to make users have more control over query results. The use of LIMIT and SLIMIT clauses allows users to control the number of rows and columns of query results, and the use of OFFSET and SOFSET clauses allows users to set the starting position of the results for display.

Note that the LIMIT and OFFSET are not supported in group by query.

This chapter mainly introduces related examples of row and column control of query results. You can also use the Java JDBC standard interface to execute queries.

Row Control over Query Results

By using LIMIT and OFFSET clauses, users control the query results in a row-related manner. We demonstrate how to use LIMIT and OFFSET clauses through the following examples.

- Example 1: basic LIMIT clause

The SQL statement is:

```
1. select status, temperature from root.ln.wf01.wt01 limit 10
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is “status” and “temperature”. The SQL statement requires the first 10 rows of the query result.

The result is shown below:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 limit 10;
+-----------------------+------------------------------+-----------------------------+
|                   Time|     root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------+------------------------------+-----------------------------+
|2017-11-01T00:00:00.000|                          true|                        25.96|
|2017-11-01T00:01:00.000|                          true|                        24.36|
|2017-11-01T00:02:00.000|                         false|                        20.09|
|2017-11-01T00:03:00.000|                         false|                        20.18|
|2017-11-01T00:04:00.000|                         false|                        21.13|
|2017-11-01T00:05:00.000|                         false|                        22.72|
|2017-11-01T00:06:00.000|                         false|                        20.71|
|2017-11-01T00:07:00.000|                         false|                        21.45|
|2017-11-01T00:08:00.000|                         false|                        22.58|
|2017-11-01T00:09:00.000|                         false|                        20.98|
+-----------------------+------------------------------+-----------------------------+
Display the first 10 lines
--------------------------------------

Total line number = 10
Execute successfully.
It costs 0.032s
```

- Example 2: LIMIT clause with OFFSET

The SQL statement is:

```
1. select status, temperature from root.ln.wf01.wt01 limit 5 offset 3
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is “status” and “temperature”. The SQL statement requires rows 3 to 7 of the query result be returned (with the first row numbered as row 0).

The result is shown below:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 limit 5 offset 3
+-----------------------+------------------------------+-----------------------------------+
|                   Time|     root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------+------------------------------+-----------------------------------+
|2017-11-01T00:03:00.000|                         false|                              20.18|
|2017-11-01T00:04:00.000|                         false|                              21.13|
|2017-11-01T00:05:00.000|                         false|                              22.72|
|2017-11-01T00:06:00.000|                         false|                              20.71|
|2017-11-01T00:07:00.000|                         false|                              21.45|
+-----------------------+------------------------------+-----------------------------------+
Display the first 5 lines
----------------------------------------

Total line number = 5
Execute successfully.
```

- Example 3: LIMIT clause combined with WHERE clause

The SQL statement is:

```
    select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time< 2017-11-
1.  01T00:12:00.000 limit 2 offset 3
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is “status” and “temperature”. The SQL statement requires rows 3 to 4 of the status and temperature sensor values between the time point of “2017-11-01T00:05:00.000” and “2017-11-01T00:12:00.000” (with the first row numbered as row 0).

The result is shown below:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and
 time < 2017-11-01T00:12:00.000 limit 2 offset 3
+-----------------------+------------------------------+-----------------------------------+
|                   Time|     root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------+------------------------------+-----------------------------------+
|2017-11-01T00:09:00.000|                         false|                              20.98|
|2017-11-01T00:10:00.000|                          true|                              25.52|
+-----------------------+------------------------------+-----------------------------------+
Display the first 2 lines
----------------------------------------

Total line number = 2
Execute successfully.
```

- Example 4: LIMIT clause combined with GROUP BY clause

The SQL statement is:

```
    select count(status), max_value(temperature) from root.ln.wf01.wt01 group by ([2017-11-01T00:00:00, 2017-11-
1.  07T23:00:00),1d) limit 5 offset 3
```

which means:

The SQL statement clause requires rows 3 to 7 of the query result be returned (with the first row numbered as row 0).

The result is shown below:

```
IoTDB> select count(status), max_value(temperature) from root.ln.wf01.wt01 group by ((1d, [2017-11-01T00:00:00,
 2017-11-07T23:00:00]) limit 5 offset 3
+-----------------------+--------------------------------------+--------------------------------------+
|                   Time|count(root.ln.wf01.wt01.status)|max_value(root.ln.wf01.wt01.temperature)|
+-----------------------+--------------------------------------+--------------------------------------+
|2017-11-04T00:00:00.000|                                  1440|                                  26.0|
|2017-11-05T00:00:00.000|                                  1440|                                  26.0|
|2017-11-06T00:00:00.000|                                  1440|                                 25.99|
|2017-11-07T00:00:00.000|                                  1381|                                  26.0|
+-----------------------+--------------------------------------+--------------------------------------+
Display the first 4 lines
----------------------------------------

Total line number = 4
Execute successfully.
It costs 0.016s
```

It is worth noting that because the current FILL clause can only fill in the missing value of timeseries at a certain time point, that is to say, the execution result of FILL clause is exactly one line, so LIMIT and OFFSET are not expected to be used in combination with FILL clause, otherwise errors will be prompted. For example, executing the following SQL statement:

```
    select temperature from root.sgcc.wf03.wt01 where time = 2017-11-01T16:37:50.000 fill(float[previous, 1m])
1.  limit 10
```

The SQL statement will not be executed and the corresponding error prompt is given as follows:

```
IoTDB> select temperature from root.sgcc.wf03.wt01 where time = 2017-11-01T16:37:50.000 fill(float[previous, 1
m]) limit 10
Msg: Statement format is not right:parsing error,statement: select temperature from root.sgcc.wf03.wt01 where
time = 2017-11-01T16:37:50.000 fill(float[previous, 1m]) limit 10
It costs 0.006s
IoTDB>
```

Column Control over Query Results

By using SLIMIT and SOFFSET clauses, users can control the query results in a column-related manner. We will demonstrate how to use SLIMIT and SOFFSET clauses through the following examples.

- Example 1: basic SLIMIT clause

The SQL statement is:

```
    select * from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 slimit
1.  1
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is the first column under this device, i.e., the power supply status. The SQL statement requires the status sensor values between the time point of “2017-11-01T00:05:00.000”

and “2017-11-01T00:12:00.000” be selected.

The result is shown below:

```
IoTDB> select * from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000
 slimit 1
+-----------------------+------------------------+
|                   Time|root.ln.wf01.wt01.status|
+-----------------------+------------------------+
|2017-11-01T00:06:00.000|                   false|
|2017-11-01T00:07:00.000|                   false|
|2017-11-01T00:08:00.000|                   false|
|2017-11-01T00:09:00.000|                   false|
|2017-11-01T00:10:00.000|                    true|
|2017-11-01T00:11:00.000|                   false|
+-----------------------+------------------------+
Display the first 6 lines
---------------------------------------
Total line number = 6
Execute successfully.
It costs 0.000s
```

- Example 2: SLIMIT clause with SOFFSET

The SQL statement is:

```
select * from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 slimit
1. 1 soffset 1
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is the second column under this device, i.e., the temperature. The SQL statement requires the temperature sensor values between the time point of “2017-11-01T00:05:00.000” and “2017-11-01T00:12:00.000” be selected.

The result is shown below:

```
IoTDB> select * from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:0
0.000 slimit 1 soffset 1
+-----------------------+-----------------------------+
|                   Time|root.ln.wf01.wt01.temperature|
+-----------------------+-----------------------------+
|2017-11-01T00:06:00.000|                        20.71|
|2017-11-01T00:07:00.000|                        21.45|
|2017-11-01T00:08:00.000|                        22.58|
|2017-11-01T00:09:00.000|                        20.98|
|2017-11-01T00:10:00.000|                        25.52|
|2017-11-01T00:11:00.000|                        22.91|
+-----------------------+-----------------------------+
Display the first 6 lines
---------------------------------------
Total line number = 6
Execute successfully.
It costs 0.093s
```

- Example 3: SLIMIT clause combined with GROUP BY clause

The SQL statement is:

```
select max_value(*) from root.ln.wf01.wt01 group by ([2017-11-01T00:00:00, 2017-11-07T23:00:00),1d) slimit 1
1. soffset 1
```

The result is shown below:

```
IoTDB> select max_value(*) from root.ln.wf01.wt01 group by (1d, [2017-11-01T00:00:00, 2017-11-07T23:00:00]) slimit 1
soffset 1
+-----------------------+-----------------------------------------+
|                   Time|max_value(root.ln.wf01.wt01.temperature)|
+-----------------------+-----------------------------------------+
|2017-11-01T00:00:00.000|                                     26.0|
|2017-11-02T00:00:00.000|                                     26.0|
|2017-11-03T00:00:00.000|                                    25.99|
|2017-11-04T00:00:00.000|                                     26.0|
|2017-11-05T00:00:00.000|                                     26.0|
|2017-11-06T00:00:00.000|                                    25.99|
|2017-11-07T00:00:00.000|                                     26.0|
+-----------------------+-----------------------------------------+
Display the first 7 lines
----------------------------------------
Total line number = 7
Execute successfully.
It costs 0.016s
```

- Example 4: SLIMIT clause combined with FILL clause

The SQL statement is:

```
    select * from root.sgcc.wf03.wt01 where time = 2017-11-01T16:37:50.000 fill(float[previous, 1m]) slimit 1
1.  soffset 1
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is the second column under this device, i.e., the temperature.

The result is shown below:

```
IoTDB> select * from root.sgcc.wf03.wt01 where time = 2017-11-01T16:37:50.000 fill(float[previous, 1m]) slimit 1
soffset 1
+-----------------------+------------------------------+
|                   Time|root.sgcc.wf03.wt01.temperature|
+-----------------------+------------------------------+
|2017-11-01T16:37:50.000|                         21.93|
+-----------------------+------------------------------+
Display the first 1 lines
----------------------------------------
Total line number = 1
Execute successfully.
It costs 0.016s
```

It is worth noting that SLIMIT clause is expected to be used in conjunction with star path or prefix path, and the system will prompt errors when SLIMIT clause is used in conjunction with complete path query. For example, executing the following SQL statement:

```
    select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-
1.  01T00:12:00.000 slimit 1
```

The SQL statement will not be executed and the corresponding error prompt is given as follows:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 slimit 1
Msg: Wrong use of SLIMIT: SLIMIT is not allowed to be used with complete paths.
It costs 0.015s
```

Row and Column Control over Query Results

In addition to row or column control over query results, IoTDB allows users to control both rows and columns of query results. Here is a complete example with both LIMIT clauses and SLIMIT clauses.

The SQL statement is:

```
1. select * from root.ln.wf01.wt01 limit 10 offset 100 slimit 2 soffset 0
```

which means:

The selected device is ln group wf01 plant wt01 device; the selected timeseries is columns 0 to 1 under this device (with the first column numbered as column 0). The SQL statement clause requires rows 100 to 109 of the query result be returned (with the first row numbered as row 0).

The result is shown below:

```
IoTDB> select * from root.ln.wf01.wt01 limit 10 offset 100 slimit 2 soffset 0
+-----------------------+------------------------------+-------------------------------+
|                   Time|     root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------+------------------------------+-------------------------------+
|2017-11-01T01:40:00.000|                         false|                          21.19|
|2017-11-01T01:41:00.000|                         false|                          22.79|
|2017-11-01T01:42:00.000|                         false|                          22.98|
|2017-11-01T01:43:00.000|                         false|                          21.52|
|2017-11-01T01:44:00.000|                          true|                          23.45|
|2017-11-01T01:45:00.000|                          true|                          24.06|
|2017-11-01T01:46:00.000|                         false|                           22.6|
|2017-11-01T01:47:00.000|                          true|                          23.78|
|2017-11-01T01:48:00.000|                          true|                          24.72|
|2017-11-01T01:49:00.000|                          true|                          24.68|
+-----------------------+------------------------------+-------------------------------+
Display the first 10 lines
----------------------------------------

Total line number = 10
Execute successfully.
It costs 0.031s
```

Use Alias

Since the unique data model of IoTDB, lots of additional information like device will be carried before each sensor. Sometimes, we want to query just one specific device, then these prefix information show frequently will be redundant in this situation, influencing the analysis of result set. At this time, we can use `AS` function provided by IoTDB, assign an alias to time series selected in query.

For example:

```
1. select s1 as temperature, s2 as speed from root.ln.wf01.wt01;
```

The result set is:

Time	temperature	speed
…	…	…

Other ResultSet Format

In addition, IoTDB supports two other result set format: ‘align by device’ and ‘disable align’.

The ‘align by device’ indicates that the deviceId is considered as a column. Therefore, there are totally limited columns in the dataset.

The SQL statement is:

```
1. select s1,s2 from root.sg1.* ALIGN BY DEVICE
```

For more syntax description, please read SQL REFERENCE.

The ‘disable align’ indicaes that there are 3 columns for each time series in the resultset. For more syntax description, please read SQL REFERENCE.

Error Handling

If the parameter N/SN of LIMIT/SLIMIT exceeds the size of the result set, IoTDB returns all the results as expected. For example, the query result of the original SQL statement consists of six rows, and we select the first 100 rows through the LIMIT clause:

```
    select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-
1. 01T00:12:00.000 limit 100
```

The result is shown below:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and
 time < 2017-11-01T00:12:00.000 limit 100
+-----------------------+---------------------------+-----------------------------+
|                   Time|     root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------+---------------------------+-----------------------------+
|2017-11-01T00:06:00.000|                      false|                        20.71|
|2017-11-01T00:07:00.000|                      false|                        21.45|
|2017-11-01T00:08:00.000|                      false|                        22.58|
|2017-11-01T00:09:00.000|                      false|                        20.98|
|2017-11-01T00:10:00.000|                       true|                        25.52|
|2017-11-01T00:11:00.000|                      false|                        22.91|
+-----------------------+---------------------------+-----------------------------+
Display the first 6 lines
---------------------------------------
Total line number = 6
Execute successfully.
It costs 0.018s
```

If the parameter N/SN of LIMIT/SLIMIT clause exceeds the allowable maximum value (N/SN is of type int32), the system prompts errors. For example, executing the following SQL statement:

```
1. select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 limit 1234567890123456789
```

The SQL statement will not be executed and the corresponding error prompt is given as follows:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and
time < 2017-11-01T00:12:00.000 limit 1234567890123456789
Msg: LIMIT <N>: N should be Int32.
It costs 0.031s
```

If the parameter N/SN of LIMIT/SLIMIT clause is not a positive intege, the system prompts errors. For example, executing the following SQL statement:

```
1. select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 limit 13.1
```

The SQL statement will not be executed and the corresponding error prompt is given as follows:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time
< 2017-11-01T00:12:00.000 limit 13.1
Msg: Statement format is not right:parsing error,statement: select status,temperature from root.ln.wf0
1.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 limit 13.1
It costs 0.010s
IoTDB>
```

If the parameter OFFSET of LIMIT clause exceeds the size of the result set, IoTDB will return an empty result set. For example, executing the following SQL statement:

```
1. select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 limit 2 offset 6
```

The result is shown below:

```
IoTDB> select status,temperature from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time
 < 2017-11-01T00:12:00.000 limit 2 offset 6
+-----------------------+------------------------------+------------------------------+
|                   Time|     root.ln.wf01.wt01.status|root.ln.wf01.wt01.temperature|
+-----------------------+------------------------------+------------------------------+
+-----------------------+------------------------------+------------------------------+
Display the first 0 lines
---------------------------------------
Total line number = 0
Execute successfully.
It costs 0.010s
```

If the parameter SOFFSET of SLIMIT clause is not smaller than the number of available timeseries, the system prompts errors. For example, executing the following SQL statement:

```
    1. select * from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00:12:00.000 slimit 1 soffset 2
```

The SQL statement will not be executed and the corresponding error prompt is given as follows:

```
IoTDB> select * from root.ln.wf01.wt01 where time > 2017-11-01T00:05:00.000 and time < 2017-11-01T00
:12:00.000 slimit 1 soffset 2
Msg: SOFFSET <SOFFSETValue>: SOFFSETValue exceeds the range.
It costs 0.007s
```

DELETE

Users can delete data that meet the deletion condition in the specified timeseries by using the DELETE statement. When deleting data, users can select one or more timeseries paths, prefix paths, or paths with star to delete data within a certain time interval.

In a JAVA programming environment, you can use the Java JDBC to execute single or batch UPDATE statements.

Delete Single Timeseries

Taking ln Group as an example, there exists such a usage scenario:

The wf02 plant’s wt02 device has many segments of errors in its power supply status before 2017-11-01 16:26:00, and the data cannot be analyzed correctly. The erroneous data affected the correlation analysis with other devices. At this point, the data before this time point needs to be deleted. The SQL statement for this operation is

```
    1. delete from root.ln.wf02.wt02.status where time<=2017-11-01T16:26:00;
```

In case we hope to merely delete the data before 2017-11-01 16:26:00 in the year of

2017, The SQL statement is:

```
delete from root.ln.wf02.wt02.status where time>=2017-01-01T00:00:00 and time<=2017-11-01T16:26:00;
```

IoTDB supports to delete a range of timeseries points. Users can write SQL expressions as follows to specify the delete interval:

```
delete from root.ln.wf02.wt02.status where time < 10
delete from root.ln.wf02.wt02.status where time <= 10
delete from root.ln.wf02.wt02.status where time < 20 and time > 10
delete from root.ln.wf02.wt02.status where time <= 20 and time >= 10
delete from root.ln.wf02.wt02.status where time > 20
delete from root.ln.wf02.wt02.status where time >= 20
delete from root.ln.wf02.wt02.status where time = 20
```

Please pay attention that multiple intervals connected by “OR” expression are not supported in delete statement:

```
delete from root.ln.wf02.wt02.status where time > 4 or time < 0
Msg: 303: Check metadata error: For delete statement, where clause can only contain atomic
expressions like : time > XXX, time <= XXX, or two atomic expressions connected by 'AND'
```

Delete Multiple Timeseries

If both the power supply status and hardware version of the ln group wf02 plant wt02 device before 2017-11-01 16:26:00 need to be deleted, the prefix path with broader meaning or the path with star can be used to delete the data. The SQL statement for this operation is:

```
delete from root.ln.wf02.wt02 where time <= 2017-11-01T16:26:00;
```

or

```
delete from root.ln.wf02.wt02.* where time <= 2017-11-01T16:26:00;
```

It should be noted that when the deleted path does not exist, IoTDB will give the corresponding error prompt as shown below:

```
IoTDB> delete from root.ln.wf03.wt02.status where time < now()
Msg: TimeSeries does not exist and its data cannot be deleted
```

Delete Time Partition (experimental)

You may delete all data in a time partition of a storage group using the following grammar:

```
1. DELETE PARTITION root.ln 0,1,2
```

The `0,1,2` above is the id of the partition that is to be deleted, you can find it from the IoTDB data folders or convert a timestamp manually to an id using `timestamp / partitionInterval` (flooring), and the `partitionInterval` should be in your config (if time-partitioning is supported in your version).

Please notice that this function is experimental and mainly for development, please use it with care.

Account Management Statements

IoTDB provides users with account privilege management operations, so as to ensure data security.

We will show you basic user privilege management operations through the following specific examples. Detailed SQL syntax and usage details can be found in SQL Documentation. At the same time, in the JAVA programming environment, you can use the Java JDBC to execute privilege management statements in a single or batch mode.

Basic Concepts

User

The user is the legal user of the database. A user corresponds to a unique username and has a password as a means of authentication. Before using a database, a person must first provide a legitimate username and password to make himself/herself a user.

Privilege

The database provides a variety of operations, and not all users can perform all operations. If a user can perform an operation, the user is said to have the privilege to perform the operation. privileges are divided into data management privilege (such as adding, deleting and modifying data) and authority management privilege (such as creation and deletion of users and roles, granting and revoking of privileges, etc.). Data management privilege often needs a path to limit its effective range, which is a subtree rooted at the path’s corresponding node.

Role

A role is a set of privileges and has a unique role name as an identifier. A user usually corresponds to a real identity (such as a traffic dispatcher), while a real identity may correspond to multiple users. These users with the same real identity tend to have the same privileges. Roles are abstractions that can unify the management of such privileges.

Default User

There is a default user in IoTDB after the initial installation: root, and the default password is root. This user is an administrator user, who cannot be deleted and has all the privileges. Neither can new privileges be granted to the root user nor can privileges owned by the root user be deleted.

Privilege Management Operation Examples

According to the sample data (opens new window), the sample data of IoTDB might belong to different power generation groups such as ln, sgcc, etc. Different power generation groups do not want others to obtain their own database data, so we need to have data privilege isolated at the group layer.

Create User

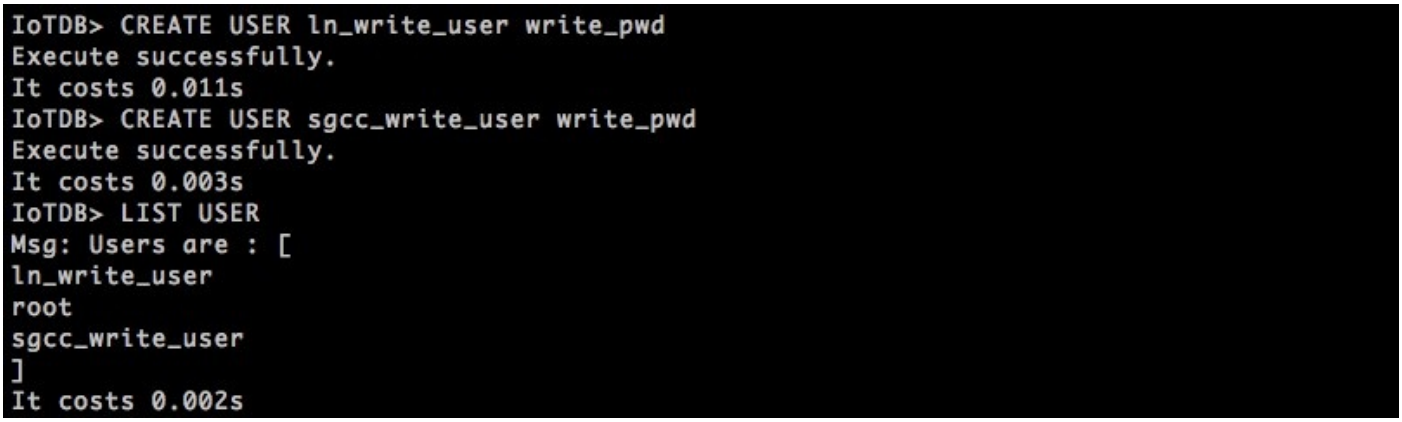
We can create two users for ln and sgcc groups, named ln_write_user and sgcc_write_user, with both passwords being write_pwd. The SQL statement is:

```
1. CREATE USER ln_write_user 'write_pwd'
2. CREATE USER sgcc_write_user 'write_pwd'
```

Then use the following SQL statement to show the user:

```
1. LIST USER
```

As can be seen from the result shown below, the two users have been created:



Grant User Privilege

At this point, although two users have been created, they do not have any privileges, so they can not operate on the database. For example, we use ln_write_user to write data in the database, the SQL statement is:

```
1. INSERT INTO root.ln.wf01.wt01(timestamp,status) values(1509465600000,true)
```

The SQL statement will not be executed and the corresponding error prompt is given as follows:

```
IoTDB> INSERT INTO root.ln.wf01.wt01(timestamp, status) values(1509465600000, true)
Msg: No permissions for this operation INSERT
It costs 0.019s
```

Now, we grant the two users write privileges to the corresponding storage groups, and try to write data again. The SQL statement is:

```
1. GRANT USER ln_write_user PRIVILEGES 'INSERT_TIMESERIES' on root.ln
2. GRANT USER sgcc_write_user PRIVILEGES 'INSERT_TIMESERIES' on root.sgcc
3. INSERT INTO root.ln.wf01.wt01(timestamp, status) values(1509465600000, true)
```

The execution result is as follows:

```
IoTDB> GRANT USER ln_write_user PRIVILEGES 'INSERT_TIMESERIES' on root.ln
Execute successfully.
It costs 0.005s
IoTDB> GRANT USER sgcc_write_user PRIVILEGES 'INSERT_TIMESERIES' on root.sgcc
Execute successfully.
It costs 0.002s
IoTDB> INSERT INTO root.ln.wf01.wt01(timestamp, status) values(1509465600000, true)
Execute successfully.
It costs 0.002s
```

Other Instructions

The Relationship among Users, Privileges and Roles

A Role is a set of privileges, and privileges and roles are both attributes of users. That is, a role can have several privileges and a user can have several roles and privileges (called the user’s own privileges).

At present, there is no conflicting privilege in IoTDB, so the real privileges of a user is the union of the user’s own privileges and the privileges of the user’s roles. That is to say, to determine whether a user can perform an operation, it depends on whether one of the user’s own privileges or the privileges of the user’s roles permits the operation. The user’s own privileges and privileges of the user’s roles may overlap, but it does not matter.

It should be noted that if users have a privilege (corresponding to operation A) themselves and their roles contain the same privilege, then revoking the privilege from the users themselves alone can not prohibit the users from performing operation A, since it is necessary to revoke the privilege from the role, or revoke the role from the user. Similarly, revoking the privilege from the users’s roles alone can not prohibit the users from performing operation A.

At the same time, changes to roles are immediately reflected on all users who own the roles. For example, adding certain privileges to roles will immediately give all users who own the roles corresponding privileges, and deleting certain privileges will also deprive the corresponding users of the privileges (unless the users themselves have

the privileges).

List of Privileges Included in the System

List of privileges Included in the System

privilege Name	Interpretation
SET_STORAGE_GROUP	create timeseries; set storage groups; path dependent
INSERT_TIMESERIES	insert data; path dependent
READ_TIMESERIES	query data; path dependent
DELETE_TIMESERIES	delete data or timeseries; path dependent
CREATE_USER	create users; path independent
DELETE_USER	delete users; path independent
MODIFY_PASSWORD	modify passwords for all users; path independent; (Those who do not have this privilege can still change their own asswords.)
LIST_USER	list all users; list a user’s privileges; list a user’s roles with three kinds of operation privileges; path independent
GRANT_USER_PRIVILEGE	grant user privileges; path independent
REVOKE_USER_PRIVILEGE	revoke user privileges; path independent
GRANT_USER_ROLE	grant user roles; path independent
REVOKE_USER_ROLE	revoke user roles; path independent
CREATE_ROLE	create roles; path independent
DELETE_ROLE	delete roles; path independent
LIST_ROLE	list all roles; list the privileges of a role; list the three kinds of operation privileges of all users owning a role; path independent
GRANT_ROLE_PRIVILEGE	grant role privileges; path independent
REVOKE_ROLE_PRIVILEGE	revoke role privileges; path independent

Username Restrictions

IoTDB specifies that the character length of a username should not be less than 4, and the username cannot contain spaces.

Password Restrictions

IoTDB specifies that the character length of a password should have no less than 4 character length, and no spaces. The password is encrypted with MD5.

Role Name Restrictions

IoTDB specifies that the character length of a role name should have no less than 4 character length, and no spaces.

SQL Reference

In this part, we will introduce you IoTDB's Query Language. IoTDB offers you a SQL-like query language for interacting with IoTDB, the query language can be devided into 4 major parts:

- Schema Statement: statements about schema management are all listed in this section.
- Data Management Statement: statements about data management (such as: data insertion, data query, etc.) are all listed in this section.
- Database Management Statement: statements about database management and authentication are all listed in this section.
- Functions: functions that IoTDB offers are all listed in this section.

All of these statements are write in IoTDB's own syntax, for details about the syntax composition, please check the `Reference` section.

Show Version

```
1. show version
```

```
1. +---------------+
2. |        version|
3. +---------------+
4. |0.11.1-SNAPSHOT|
5. +---------------+
6. Total line number = 1
7. It costs 0.417s
```

Schema Statement

- Set Storage Group

```
1. SET STORAGE GROUP TO <FullPath>
2. Eg: IoTDB > SET STORAGE GROUP TO root.ln.wf01.wt01
3. Note: FullPath can not include `*`
```

- Delete Storage Group

```
1. DELETE STORAGE GROUP <FullPath> [COMMA <FullPath>]*
2. Eg: IoTDB > DELETE STORAGE GROUP root.ln.wf01.wt01
3. Eg: IoTDB > DELETE STORAGE GROUP root.ln.wf01.wt01, root.ln.wf01.wt02
4. Eg: IoTDB > DELETE STORAGE GROUP root.ln.wf01.*
5. Eg: IoTDB > DELETE STORAGE GROUP root.*
```

- Create Timeseries Statement

```
CREATE TIMESERIES <FullPath> WITH <AttributeClauses>
alias
    : LR_BRACKET ID RR_BRACKET
    ;
attributeClauses
    : DATATYPE OPERATOR_EQ dataType COMMA ENCODING OPERATOR_EQ encoding
    (COMMA (COMPRESSOR | COMPRESSION) OPERATOR_EQ compressor=propertyValue)?
    (COMMA property)*
    tagClause
    attributeClause
    ;
attributeClause
    : (ATTRIBUTES LR_BRACKET property (COMMA property)* RR_BRACKET)?
    ;
tagClause
    : (TAGS LR_BRACKET property (COMMA property)* RR_BRACKET)?
    ;
DataTypeValue: BOOLEAN | DOUBLE | FLOAT | INT32 | INT64 | TEXT
EncodingValue: GORILLA | PLAIN | RLE | TS_2DIFF | REGULAR
CompressorValue: UNCOMPRESSED | SNAPPY
Eg: CREATE TIMESERIES root.ln.wf01.wt01.status WITH DATATYPE=BOOLEAN, ENCODING=PLAIN
Eg: CREATE TIMESERIES root.ln.wf01.wt01.temperature WITH DATATYPE=FLOAT, ENCODING=RLE
Eg: CREATE TIMESERIES root.ln.wf01.wt01.temperature WITH DATATYPE=FLOAT, ENCODING=RLE, COMPRESSOR=SNAPPY,
MAX_POINT_NUMBER=3
Eg: create timeseries root.turbine.d0.s0(temperature) with datatype=FLOAT, encoding=RLE, compression=SNAPPY
tags(unit=f, description='turbine this is a test1') attributes(H_Alarm=100, M_Alarm=50)
Note: Datatype and encoding type must be corresponding. Please check Chapter 3 Encoding Section for details.
```

- Delete Timeseries Statement

```
DELETE TIMESERIES <PrefixPath> [COMMA <PrefixPath>]*
Eg: IoTDB > DELETE TIMESERIES root.ln.wf01.wt01.status
Eg: IoTDB > DELETE TIMESERIES root.ln.wf01.wt01.status, root.ln.wf01.wt01.temperature
Eg: IoTDB > DELETE TIMESERIES root.ln.wf01.wt01.*
```

- Alter Timeseries Statement

```
ALTER TIMESERIES fullPath alterClause
alterClause
    : RENAME beforeName=ID TO currentName=ID
    | SET property (COMMA property)*
    | DROP ID (COMMA ID)*
    | ADD TAGS property (COMMA property)*
    | ADD ATTRIBUTES property (COMMA property)*
    | UPSERT tagClause attributeClause
    ;
attributeClause
    : (ATTRIBUTES LR_BRACKET property (COMMA property)* RR_BRACKET)?
    ;
```

```
tagClause
    : (TAGS LR_BRACKET property (COMMA property)* RR_BRACKET)?
    ;
Eg: ALTER timeseries root.turbine.d1.s1 RENAME tag1 TO newTag1
Eg: ALTER timeseries root.turbine.d1.s1 SET tag1=newV1, attr1=newV1
Eg: ALTER timeseries root.turbine.d1.s1 DROP tag1, tag2
Eg: ALTER timeseries root.turbine.d1.s1 ADD TAGS tag3=v3, tag4=v4
Eg: ALTER timeseries root.turbine.d1.s1 ADD ATTRIBUTES attr3=v3, attr4=v4
EG: ALTER timeseries root.turbine.d1.s1 UPSERT TAGS(tag2=newV2, tag3=v3) ATTRIBUTES(attr3=v3, attr4=v4)
```

- Show All Timeseries Statement

```
SHOW TIMESERIES
Eg: IoTDB > SHOW TIMESERIES
Note: This statement can only be used in IoTDB Client. If you need to show all timeseries in JDBC, please use
`DataBaseMetadata` interface.
```

- Show Specific Timeseries Statement

```
SHOW TIMESERIES <Path>
Eg: IoTDB > SHOW TIMESERIES root
Eg: IoTDB > SHOW TIMESERIES root.ln
Eg: IoTDB > SHOW TIMESERIES root.ln.*.*.status
Eg: IoTDB > SHOW TIMESERIES root.ln.wf01.wt01.status
Note: The path can be prefix path, star path or timeseries path
Note: This statement can be used in IoTDB Client and JDBC.
```

- Show Specific Timeseries Statement with where clause

```
SHOW TIMESERIES prefixPath? showWhereClause?
showWhereClause
    : WHERE (property | containsExpression)
    ;
containsExpression
    : name=ID OPERATOR_CONTAINS value=propertyValue
    ;

Eg: show timeseries root.ln where unit='c'
Eg: show timeseries root.ln where description contains 'test1'
```

- Show Specific Timeseries Statement with where clause start from offset and limit the total number of result

```
SHOW TIMESERIES prefixPath? showWhereClause? limitClause?

showWhereClause
    : WHERE (property | containsExpression)
    ;
containsExpression
    : name=ID OPERATOR_CONTAINS value=propertyValue
```

```
    ;
limitClause
    : LIMIT INT offsetClause?
    | offsetClause? LIMIT INT
    ;

Eg: show timeseries root.ln where unit='c'
Eg: show timeseries root.ln where description contains 'test1'
Eg: show timeseries root.ln where unit='c' limit 10 offset 10
```

- Show Storage Group Statement

```
SHOW STORAGE GROUP
Eg: IoTDB > SHOW STORAGE GROUP
Note: This statement can be used in IoTDB Client and JDBC.
```

- Show Specific Storage Group Statement

```
SHOW STORAGE GROUP <PrefixPath>
Eg: IoTDB > SHOW STORAGE GROUP root.*
Eg: IoTDB > SHOW STORAGE GROUP root.ln
Note: The path can be prefix path or star path.
Note: This statement can be used in IoTDB Client and JDBC.
```

- Show Merge Status Statement

```
SHOW MERGE
Eg: IoTDB > SHOW MERGE
Note: This statement can be used in IoTDB Client and JDBC.
```

- Count Timeseries Statement

```
COUNT TIMESERIES <Path>
Eg: IoTDB > COUNT TIMESERIES root
Eg: IoTDB > COUNT TIMESERIES root.ln
Eg: IoTDB > COUNT TIMESERIES root.ln.*.*.status
Eg: IoTDB > COUNT TIMESERIES root.ln.wf01.wt01.status
Note: The path can be prefix path, star path or timeseries path.
Note: This statement can be used in IoTDB Client and JDBC.
```

```
COUNT TIMESERIES <Path> GROUP BY LEVEL=<INTEGER>
Eg: IoTDB > COUNT TIMESERIES root GROUP BY LEVEL=1
Eg: IoTDB > COUNT TIMESERIES root.ln GROUP BY LEVEL=2
Eg: IoTDB > COUNT TIMESERIES root.ln.wf01 GROUP BY LEVEL=3
Note: The path can be prefix path or timeseries path.
Note: This statement can be used in IoTDB Client and JDBC.
```

- Count Nodes Statement

```
COUNT NODES <Path> LEVEL=<INTEGER>
Eg: IoTDB > COUNT NODES root LEVEL=2
Eg: IoTDB > COUNT NODES root.ln LEVEL=2
Eg: IoTDB > COUNT NODES root.ln.* LEVEL=3
Eg: IoTDB > COUNT NODES root.ln.wf01 LEVEL=3
Note: The path can be prefix path or timeseries path.
Note: This statement can be used in IoTDB Client and JDBC.
```

- Show All Devices Statement

```
SHOW Devices
Eg: IoTDB > SHOW Devices
Note: This statement can be used in IoTDB Client and JDBC.
```

- Show Specific Devices Statement

```
SHOW DEVICES <PrefixPath>
Eg: IoTDB > SHOW DEVICES root
Eg: IoTDB > SHOW DEVICES root.ln
Eg: IoTDB > SHOW DEVICES root.*.wf01
Note: The path can be prefix path or star path.
Note: This statement can be used in IoTDB Client and JDBC.
```

- Show Child Paths of Root Statement

```
SHOW CHILD PATHS
Eg: IoTDB > SHOW CHILD PATHS
Note: This statement can be used in IoTDB Client and JDBC.
```

- Show Child Paths Statement

```
SHOW CHILD PATHS <Path>
Eg: IoTDB > SHOW CHILD PATHS root
Eg: IoTDB > SHOW CHILD PATHS root.ln
Eg: IoTDB > SHOW CHILD PATHS root.*.wf01
Eg: IoTDB > SHOW CHILD PATHS root.ln.wf*
Note: The path can be prefix path or star path, the nodes can be in a "prefix + star" format.
Note: This statement can be used in IoTDB Client and JDBC.
```

- Create snapshot for schema

```
CREATE SNAPSHOT FOR SCHEMA
```

Data Management Statement

- Insert Record Statement

```
INSERT INTO <PrefixPath> LPAREN TIMESTAMP COMMA <Sensor> [COMMA <Sensor>]* RPAREN VALUES LPAREN <TimeValue>,
<PointValue> [COMMA <PointValue>]* RPAREN
Sensor : Identifier
Eg: IoTDB > INSERT INTO root.ln.wf01.wt01(timestamp,status) values(1509465600000,true)
Eg: IoTDB > INSERT INTO root.ln.wf01.wt01(timestamp,status) VALUES(NOW(), false)
Eg: IoTDB > INSERT INTO root.ln.wf01.wt01(timestamp,temperature) VALUES(2017-11-
01T00:17:00.000+08:00,24.22028)
Eg: IoTDB > INSERT INTO root.ln.wf01.wt01(timestamp, status, temperature) VALUES (1509466680000, false,
20.060787);
Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
Note: The order of Sensor and PointValue need one-to-one correspondence
```

- Delete Record Statement

```
DELETE FROM <PrefixPath> [COMMA <PrefixPath>]* [WHERE <WhereClause>]?
WhereClause : <Condition> [(AND) <Condition>]*
Condition  : <TimeExpr> [(AND) <TimeExpr>]*
TimeExpr : TIME PrecedenceEqualOperator (<TimeValue> | <RelativeTime>)
Eg: DELETE FROM root.ln.wf01.wt01.temperature WHERE time > 2016-01-05T00:15:00+08:00 and time < 2017-11-
1T00:05:00+08:00
Eg: DELETE FROM root.ln.wf01.wt01.status, root.ln.wf01.wt01.temperature WHERE time < NOW()
Eg: DELETE FROM root.ln.wf01.wt01.* WHERE time >= 1509466140000
```

- Select Record Statement

```
SELECT <SelectClause> FROM <FromClause> [WHERE <WhereClause>]?
SelectClause : <SelectPath> (COMMA <SelectPath>)*
SelectPath : <FUNCTION> LPAREN <Path> RPAREN | <Path>
FUNCTION : 'COUNT' , 'MIN_TIME', 'MAX_TIME', 'MIN_VALUE', 'MAX_VALUE'
FromClause : <PrefixPath> (COMMA <PrefixPath>)?
WhereClause : <Condition> [(AND | OR) <Condition>]*
Condition  : <Expression> [(AND | OR) <Expression>]*
Expression : [NOT | !]? <TimeExpr> | [NOT | !]? <SensorExpr>
TimeExpr : TIME PrecedenceEqualOperator (<TimeValue> | <RelativeTime>)
RelativeTimeDurationUnit = Integer ('Y'|'MO'|'W'|'D'|'H'|'M'|'S'|'MS'|'US'|'NS')
RelativeTime : (now() | <TimeValue>) [(+|-) RelativeTimeDurationUnit]+
SensorExpr : (<Timeseries> | <Path>) PrecedenceEqualOperator <PointValue>
Eg: IoTDB > SELECT status, temperature FROM root.ln.wf01.wt01 WHERE temperature < 24 and time > 2017-11-1
0:13:00
Eg. IoTDB > SELECT * FROM root
Eg. IoTDB > SELECT * FROM root where time > now() - 5m
Eg. IoTDB > SELECT * FROM root.ln.*.wf*
Eg. IoTDB > SELECT COUNT(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature < 25
Eg. IoTDB > SELECT MIN_TIME(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature < 25
Eg. IoTDB > SELECT MAX_TIME(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature > 24
Eg. IoTDB > SELECT MIN_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature > 23
Eg. IoTDB > SELECT MAX_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature < 25
Eg. IoTDB > SELECT COUNT(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature < 25 GROUP BY
LEVEL=1
Note: the statement needs to satisfy this constraint: <Path>(SelectClause) + <PrefixPath>(FromClause) =
<Timeseries>
Note: If the <SensorExpr>(WhereClause) is started with <Path> and not with ROOT, the statement needs to
satisfy this constraint: <PrefixPath>(FromClause) + <Path>(SensorExpr) = <Timeseries>
Note: In Version 0.7.0, if <WhereClause> includes `OR`, time filter can not be used.
```

```
Note: There must be a space on both sides of the plus and minus operator appearing in the time expression
```

- Group By Statement

```
SELECT <SelectClause> FROM <FromClause> WHERE  <WhereClause> GROUP BY <GroupByTimeClause>
SelectClause : <Function> [COMMA < Function >]*
Function : <AggregationFunction> LPAREN <Path> RPAREN
FromClause : <PrefixPath>
WhereClause : <Condition> [(AND | OR) <Condition>]*
Condition  : <Expression> [(AND | OR) <Expression>]*
Expression : [NOT | !]? <TimeExpr> | [NOT | !]? <SensorExpr>
TimeExpr : TIME PrecedenceEqualOperator (<TimeValue> | <RelativeTime>)
RelativeTimeDurationUnit = Integer ('Y'|'MO'|'W'|'D'|'H'|'M'|'S'|'MS'|'US'|'NS')
RelativeTime : (now() | <TimeValue>) [(+|-) RelativeTimeDurationUnit]+
SensorExpr : (<Timeseries> | <Path>) PrecedenceEqualOperator <PointValue>
GroupByTimeClause : LPAREN <TimeInterval> COMMA <TimeUnit> (COMMA <TimeUnit>)? RPAREN
TimeInterval: LSBRACKET <TimeValue> COMMA <TimeValue> RRBRACKET | LRBRACKET <TimeValue> COMMA <TimeValue>
RSBRACKET
TimeUnit : Integer <DurationUnit>
DurationUnit : "ms" | "s" | "m" | "h" | "d" | "w"
Eg: SELECT COUNT(status), COUNT(temperature) FROM root.ln.wf01.wt01 where temperature < 24 GROUP
BY([1509465720000, 1509466380000), 5m)
Eg: SELECT COUNT(status), COUNT(temperature) FROM root.ln.wf01.wt01 where temperature < 24 GROUP
BY((1509465720000, 1509466380000], 5m)
Eg. SELECT COUNT (status), MAX_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE time < 1509466500000 GROUP
BY([1509465720000, 1509466380000), 5m, 10m)
Eg. SELECT MIN_TIME(status), MIN_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE temperature < 25 GROUP BY
([1509466140000, 1509466380000), 3m, 5ms)
Eg. SELECT MIN_TIME(status), MIN_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE temperature < 25 GROUP BY
((1509466140000, 1509466380000], 3m, 5ms)
Note: the statement needs to satisfy this constraint: <Path>(SelectClause) + <PrefixPath>(FromClause) =
<Timeseries>
Note: If the <SensorExpr>(WhereClause) is started with <Path> and not with ROOT, the statement needs to
satisfy this constraint: <PrefixPath>(FromClause) + <Path>(SensorExpr) = <Timeseries>
Note: <TimeValue>(TimeInterval) needs to be greater than 0
Note: First <TimeValue>(TimeInterval) in needs to be smaller than second <TimeValue>(TimeInterval)
Note: <TimeUnit> needs to be greater than 0
Note: Third <TimeUnit> if set shouldn't be smaller than second <TimeUnit>
```

- Fill Statement

```
SELECT <SelectClause> FROM <FromClause> WHERE <WhereClause> FILL <FillClause>
SelectClause : <Path> [COMMA <Path>]*
FromClause : < PrefixPath > [COMMA < PrefixPath >]*
WhereClause : <WhereExpression>
WhereExpression : TIME EQUAL <TimeValue>
FillClause : LPAREN <TypeClause> [COMMA <TypeClause>]* RPAREN
TypeClause : <Int32Clause> | <Int64Clause> | <FloatClause> | <DoubleClause> | <BoolClause> | <TextClause>
Int32Clause: INT32 LBRACKET (<LinearClause> | <PreviousClause>)  RBRACKET
Int64Clause: INT64 LBRACKET (<LinearClause> | <PreviousClause>)  RBRACKET
FloatClause: FLOAT LBRACKET (<LinearClause> | <PreviousClause>)  RBRACKET
DoubleClause: DOUBLE LBRACKET (<LinearClause> | <PreviousClause>)  RBRACKET
BoolClause: BOOLEAN LBRACKET (<LinearClause> | <PreviousClause>)  RBRACKET
TextClause: TEXT LBRACKET (<LinearClause> | <PreviousClause>)  RBRACKET
```

```
14. PreviousClause : PREVIOUS [COMMA <ValidPreviousTime>]?
15. LinearClause : LINEAR [COMMA <ValidPreviousTime> COMMA <ValidBehindTime>]?
16. ValidPreviousTime, ValidBehindTime: <TimeUnit>
17. TimeUnit : Integer <DurationUnit>
18. DurationUnit : "ms" | "s" | "m" | "h" | "d" | "w"
19. Eg: SELECT temperature FROM root.ln.wf01.wt01 WHERE time = 2017-11-01T16:37:50.000 FILL(float[previous, 1m])
    Eg: SELECT temperature,status FROM root.ln.wf01.wt01 WHERE time = 2017-11-01T16:37:50.000 FILL (float[linear,
20. 1m, 1m], boolean[previous, 1m])
    Eg: SELECT temperature,status,hardware FROM root.ln.wf01.wt01 WHERE time = 2017-11-01T16:37:50.000 FILL
21. (float[linear, 1m, 1m], boolean[previous, 1m], text[previous])
    Eg: SELECT temperature,status,hardware FROM root.ln.wf01.wt01 WHERE time = 2017-11-01T16:37:50.000 FILL
22. (float[linear], boolean[previous, 1m], text[previous])
    Note: the statement needs to satisfy this constraint: <PrefixPath>(FromClause) + <Path>(SelectClause) =
23. <Timeseries>
24. Note: Integer in <TimeUnit> needs to be greater than 0
```

- Group By Fill Statement

```
    SELECT <SelectClause> FROM <FromClause> WHERE  <WhereClause> GROUP BY <GroupByClause> (FILL
 1. <GROUPBYFillClause>)?
 2. GroupByClause : LPAREN <TimeInterval> COMMA <TimeUnit> RPAREN
 3. GROUPBYFillClause : LPAREN <TypeClause> RPAREN
    TypeClause : <AllClause> | <Int32Clause> | <Int64Clause> | <FloatClause> | <DoubleClause> | <BoolClause> |
 4. <TextClause>
 5. AllClause: ALL LBRACKET (<PreviousUntilLastClause> | <PreviousClause>)  RBRACKET
 6. Int32Clause: INT32 LBRACKET (<PreviousUntilLastClause> | <PreviousClause>)  RBRACKET
 7. Int64Clause: INT64 LBRACKET (<PreviousUntilLastClause> | <PreviousClause>)  RBRACKET
 8. FloatClause: FLOAT LBRACKET (<PreviousUntilLastClause> | <PreviousClause>)  RBRACKET
 9. DoubleClause: DOUBLE LBRACKET (<PreviousUntilLastClause> | <PreviousClause>)  RBRACKET
10. BoolClause: BOOLEAN LBRACKET (<PreviousUntilLastClause> | <PreviousClause>)  RBRACKET
11. TextClause: TEXT LBRACKET (<PreviousUntilLastClause> | <PreviousClause>)  RBRACKET
12. PreviousClause : PREVIOUS
13. PreviousUntilLastClause : PREVIOUSUNTILLAST
14. Eg: SELECT last_value(temperature) FROM root.ln.wf01.wt01 GROUP BY([20, 100), 5m) FILL (float[PREVIOUS])
15. Eg: SELECT last_value(temperature) FROM root.ln.wf01.wt01 GROUP BY((15, 100], 5m) FILL (float[PREVIOUS])
16. Eg: SELECT last_value(power) FROM root.ln.wf01.wt01 GROUP BY([20, 100), 5m) FILL (int32[PREVIOUSUNTILLAST])
    Eg: SELECT last_value(power) FROM root.ln.wf01.wt01 GROUP BY([20, 100), 5m) FILL (int32[PREVIOUSUNTILLAST,
17. 5m])
    Eg: SELECT last_value(temperature), last_value(power) FROM root.ln.wf01.wt01 GROUP BY([20, 100), 5m) FILL
18. (ALL[PREVIOUS])
    Eg: SELECT last_value(temperature), last_value(power) FROM root.ln.wf01.wt01 GROUP BY([20, 100), 5m) FILL
19. (ALL[PREVIOUS, 5m])
20. Note: In group by fill, sliding step is not supported in group by clause
21. Note: Now, only last_value aggregation function is supported in group by fill.
22. Note: Linear fill is not supported in group by fill.
```

- Order by time Statement

```
    SELECT <SelectClause> FROM <FromClause> WHERE  <WhereClause> GROUP BY <GroupByClause> (FILL
 1. <GROUPBYFillClause>)? orderByTimeClause?
 2. orderByTimeClause: order by time (asc | desc)?
 3.
    Eg: SELECT last_value(temperature) FROM root.ln.wf01.wt01 GROUP BY([20, 100), 5m) FILL (float[PREVIOUS]) order
 4. by time desc
 5. Eg: SELECT * from root order by time desc
```

```
Eg: SELECT * from root order by time desc align by device
Eg: SELECT * from root order by time desc disable align
Eg: SELECT last * from root order by time desc
```

- Limit Statement

```
SELECT <SelectClause> FROM <FromClause> [WHERE <WhereClause>] [<LIMITClause>] [<SLIMITClause>]
SelectClause : [<Path> | Function]+
Function : <AggregationFunction> LPAREN <Path> RPAREN
FromClause : <Path>
WhereClause : <Condition> [(AND | OR) <Condition>]*
Condition : <Expression> [(AND | OR) <Expression>]*
Expression: [NOT|!]?<TimeExpr> | [NOT|!]?<SensorExpr>
TimeExpr : TIME PrecedenceEqualOperator (<TimeValue> | <RelativeTime>)
RelativeTimeDurationUnit = Integer ('Y'|'MO'|'W'|'D'|'H'|'M'|'S'|'MS'|'US'|'NS')
RelativeTime : (now() | <TimeValue>) [(+|-) RelativeTimeDurationUnit]+
SensorExpr : (<Timeseries>|<Path>) PrecedenceEqualOperator <PointValue>
LIMITClause : LIMIT <N> [OFFSETClause]?
N : Integer
OFFSETClause : OFFSET <OFFSETValue>
OFFSETValue : Integer
SLIMITClause : SLIMIT <SN> [SOFFSETClause]?
SN : Integer
SOFFSETClause : SOFFSET <SOFFSETValue>
SOFFSETValue : Integer
Eg: IoTDB > SELECT status, temperature FROM root.ln.wf01.wt01 WHERE temperature < 24 and time > 2017-11-1
0:13:00 LIMIT 3 OFFSET 2
Eg. IoTDB > SELECT COUNT (status), MAX_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE time < 1509466500000
GROUP BY([1509465720000, 1509466380000], 5m) LIMIT 3
Note: N, OFFSETValue, SN and SOFFSETValue must be greater than 0.
Note: The order of <LIMITClause> and <SLIMITClause> does not affect the grammatical correctness.
Note: <FillClause> can not use <LIMITClause> but not <SLIMITClause>.
```

- Align By Device Statement

```
AlignbyDeviceClause : ALIGN BY DEVICE

Rules:
1. Both uppercase and lowercase are ok.
Correct example: select * from root.sg1 align by device
Correct example: select * from root.sg1 ALIGN BY DEVICE

2. AlignbyDeviceClause can only be used at the end of a query statement.
Correct example: select * from root.sg1 where time > 10 align by device
Wrong example: select * from root.sg1 align by device where time > 10

3. The paths of the SELECT clause can only be single level. In other words, the paths of the SELECT clause can
only be measurements or STAR, without DOT.
Correct example: select s0,s1 from root.sg1.* align by device
Correct example: select s0,s1 from root.sg1.d0, root.sg1.d1 align by device
Correct example: select * from root.sg1.* align by device
Correct example: select * from root align by device
```

Correct example: select s0,s1,* from root.*.* align by device
Wrong example: select d0.s1, d0.s2, d1.s0 from root.sg1 align by device
Wrong example: select *.s0, *.s1 from root.* align by device
Wrong example: select *.*.* from root align by device

4. The data types of the same measurement column should be the same across devices.
Note that when it comes to aggregated paths, the data type of the measurement column will reflect
the aggregation function rather than the original timeseries.

Correct example: select s0 from root.sg1.d0,root.sg1.d1 align by device
root.sg1.d0.s0 and root.sg1.d1.s0 are both INT32.

Correct example: select count(s0) from root.sg1.d0,root.sg1.d1 align by device
count(root.sg1.d0.s0) and count(root.sg1.d1.s0) are both INT64.

Wrong example: select s0 from root.sg1.d0, root.sg2.d3 align by device
root.sg1.d0.s0 is INT32 while root.sg2.d3.s0 is FLOAT.

5. The display principle of the result table is that all the columns (no matther whther a column has has existing data) will be shown, with nonexistent cells being null. Besides, the select clause support const column (e.g., 'a', '123' etc..).
For example, "select s0,s1,s2,'abc',s1,s2 from root.sg.d0, root.sg.d1, root.sg.d2 align by device". Suppose that the actual existing timeseries are as follows:
- root.sg.d0.s0
- root.sg.d0.s1
- root.sg.d1.s0

Then you could expect a table like:

Time	Device	s0	s1	s2	'abc'	s1	s2
---	---	---	---	null	'abc'	---	null
1	root.sg.d0	20	2.5	null	'abc'	2.5	null
2	root.sg.d0	23	3.1	null	'abc'	3.1	null
...	...	...	...	null	'abc'	...	null
1	root.sg.d1	12	null	null	'abc'	null	null
2	root.sg.d1	19	null	null	'abc'	null	null
...	...	...	...	null	'abc'	...	null

Note that the cells of measurement 's0' and device 'root.sg.d1' are all null.

6. The duplicated devices in the prefix paths are neglected.
For example, "select s0,s1 from root.sg.d0,root.sg.d0,root.sg.d1 align by device" is equal to "select s0,s1 from root.sg.d0,root.sg.d1 align by device".
For example. "select s0,s1 from root.sg.*,root.sg.d0 align by device" is equal to "select s0,s1 from root.sg.* align by device".

7. The duplicated measurements in the suffix paths are not neglected.
For example, "select s0,s0,s1 from root.sg.* align by device" is not equal to "select s0,s1 from root.sg.* align by device".

8. Both time predicates and value predicates are allowed in Where Clause. The paths of the value predicates can be the leaf node or full path started with ROOT. And wildcard is not allowed here. For example:
- select * from root.sg.* where time = 1 align by device
- select * from root.sg.* where s0 < 100 align by device
- select * from root.sg.* where time < 20 AND s0 > 50 align by device

```
- select * from root.sg.d0 where root.sg.d0.s0 = 15 align by device

9. More correct examples:
   - select * from root.vehicle align by device
   - select s0,s0,s1 from root.vehicle.* align by device
   - select s0,s1 from root.vehicle.* limit 10 offset 1 align by device
   - select * from root.vehicle slimit 10 soffset 2 align by device
   - select * from root.vehicle where time > 10 align by device
   - select * from root.vehicle.* where time < 10 AND s0 > 25 align by device
   - select * from root.vehicle where root.vehicle.d0.s0>0 align by device
   - select count(*) from root.vehicle align by device
   - select sum(*) from root.vehicle GROUP BY (20ms,0,[2,50]) align by device
   - select * from root.vehicle where time = 3 Fill(int32[previous, 5ms]) align by device
```

- Disable Align Statement

```
Disable Align Clause: DISABLE ALIGN

Rules:
1. Both uppercase and lowercase are ok.
Correct example: select * from root.sg1 disable align
Correct example: select * from root.sg1 DISABLE ALIGN

2. Disable Align Clause can only be used at the end of a query statement.
Correct example: select * from root.sg1 where time > 10 disable align
Wrong example: select * from root.sg1 disable align where time > 10

3. Disable Align Clause cannot be used with Aggregation, Fill Statements, Group By or Group By Device
Statements, but can with Limit Statements.
Correct example: select * from root.sg1 limit 3 offset 2 disable align
Correct example: select * from root.sg1 slimit 3 soffset 2 disable align
Wrong example: select count(s0),count(s1) from root.sg1.d1 disable align
Wrong example: select * from root.vehicle where root.vehicle.d0.s0>0 disable align
Wrong example: select * from root.vehicle align by device disable align

4. The display principle of the result table is that only when the column (or row) has existing data will the
column (or row) be shown, with nonexistent cells being empty.

You could expect a table like:
| Time | root.sg.d0.s1 | Time | root.sg.d0.s2 | Time | root.sg.d1.s1 |
| ---  | ---           | ---  | ---           | ---  | ---           |
|  1   | 100           | 20   | 300           | 400  | 600           |
|  2   | 300           | 40   | 800           | 700  | 900           |
|  4   | 500           |      |               | 800  | 1000          |
|      |               |      |               | 900  | 8000          |

5. More correct examples:
   - select * from root.vehicle disable align
   - select s0,s0,s1 from root.vehicle.* disable align
   - select s0,s1 from root.vehicle.* limit 10 offset 1 disable align
   - select * from root.vehicle slimit 10 soffset 2 disable align
   - select * from root.vehicle where time > 10 disable align
```

- Select Last Record Statement

The LAST function returns the last time-value pair of the given timeseries. Currently filters are not supported in LAST queries.

```
SELECT LAST <SelectClause> FROM <FromClause>
Select Clause : <Path> [COMMA <Path>]*
FromClause : < PrefixPath > [COMMA < PrefixPath >]*
WhereClause : <TimeExpr> [(AND | OR) <TimeExpr>]*
TimeExpr : TIME PrecedenceEqualOperator (<TimeValue> | <RelativeTime>)

Eg. SELECT LAST s1 FROM root.sg.d1
Eg. SELECT LAST s1, s2 FROM root.sg.d1
Eg. SELECT LAST s1 FROM root.sg.d1, root.sg.d2
Eg. SELECT LAST s1 FROM root.sg.d1 where time > 100
Eg. SELECT LAST s1, s2 FROM root.sg.d1 where time >= 500

Rules:
1. the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>

2. SELECT LAST only supports time filter that contains '>' or '>=' currently.

3. The result set of last query will always be displayed in a fixed three column table format.
For example, "select last s1, s2 from root.sg.d1, root.sg.d2", the query result would be:

| Time | Path         | Value |
| ---  | ------------ | ----- |
|  5   | root.sg.d1.s1| 100   |
|  2   | root.sg.d1.s2| 400   |
|  4   | root.sg.d2.s1| 250   |
|  9   | root.sg.d2.s2| 600   |

4. It is not supported to use "diable align" in LAST query.
```

- As Statement

As statement assigns an alias to time seires queried in SELECT statement

```
You can use as statement in all queries, but some rules are restricted about wildcard.

1. Raw data query
select s1 as speed, s2 as temperature from root.sg.d1

The result set will be like:
| Time | speed | temperature |
|  ... |  ...  |     ....    |

2. Aggregation query
select count(s1) as s1_num, max_value(s2) as s2_max from root.sg.d1

3. Down-frequence query
select count(s1) as s1_num from root.sg.d1 group by ([100,500), 80ms)
```

```
15.
16. 4. Align by device query
17. select s1 as speed, s2 as temperature from root.sg.d1 align by device
18.
19. select count(s1) as s1_num, count(s2), count(s3) as s3_num from root.sg.d2 align by device
20.
21. 5. Last Record query
22. select last s1 as speed, s2 from root.sg.d1
23.
24. Rules:
25. 1. In addition to Align by device query, each AS statement has to corresponding to one time series exactly.
26.
27. E.g. select s1 as temperature from root.sg.*
28.
29. At this time if `root.sg.*` includes more than one device, then an exception will be thrown。
30.
    2. In align by device query, the prefix path that each AS statement corresponding to can includes multiple
31. device, but the suffix path can only be single sensor.
32.
33. E.g. select s1 as temperature from root.sg.*
34.
35. In this situation, it will be show correctly even if multiple devices are selected.
36.
37. E.g. select * as temperature from root.sg.d1
38.
39. In this situation, it will throws an exception if * corresponds to multiple sensors.
```

Database Management Statement

- Create User

```
1. CREATE USER <userName> <password>;
2. userName:=identifier
3. password:=string
4. Eg: IoTDB > CREATE USER thulab 'pwd';
```

- Delete User

```
1. DROP USER <userName>;
2. userName:=identifier
3. Eg: IoTDB > DROP USER xiaoming;
```

- Create Role

```
1. CREATE ROLE <roleName>;
2. roleName:=identifie
3. Eg: IoTDB > CREATE ROLE admin;
```

- Delete Role

```
DROP ROLE <roleName>;
roleName:=identifier
Eg: IoTDB > DROP ROLE admin;
```

- Grant User Privileges

```
GRANT USER <userName> PRIVILEGES <privileges> ON <nodeName>;
userName:=identifier
nodeName:=identifier (DOT identifier)*
privileges:= string (COMMA string)*
Eg: IoTDB > GRANT USER tempuser PRIVILEGES 'DELETE_TIMESERIES' on root.ln;
```

- Grant Role Privileges

```
GRANT ROLE <roleName> PRIVILEGES <privileges> ON <nodeName>;
privileges:= string (COMMA string)*
roleName:=identifier
nodeName:=identifier (DOT identifier)*
Eg: IoTDB > GRANT ROLE temprole PRIVILEGES 'DELETE_TIMESERIES' ON root.ln;
```

- Grant User Role

```
GRANT <roleName> TO <userName>;
roleName:=identifier
userName:=identifier
Eg: IoTDB > GRANT temprole TO tempuser;
```

- Revoke User Privileges

```
REVOKE USER <userName> PRIVILEGES <privileges> ON <nodeName>;
privileges:= string (COMMA string)*
userName:=identifier
nodeName:=identifier (DOT identifier)*
Eg: IoTDB > REVOKE USER tempuser PRIVILEGES 'DELETE_TIMESERIES' on root.ln;
```

- Revoke Role Privileges

```
REVOKE ROLE <roleName> PRIVILEGES <privileges> ON <nodeName>;
privileges:= string (COMMA string)*
roleName:= identifier
nodeName:=identifier (DOT identifier)*
Eg: IoTDB > REVOKE ROLE temprole PRIVILEGES 'DELETE_TIMESERIES' ON root.ln;
```

- Revoke Role From User

```
REVOKE <roleName> FROM <userName>;
roleName:=identifier
userName:=identifier
```

```
4. Eg: IoTDB > REVOKE temprole FROM tempuser;
```

- List Users

```
1. LIST USER
2. Eg: IoTDB > LIST USER
```

- List Roles

```
1. LIST ROLE
2. Eg: IoTDB > LIST ROLE
```

- List Privileges

```
1. LIST PRIVILEGES USER  <username> ON <path>;
2. username:=identifier
3. path='root' (DOT identifier)*
4. Eg: IoTDB > LIST PRIVILEGES USER sgcc_wirte_user ON root.sgcc;
```

- List Privileges of Roles(On Specific Path)

```
1. LIST PRIVILEGES ROLE <roleName> ON <path>;
2. roleName:=identifier
3. path='root' (DOT identifier)*
4. Eg: IoTDB > LIST PRIVILEGES ROLE wirte_role ON root.sgcc;
```

- List Privileges of Users

```
1. LIST USER PRIVILEGES <username> ;
2. username:=identifier
3. Eg: IoTDB > LIST USER PRIVILEGES tempuser;
```

- List Privileges of Roles

```
1. LIST ROLE PRIVILEGES <roleName>
2. roleName:=identifier
3. Eg: IoTDB > LIST ROLE PRIVILEGES actor;
```

- List Roles of Users

```
1. LIST ALL ROLE OF USER <username> ;
2. username:=identifier
3. Eg: IoTDB > LIST ALL ROLE OF USER tempuser;
```

- List Users of Role

```
LIST ALL USER OF ROLE <roleName>;
roleName:=identifier
Eg: IoTDB > LIST ALL USER OF ROLE roleuser;
```

- Alter Password

```
ALTER USER <username> SET PASSWORD <password>;
roleName:=identifier
password:=identifier
Eg: IoTDB > ALTER USER tempuser SET PASSWORD 'newpwd';
```

Functions

- COUNT

The COUNT function returns the value number of timeseries(one or more) non-null values selected by the SELECT statement. The result is a signed 64-bit integer. If there are no matching rows, COUNT () returns 0.

```
SELECT COUNT(Path) (COMMA COUNT(Path))* FROM <FromClause> [WHERE <WhereClause>]?
Eg. SELECT COUNT(status), COUNT(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature < 24
Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- FIRST_VALUE(Rename from `FIRST` at `V0.10.0`)

The FIRST_VALUE function returns the first point value of the choosen timeseries(one or more).

```
SELECT FIRST_VALUE (Path) (COMMA FIRST_VALUE (Path))* FROM <FromClause> [WHERE <WhereClause>]?
Eg. SELECT FIRST_VALUE (status), FIRST_VALUE (temperature) FROM root.ln.wf01.wt01 WHERE
root.ln.wf01.wt01.temperature < 24
Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- LAST_VALUE

The LAST_VALUE function returns the last point value of the choosen timeseries(one or more).

```
SELECT LAST_VALUE (Path) (COMMA LAST_VALUE (Path))* FROM <FromClause> [WHERE <WhereClause>]?
Eg. SELECT LAST_VALUE (status), LAST_VALUE (temperature) FROM root.ln.wf01.wt01 WHERE
root.ln.wf01.wt01.temperature < 24
Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- MAX_TIME

The MAX_TIME function returns the maximum timestamp of the choosen timeseries(one or more). The result is a signed 64-bit integer, greater than 0.

```
SELECT MAX_TIME (Path) (COMMA MAX_TIME (Path))* FROM <FromClause> [WHERE <WhereClause>]?
Eg. SELECT MAX_TIME(status), MAX_TIME(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature
< 24
Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- MAX_VALUE

The MAX_VALUE function returns the maximum value(lexicographically ordered) of the choosen timeseries (one or more).

```
1. SELECT MAX_VALUE (Path) (COMMA MAX_VALUE (Path))* FROM <FromClause> [WHERE <WhereClause>]?
   Eg. SELECT MAX_VALUE(status), MAX_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE
2. root.ln.wf01.wt01.temperature < 24
3. Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- AVG(Rename from `MEAN` at `V0.9.0`)

The AVG function returns the arithmetic mean value of the choosen timeseries over a specified period of time. The timeseries must be int32, int64, float, double type, and the other types are not to be calculated. The result is a double type number.

```
1. SELECT AVG (Path) (COMMA AVG (Path))* FROM <FromClause> [WHERE <WhereClause>]?
2. Eg. SELECT AVG (temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature < 24
3. Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- MIN_TIME

The MIN_TIME function returns the minimum timestamp of the choosen timeseries(one or more). The result is a signed 64-bit integer, greater than 0.

```
1. SELECT MIN_TIME (Path) (COMMA MIN_TIME (Path))*FROM <FromClause> [WHERE <WhereClause>]?
   Eg. SELECT MIN_TIME(status), MIN_TIME(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature
2. < 24
3. Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- MIN_VALUE

The MIN_VALUE function returns the minimum value(lexicographically ordered) of the choosen timeseries (one or more).

```
1. SELECT MIN_VALUE (Path) (COMMA MIN_VALUE (Path))* FROM <FromClause> [WHERE <WhereClause>]?
   Eg. SELECT MIN_VALUE(status),MIN_VALUE(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature
2. < 24
3. Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

- NOW

The NOW function returns the current timestamp. This function can be used in the data operation statement to represent time. The result is a signed 64-bit integer, greater than 0.

```
1. NOW()
2. Eg. INSERT INTO root.ln.wf01.wt01(timestamp,status) VALUES(NOW(), false)
3. Eg. DELETE FROM root.ln.wf01.wt01.status, root.ln.wf01.wt01.temperature WHERE time < NOW()
4. Eg. SELECT * FROM root WHERE time < NOW()
5. Eg. SELECT COUNT(temperature) FROM root.ln.wf01.wt01 WHERE time < NOW()
```

- SUM

The SUM function returns the sum of the choosen timeseries (one or more) over a specified period of time. The timeseries must be int32, int64, float, double type, and the other types are not to be calculated. The result is a double type number.

```
SELECT SUM(Path) (COMMA SUM(Path))* FROM <FromClause> [WHERE <WhereClause>]?
Eg. SELECT SUM(temperature) FROM root.ln.wf01.wt01 WHERE root.ln.wf01.wt01.temperature < 24
Note: the statement needs to satisfy this constraint: <PrefixPath> + <Path> = <Timeseries>
```

TTL

IoTDB supports storage-level TTL settings, which means it is able to delete old data automatically and periodically. The benefit of using TTL is that hopefully you can control the total disk space usage and prevent the machine from running out of disks. Moreover, the query performance may downgrade as the total number of files goes up and the memory usage also increase as there are more files. Timely removing such files helps to keep at a high query performance level and reduce memory usage. The TTL operations in IoTDB are supported by the following three statements:

- Set TTL

```
SET TTL TO StorageGroupName TTLTime
Eg. SET TTL TO root.group1 3600000
This example means that for data in root.group1, only that of the latest 1 hour will remain, the
older one is removed or made invisible.
Note: TTLTime should be millisecond timestamp. When TTL is set, insertions that fall
out of TTL will be rejected.
```

- Unset TTL

```
UNSET TTL TO StorageGroupName
Eg. UNSET TTL TO root.group1
This example means that data of all time will be accepted in this group.
```

- Show TTL

```
SHOW ALL TTL
SHOW TTL ON StorageGroupNames
Eg.1 SHOW ALL TTL
This example will show TTLs of all storage groups.
Eg.2 SHOW TTL ON root.group1,root.group2,root.group3
This example will show TTLs of the specified 3 groups.
Notice: storage groups without TTL will show a "null"
```

Notice: When you set TTL to some storage groups, data out of the TTL will be made invisible immediately, but because the data files may contain both out-dated and living data or the data files may be being used by queries, the physical removal of data is stale. If you increase or unset TTL just after setting it previously, some

previously invisible data may be seen again, but the physically removed one is lost forever. In other words, different from delete statement, the atomicity of data deletion is not guaranteed for efficiency concerns. So we recommend that you do not change the TTL once it is set or at least do not reset it frequently, unless you are determined to suffer the unpredictability.

- Delete Partition (experimental)

```
DELETE PARTITION StorageGroupName INT(COMMA INT)*
Eg DELETE PARTITION root.sg1 0,1,2
This example will delete the first 3 time partitions of storage group root.sg1.
```

The partitionId can be found in data folders or converted using `timestamp / partitionInterval` .

Performance Tracing

IoTDB supports tracking the execution of query statements by using `TRACING` statements. The number of tsfile files and chunks accessed by the query etc are output through the log file. The default output location is in `./data/tracing` . The performance tracing function is turned off by default. Users can use the TRACING ON/OFF command to turn this function on/off.

```
TRACING ON    // Open performance tracing
TRACING OFF   // Close performance tracing
```

Reference

Identifiers

```
QUOTE := '\'';
DOT := '.';
COLON : ':' ;
COMMA := ',' ;
SEMICOLON := ';' ;
LPAREN := '(' ;
RPAREN := ')' ;
LBRACKET := '[';
RBRACKET := ']';
EQUAL := '=' | '==';
NOTEQUAL := '<>' | '!=';
LESSTHANOREQUALTO := '<=';
LESSTHAN := '<';
GREATERTHANOREQUALTO := '>=';
GREATERTHAN := '>';
DIVIDE := '/';
```

```
PLUS := '+';
MINUS := '-';
STAR := '*';
Letter := 'a'..'z' | 'A'..'Z';
HexDigit := 'a'..'f' | 'A'..'F';
Digit := '0'..'9';
Boolean := TRUE | FALSE | 0 | 1 (case insensitive)
```

```
StringLiteral := ( '\'' ( ~('\'') )* '\'' | '\"' ( ~('\"') )* '\"');
eg. 'abc'
eg. 'abc'
```

```
Integer := ('-' | '+')? Digit+;
eg. 123
eg. -222
```

```
Float := ('-' | '+')? Digit+ DOT Digit+ (('e' | 'E') ('-' | '+')? Digit+)?;
eg. 3.1415
eg. 1.2E10
eg. -1.33
```

```
Identifier := (Letter | '_') (Letter | Digit | '_' | MINUS)*;
eg. a123
eg. _abc123
```

Literals

```
PointValue : Integer | Float | StringLiteral | Boolean
```

```
TimeValue : Integer | DateTime | ISO8601 | NOW()
Note: Integer means timestamp type.

DateTime :
eg. 2016-11-16T16:22:33+08:00
eg. 2016-11-16 16:22:33+08:00
eg. 2016-11-16T16:22:33.000+08:00
eg. 2016-11-16 16:22:33.000+08:00
Note: DateTime Type can support several types, see Chapter 3 Datetime section for details.
```

```
PrecedenceEqualOperator : EQUAL | NOTEQUAL | LESSTHANOREQUALTO | LESSTHAN | GREATERTHANOREQUALTO | GREATERTHAN
```

```
Timeseries : ROOT [DOT <LayerName>]* DOT <SensorName>
LayerName : Identifier
SensorName : Identifier
eg. root.ln.wf01.wt01.status
```

```
eg. root.sgcc.wf03.wt01.temperature
Note: Timeseries must be start with `root`(case insensitive) and end with sensor name.
```

```
PrefixPath : ROOT (DOT <LayerName>)*
LayerName : Identifier | STAR
eg. root.sgcc
eg. root.*
```

```
Path: (ROOT | <LayerName>) (DOT <LayerName>)*
LayerName: Identifier | STAR
eg. root.ln.wf01.wt01.status
eg. root.*.wf01.wt01.status
eg. root.ln.wf01.wt01.*
eg. *.wt01.*
eg. *
```

- Sync Tool
- JMX Tool
- Watermark Tool
- Query History Visualization Tool
- Monitor and Log Tools
- Load External Tsfile
- Performance Tracing Tool

Sync Tool

Data Import

- Chapter 6: System Tools
  - Data Import
- Introduction
- Application Scenario
- Configuration
  - Sync Receiver
  - Sync Sender
- Usage
  - Start Sync Receiver
  - Stop Sync Receiver
  - Start Sync Sender
  - Stop Sync Sender

Introduction

The Sync Tool is an IoTDB suite tool that periodically uploads persistent tsfiles from the sender disk to the receiver and loads them.

On the sender side of the sync, the sync module is a separate process, independent of the IoTDB process. It can be started and closed through a separate script (see Sections `Usage` for details). The frequency cycle of the sync can be set by the user.

On the receiver side of the sync, the sync module is embedded in the engine of IoTDB and is in the same process with IoTDB. The receiver module listens for a separate port, which can be set by the user (see Section `Configuration` for details). Before using it, it needs to set up a whitelist at the sync receiver, which is expressed as a network segment. The receiver only accepts the data transferred from the sender located in the whitelist segment, as detailed in Section `Configuration` .

The sync tool has a many-to-one sender-receiver mode - that is, one sync receiver can receive data from multiple sync senders simultaneously while one sync sender can only send data to one sync receiver.

> Note: Before using the sync tool, the client and server need to be configured separately. The configuration is detailed in Sections Configuration.

Application Scenario

In the case of a factory application, there are usually multiple sub-factories and

multiple general(main) factories. Each sub-factory uses an IoTDB instance to collect data, and then synchronize the data to the general factory for backup or analysis. A general factory can receive data from multiple sub-factories and a sub-factory can also synchronize data to multiple general factories. In this scenario, each IoTDB instance manages different devices.
In the sync module, each sub-factory is a sender, a general factory is a receiver, and senders periodically synchronizes the data to receivers. In the above application scenario, the data of one device can only be collected by one sender, so there is no device overlap between the data synchronized by multiple senders. Otherwise, the application scenario of the sync module is not satisfied.

When there is an abnormal scenario, namely, two or more senders synchronize the data of the same device (whose storage group is set as root.sg) to the same receiver, the root.sg data of the sender containing the device data received later by the receiver will be rejected. Example: the engine 1 synchronizes the storage groups root.sg1 and root.sg2 to the receiver, and the engine 2 synchronizes the storage groups root.sg2 and root.sg3 to the receiver. All of them include the time series root.sg2.d0.s0. If the receiver receives the data of root.sg2.d0.s0 of the sender 1 first, the receiver will reject the data of root.sg2 of the sender 2.

Configuration

Sync Receiver

The parameter configuration of the sync receiver is located in the configuration file `iotdb-engine.properties` of IoTDB, and its directory is `$IOTDB_HOME/conf/iotdb-engine.properties`. In this configuration file, there are four parameters related to the sync receiver. The configuration instructions are as follows:

parameter: is_sync_enable	
Description	Sync function switch, which is configured as true to indicate that the receiver is allowed to receive the data from the sender and load it. When set to false, it means that the receiver is not allowed to receive the data from any sender.
Type	Boolean
Default	false
Modalities for Entry into Force after Modification	Restart receiver

parameter: IP_white_list	
	Set up a white list of sender IP addresses, which is expressed in the form of network segments and separated by commas between multiple network segments. When the sender transfers data to the receiver, only when the IP address of the sender is within the

	network segment set by the whitelist can the receiver allow the sync operation. If the whitelist is empty, the receiver does not allow any sender to sync data. The default receiver accepts all IP sync requests.
Type	String
Default	0.0.0.0/0
Modalities for Entry into Force after Modification	Restart receiver

parameter: sync_server_port	
Description	Sync receiver port to listen. Make sure that the port is not a system reserved port and is not occupied. This paramter is valid only when the parameter is_sync_enable is set to TRUE.
Type	Short Int : [0,65535]
Default	5555
Modalities for Entry into Force after Modification	Restart receiver

Sync Sender

The parameters of the sync sender are configured in a separate configuration file iotdb-sync-client.properties with the installation directory of `$IOTDB_HOME/conf/iotdb-sync-client.properties` . In this configuration file, there are five parameters related to the sync sender. The configuration instructions are as follows:

parameter: server_ip	
Description	IP address of sync receiver.
Type	String
Default	127.0.0.1
Modalities for Entry into Force after Modification	Restart client

parameter: server_port	
Description	Listening port of sync receiver, it is necessary to ensure that the port is consistent with the configuration of the listening port set in receiver.
Type	Short Int : [0,65535]
Default	5555
Modalities for Entry into Force after Modification	Restart client

parameter: sync_period_in_second

parameter: sync_period_in_second	
Description	The period time of sync process, the time unit is second.
Type	Int : [0,2147483647]
Default	600
Modalities for Entry into Force after Modification	Restart client

parameter: iotdb_schema_directory	
Description	The absolute path of the sender’s IoTDB schema file, such as $IOTDB_HOME/data/system/schema/mlog.txt (if the user does not manually set the path of schema metadata, the path is the default path of IoTDB engine). This parameter is not valid by default and is set manually when the user needs it.
Type	String
Modalities for Entry into Force after Modification	Restart client

parameter: sync_storage_groups	
Description	This parameter represents storage groups that participate in the synchronization task, which distinguishes each storage group by comma. If the list is empty, it means that all storage groups participate in synchronization. By default, it is an empty list.
Type	String
Example	root.sg1, root.sg2
Modalities for Entry into Force after Modification	Restart client

parameter: max_number_of_sync_file_retry	
Description	The maximum number of retry when syncing a file to receiver fails.
Type	Int : [0,2147483647]
Example	5
Modalities for Entry into Force after Modification	Restart client

Usage

Start Sync Receiver

```
####################
### Sync Server Configuration
####################

# Whether to open the sync_server_port for receiving data from sync client, the default allowed
is_sync_enable=true

# Sync server port to listen
sync_server_port=5555

# White IP list of Sync client.
# Please use the form of network segment to present the range of IP, for example: 192.168.0.0/16
# If there are more than one IP segment, please separate them by commas
# The default is to allow all IP to sync
ip_white_list=0.0.0.0/0
```

2. Start IoTDB engine, and the sync receiver will start at the same time, and the LOG log will start with the sentence `IoTDB: start SYNC ServerService successfully` indicating the successful start of the return receiver.

Stop Sync Receiver

Stop IoTDB and the sync receiver will be closed at the same time.

Start Sync Sender

1. Set up parameters of sync sender. For example:

```
# Sync receiver server address
server_ip=127.0.0.1

# Sync receiver server port
server_port=5555

# The period time of sync process, the time unit is second.
sync_period_in_second=600

# This parameter represents storage groups that participate in the synchronization task, which distinguishes each storage group by comma.
# If the list is empty, it means that all storage groups participate in synchronization.
# By default, it is empty list.
sync_storage_groups = root.sg1, root.sg2
```

2. Start sync sender Users can use the scripts under the `$IOTDB_HOME/bin` folder to start the sync sender. For Linux and Mac OS X users:

```
Shell >$IOTDB_HOME/bin/start-sync-client.sh
```

For Windows users:

```
Shell >$IOTDB_HOME\bin\start-sync-client.bat
```

```
2019-06-14 16:23:59,762 [main] INFO  org.apache.iotdb.db.conf.IoTDBDescriptor:82 - Start to read config file /Users/litianan/software/iotdb/conf/iotdb-engine.properties
2019-06-14 16:23:59,765 [main] INFO  org.apache.iotdb.db.conf.IoTDBDescriptor:102 - The stat_monitor_detect_freq_sec value is smaller than default, use default value
2019-06-14 16:23:59,765 [main] INFO  org.apache.iotdb.db.conf.IoTDBDescriptor:109 - The stat_monitor_retain_interval_sec value is smaller than default, use default value
2019-06-14 16:23:59,766 [main] INFO  org.apache.iotdb.db.conf.IoTDBDescriptor:248 - Time zone has been set to +08:00
2019-06-14 16:23:59,767 [main] INFO  org.apache.iotdb.db.sync.conf.SyncSenderDescriptor:82 - Start to read sync config file /Users/litianan/software/iotdb/conf/iotdb-sync-client.properties
2019-06-14 16:23:59,789 [pool-1-IoTDB-sync-client-timer-thread-1] INFO  org.apache.iotdb.db.sync.sender.SyncFileManager:98 - Acquire list of valid files.
2019-06-14 16:23:59,790 [pool-1-IoTDB-sync-client-timer-thread-1] INFO  org.apache.iotdb.db.sync.sender.SyncSenderImpl:183 - There has no file to sync !
```

Stop Sync Sender

Users can use the scripts under the `$IOTDB_HOME/bin` folder to stop the sync sender. For Linux and Mac OS X users:

```
Shell >$IOTDB_HOME/bin/stop-sync-client.sh
```

For Windows users:

```
Shell >$IOTDB_HOME\bin\stop-sync-client.bat
```

JMX Tool

Java VisualVM is a tool that provides a visual interface for viewing detailed information about Java applications while they are running on a Java Virtual Machine (JVM), and for troubleshooting and profiling these applications.

Usage

Step1: Fetch IoTDB-sever.

Step2: Edit configuration.

- IoTDB is LOCAL View `$IOTDB_HOME/conf/jmx.password`, and use default user or add new users here. If new users are added, remember to edit `$IOTDB_HOME/conf/jmx.access` and add new users' access
- IoTDB is not LOCAL Edit `$IOTDB_HOME/conf/iotdb-env.sh`, and modify config below:

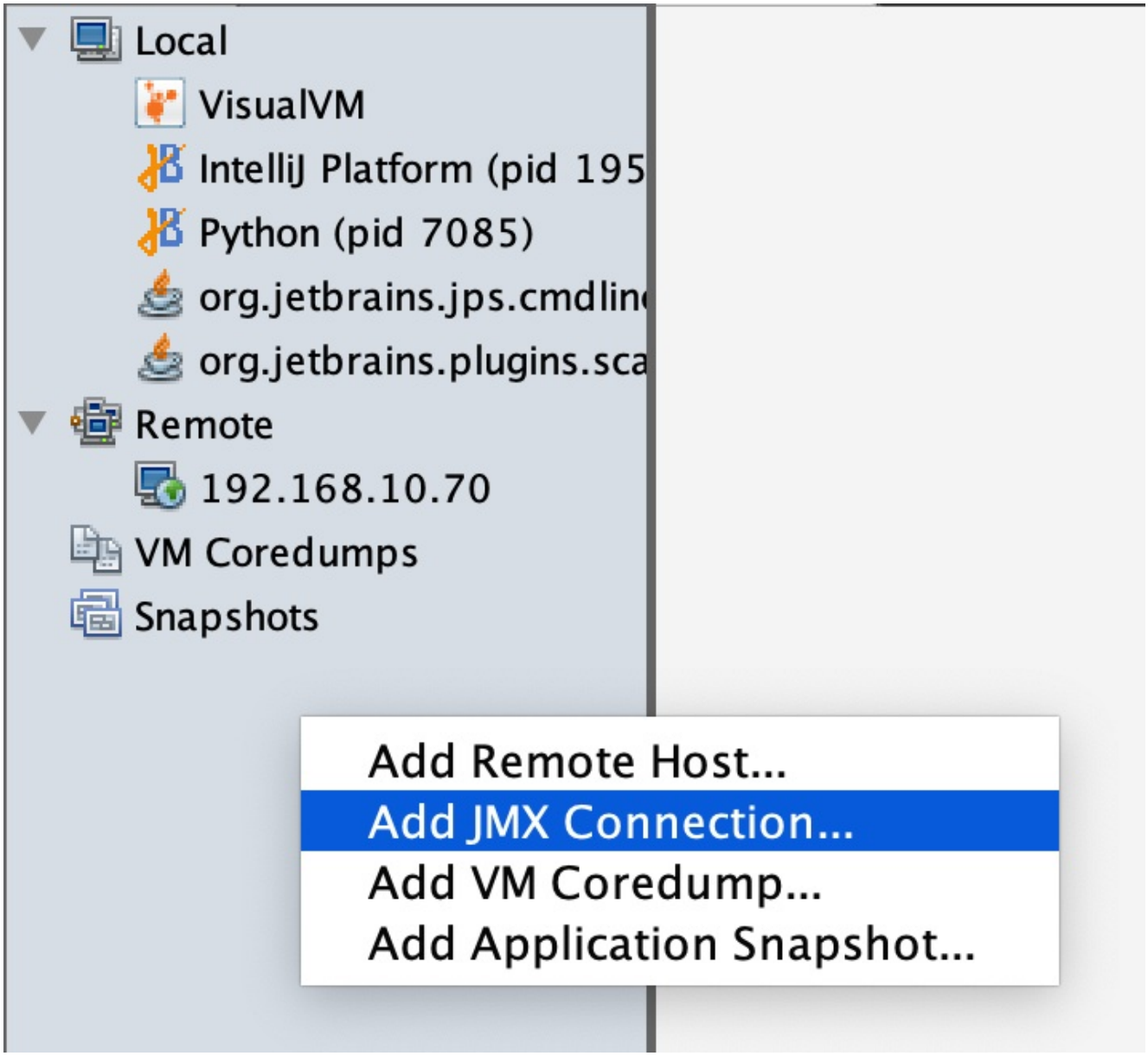
```
JMX_LOCAL="false"
JMX_IP="the_real_iotdb_server_ip"  # Write the actual IoTDB IP address
```

View `$IOTDB_HOME/conf/jmx.password`, and use default user or add new users here. If new users are added, remember to edit `$IOTDB_HOME/conf/jmx.access` and add new users' access

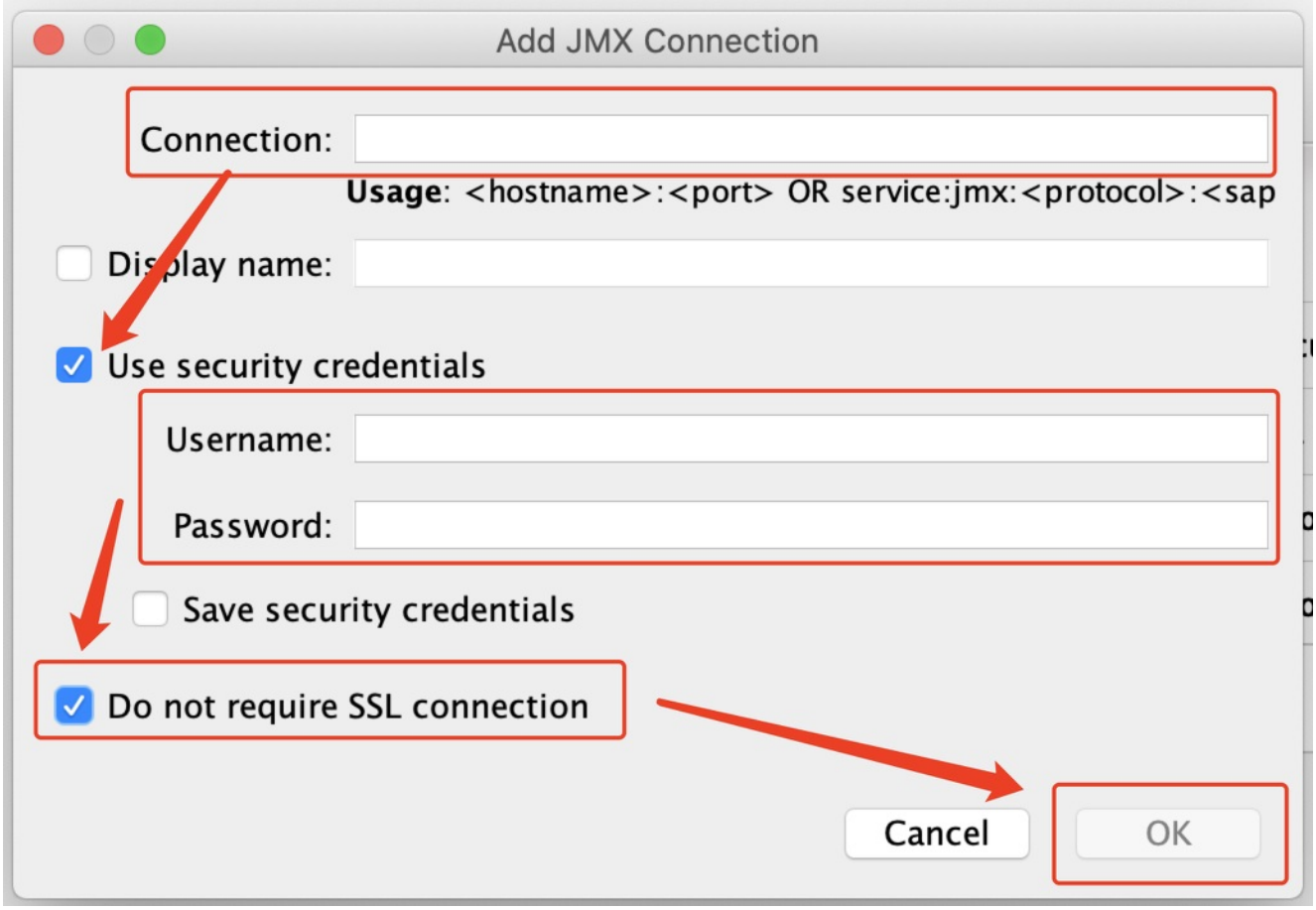
Step 3: Start IoTDB-server.

Step 4: Use jvisualvm

1. Make sure jdk 8 is installed. For versions later than jdk 8, you need to download visualvm (opens new window)
2. Open jvisualvm
3. Right-click at the left navigation area -> Add JMX connection



4. Fill in information and log in as below. Remember to check “Do not require SSL connection”. Here is an example: Connection: 192.168.130.15:31999 Username: iotdb Password: passw!d


Add JMX Connection
Connection:
Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap
Display name:
Use security credentials
Username:
Password:
Save security credentials
Do not require SSL connection
Cancel
OK

Watermark Tool

This tool has two functions: 1) watermark embedding of the IoTDB query result and 2) watermark detection of the suspected data.

1. Watermark Embedding

1.1 Configuration

Watermark is disabled by default in IoTDB. To enable watermark embedding, the first thing is to modify the following fields in the configuration file `iotdb-engine.properties` :

Name	Example	Ex
watermark_module_opened	false	t en wa en of Io se f di
watermark_secret_key	IoTDB*2019@Beijing	se de se
watermark_bit_string	100101110100	0- st be
watermark_method	GroupBasedLSBMethod(embed_row_cycle=2,embed_lsb_num=5)	sp th wa a_ ar pa

Notes:

- `watermark_module_opened` : Set it to be true if you want to enable watermark embedding
- `watermark_secret_key` : Character ‘&’ is not allowed. There is no constraint on the length of the secret key. Generally, the longer the key is, the higher the bar to intruders.
- `watermark_bit_string` : There is no constraint on the length of the bit string (except that it should not be empty). But note that it is difficult to reach the required significance level at the watermark detection phase if the bit string is way too short.
- `watermark_method` : Now only GroupBasedLSBMethod is supported, so actually you can only tune the two parameters of this method, which are `embed_row_cycle` and `embed_lsb_num` .

    - Both of them should be positive integers.
    - `embed_row_cycle` controls the ratio of rows watermarked. The smaller the `embed_row_cycle`, the larger the ratio of rows watermarked. When `embed_row_cycle` equals 1, every row is watermarked.
    - GroupBasedLSBMethod uses LSB embedding. `embed_lsb_num` controls the number of least significant bits available for watermark embedding. The biggger the `embed_lsb_num`, the larger the varying range of a data point.
- `watermark_secret_key`, `watermark_bit_string` and `watermark_method` should be kept secret from possible attackers. That is, it is your responsiblity to take care of `iotdb-engine.properties`.

1.2 Usage Example

step 1. Create a new user Alice, grant read privilege and query

A newly created user doesn’t use watermark by default. So the query result is the original data.

```
.\start-client.bat -u root -pw root
create user Alice 1234
grant user Alice privileges 'READ_TIMESERIES' on root.vehicle
exit

.\start-client.bat -u Alice -pw 1234
select * from root
+-----------------------------------+------------------+
|                               Time|root.vehicle.d0.s0|
+-----------------------------------+------------------+
|      1970-01-01T08:00:00.001+08:00|              21.5|
|      1970-01-01T08:00:00.002+08:00|              22.5|
|      1970-01-01T08:00:00.003+08:00|              23.5|
|      1970-01-01T08:00:00.004+08:00|              24.5|
|      1970-01-01T08:00:00.005+08:00|              25.5|
|      1970-01-01T08:00:00.006+08:00|              26.5|
|      1970-01-01T08:00:00.007+08:00|              27.5|
|      1970-01-01T08:00:00.008+08:00|              28.5|
|      1970-01-01T08:00:00.009+08:00|              29.5|
|      1970-01-01T08:00:00.010+08:00|              30.5|
|      1970-01-01T08:00:00.011+08:00|              31.5|
|      1970-01-01T08:00:00.012+08:00|              32.5|
|      1970-01-01T08:00:00.013+08:00|              33.5|
|      1970-01-01T08:00:00.014+08:00|              34.5|
|      1970-01-01T08:00:00.015+08:00|              35.5|
|      1970-01-01T08:00:00.016+08:00|              36.5|
|      1970-01-01T08:00:00.017+08:00|              37.5|
|      1970-01-01T08:00:00.018+08:00|              38.5|
|      1970-01-01T08:00:00.019+08:00|              39.5|
|      1970-01-01T08:00:00.020+08:00|              40.5|
|      1970-01-01T08:00:00.021+08:00|              41.5|
```

```
|      1970-01-01T08:00:00.022+08:00|              42.5|
|      1970-01-01T08:00:00.023+08:00|              43.5|
|      1970-01-01T08:00:00.024+08:00|              44.5|
|      1970-01-01T08:00:00.025+08:00|              45.5|
|      1970-01-01T08:00:00.026+08:00|              46.5|
|      1970-01-01T08:00:00.027+08:00|              47.5|
|      1970-01-01T08:00:00.028+08:00|              48.5|
|      1970-01-01T08:00:00.029+08:00|              49.5|
|      1970-01-01T08:00:00.030+08:00|              50.5|
|      1970-01-01T08:00:00.031+08:00|              51.5|
|      1970-01-01T08:00:00.032+08:00|              52.5|
|      1970-01-01T08:00:00.033+08:00|              53.5|
+-----------------------------------+-------------------+
```

step 2. grant watermark_embedding to Alice

Usage: `grant watermark_embedding to Alice`

Note that you can use `grant watermark_embedding to user1,user2,...` to grant watermark_embedding to multiple users.

Only root can run this command. After root grants watermark_embedding to Alice, all query results of Alice are watermarked.

```
.\start-client.bat -u root -pw root
grant watermark_embedding to Alice
exit

.\start-client.bat -u Alice -pw 1234
select * from root
+-----------------------------------+-------------------+
|                               Time|root.vehicle.d0.s0|
+-----------------------------------+-------------------+
|      1970-01-01T08:00:00.001+08:00|              21.5|
|      1970-01-01T08:00:00.002+08:00|              22.5|
|      1970-01-01T08:00:00.003+08:00|         23.500008|
|      1970-01-01T08:00:00.004+08:00|         24.500015|
|      1970-01-01T08:00:00.005+08:00|              25.5|
|      1970-01-01T08:00:00.006+08:00|         26.500015|
|      1970-01-01T08:00:00.007+08:00|              27.5|
|      1970-01-01T08:00:00.008+08:00|         28.500004|
|      1970-01-01T08:00:00.009+08:00|              29.5|
|      1970-01-01T08:00:00.010+08:00|              30.5|
|      1970-01-01T08:00:00.011+08:00|              31.5|
|      1970-01-01T08:00:00.012+08:00|              32.5|
|      1970-01-01T08:00:00.013+08:00|              33.5|
|      1970-01-01T08:00:00.014+08:00|              34.5|
|      1970-01-01T08:00:00.015+08:00|         35.500004|
|      1970-01-01T08:00:00.016+08:00|              36.5|
|      1970-01-01T08:00:00.017+08:00|              37.5|
|      1970-01-01T08:00:00.018+08:00|              38.5|
|      1970-01-01T08:00:00.019+08:00|              39.5|
```

```
|      1970-01-01T08:00:00.020+08:00|             40.5|
|      1970-01-01T08:00:00.021+08:00|             41.5|
|      1970-01-01T08:00:00.022+08:00|         42.500015|
|      1970-01-01T08:00:00.023+08:00|             43.5|
|      1970-01-01T08:00:00.024+08:00|         44.500008|
|      1970-01-01T08:00:00.025+08:00|          45.50003|
|      1970-01-01T08:00:00.026+08:00|         46.500008|
|      1970-01-01T08:00:00.027+08:00|         47.500008|
|      1970-01-01T08:00:00.028+08:00|             48.5|
|      1970-01-01T08:00:00.029+08:00|             49.5|
|      1970-01-01T08:00:00.030+08:00|             50.5|
|      1970-01-01T08:00:00.031+08:00|         51.500008|
|      1970-01-01T08:00:00.032+08:00|             52.5|
|      1970-01-01T08:00:00.033+08:00|             53.5|
+-----------------------------------+------------------+
```

step 3. revoke watermark_embedding from Alice

Usage: `revoke watermark_embedding from Alice`

Note that you can use `revoke watermark_embedding from user1,user2,...` to revoke watermark_embedding from multiple users.

Only root can run this command. After root revokes watermark_embedding from Alice, all query results of Alice are original again.

2. Watermark Detection

`detect-watermark.sh` and `detect-watermark.bat` are provided for different platforms.

Usage: ./detect-watermark.sh [filePath] [secretKey] [watermarkBitString] [embed_row_cycle] [embed_lsb_num] [alpha] [columnIndex] [dataType: int/float/double]

Example: ./detect-watermark.sh /home/data/dump1.csv IoTDB*2019@Beijing 100101110100 2 5 0.05 1 float

Args	Example	Explanation
filePath	/home/data/dump1.csv	suspected data file path
secretKey	IoTDB*2019@Beijing	see watermark embedding section
watermarkBitString	100101110100	see watermark embedding section
embed_row_cycle	2	see watermark embedding section
embed_lsb_num	5	see watermark embedding section
alpha	0.05	significance level
columnIndex	1	specifies one column of the data to detect
dataType	float	specifies the data type of the detected column; int/float/double

Notes:

- `filePath` : You can use export-csv tool to generate such data file. The first row is header and the first column is time. Data in the file looks like this:

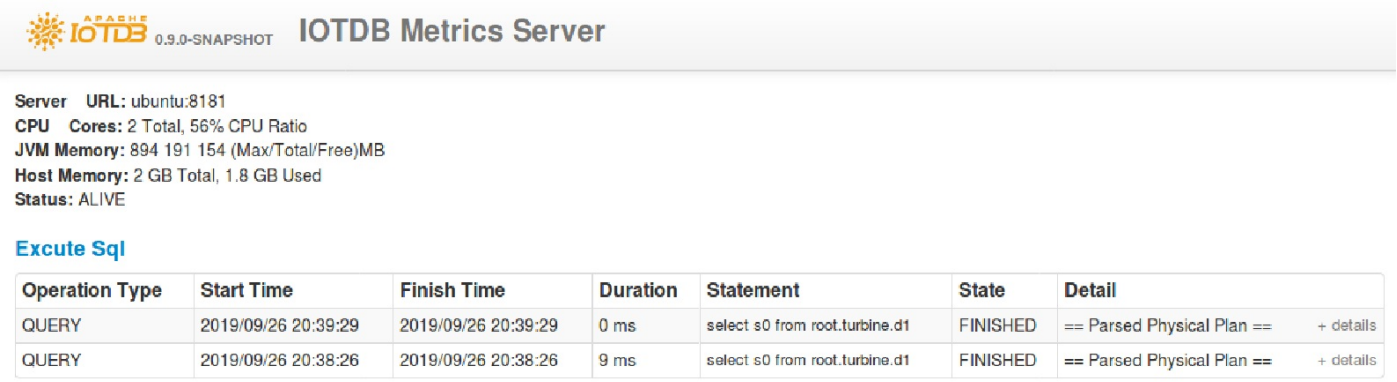
Time	root.vehicle.d0.s1	root.vehicle.d0.s1
1970-01-01T08:00:00.001+08:00	100	null
…	…	…

- `watermark_secret_key` , `watermark_bit_string` , `embed_row_cycle` and `embed_lsb_num` should be consistent with those used in the embedding phase.
- `alpha` : It should be in the range of [0,1]. The watermark detection is based on the significance test. The smaller the `alpha` is, the lower the probability that the data without the watermark is detected to be watermark embedded, and thus the higher the credibility of the result of detecting the existence of the watermark in data.
- `columnIndex` : It should be a postive integer.

Query History Visualization Tool

IoTDB Query History Visualization Tool uses a monitoring web page to provide metrics service for viewing the query history and SQL execution time. It can also provide the memory and CPU usage of the current host.

The port of IoTDB Query History Visualization Tool is `8181` . Just print your `ip:8181` in your browser, and you can view page like this:



> Note: Currently, we only support showing CPU ratio of Windows and Linux os. If you are using other OS, you may get a warning information: “Can’t get the cpu ratio, because this OS is not support”.

Monitor and Log Tools

System Monitor

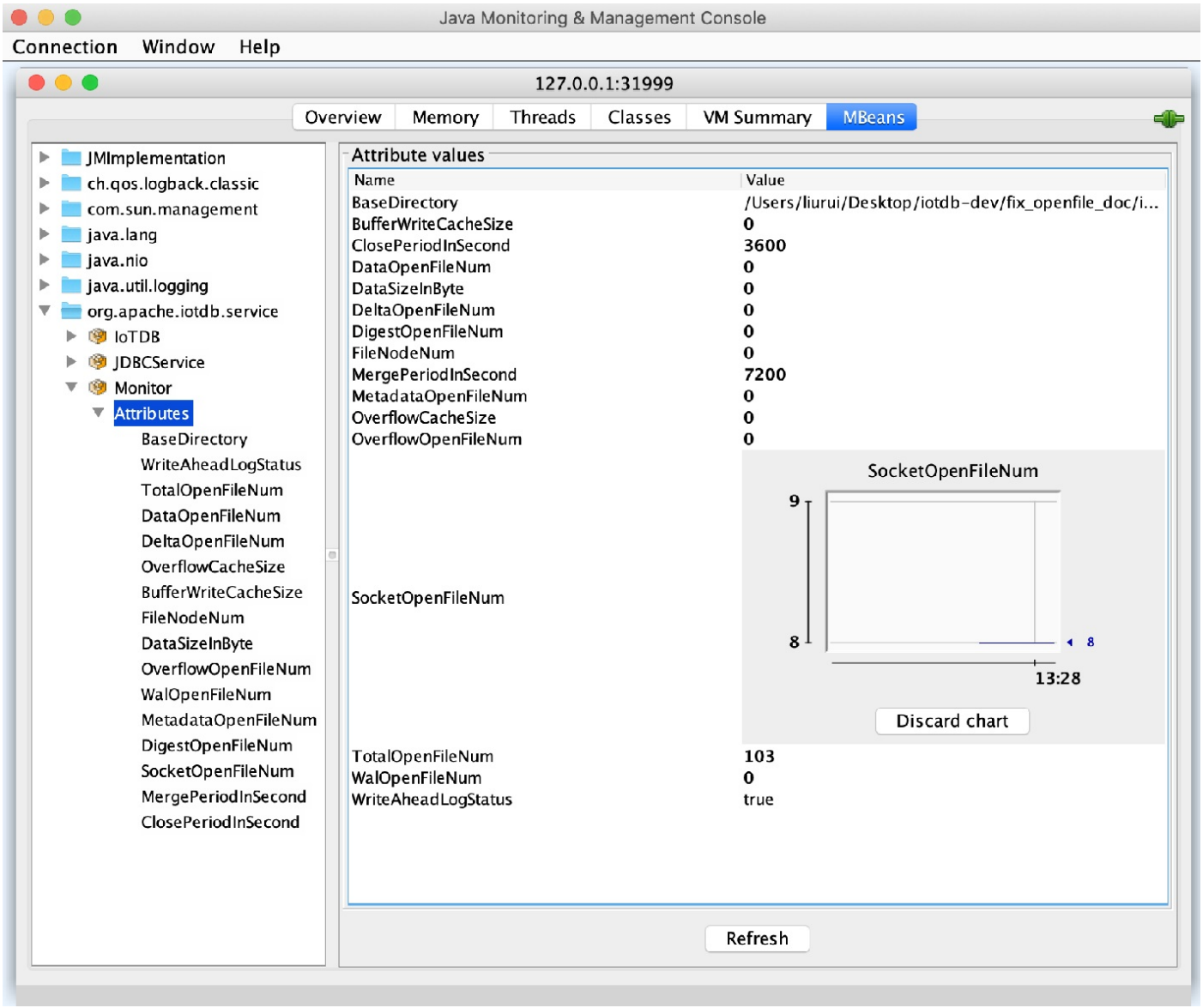
Currently, IoTDB provides users to use Java's JConsole tool to monitor system status or use IoTDB's open API to check data status.

System Status Monitoring

After starting JConsole tool and connecting to IoTDB server, you will have a basic look at IoTDB system status(CPU Occupation, in-memory information, etc.). See official documentation (opens new window) for more informations.

JMX MBean Monitoring

By using JConsole tool and connecting with JMX you can see some system statistics and parameters. This section describes how to use the JConsole `Mbean` tab to monitor the number of files opened by the IoTDB service process, the size of the data file, and so on. Once connected to JMX, you can find the `MBean` named `org.apache.iotdb.service` through the `MBeans` tab, as shown in the following Figure.



There are several attributes under Monitor, including the numbers of files opened in different folders, the data file size statistics and the values of some system parameters. By double-clicking the value corresponding to an attribute it can also display a line chart of that attribute. In particular, all the opened file count statistics are currently only supported on `MacOS` and most `Linux` distro except `CentOS`. For the OS not supported these statistics will return `-2`. See the following section for specific introduction of the Monitor attributes.

MBean Monitor Attributes List

- DataSizeInByte

Name	DataSizeInByte
Description	The total size of data file.
Unit	Byte
Type	Long

- FileNodeNum

Name	FileNodeNum
Description	The count number of FileNode. (Currently not supported)
Type	Long

- OverflowCacheSize

Name	OverflowCacheSize
Description	The size of out-of-order data cache. (Currently not supported)
Unit	Byte
Type	Long

- BufferWriteCacheSize

Name	BufferWriteCacheSize
Description	The size of BufferWriter cache. (Currently not supported)
Unit	Byte
Type	Long

- BaseDirectory

Name	BaseDirectory
Description	The absolute directory of data file.
Type	String

- WriteAheadLogStatus

Name	WriteAheadLogStatus
Description	The status of write-ahead-log (WAL). `True` means WAL is enabled.
Type	Boolean

- TotalOpenFileNum

Name	TotalOpenFileNum
Description	All the opened file number of IoTDB server process.
Type	Int

- DeltaOpenFileNum

Name	DeltaOpenFileNum
Description	The opened TsFile file number of IoTDB server process.
Default Directory	/data/data/settled
Type	Int

- WalOpenFileNum

Name	WalOpenFileNum
Description	The opened write-ahead-log file number of IoTDB server process.
Default Directory	/data/wal
Type	Int

- MetadataOpenFileNum

Name	MetadataOpenFileNum
Description	The opened meta-data file number of IoTDB server process.
Default Directory	/data/system/schema
Type	Int

- DigestOpenFileNum

Name	DigestOpenFileNum
Description	The opened info file number of IoTDB server process.
Default Directory	/data/system/info
Type	Int

- SocketOpenFileNum

Name	SocketOpenFileNum
Description	The Socket link (TCP or UDP) number of the operation system.
Type	Int

- MergePeriodInSecond

Name	MergePeriodInSecond
Description	The interval at which the IoTDB service process periodically triggers the merge process.
Unit	Second
Type	Long

- ClosePeriodInSecond

Name	ClosePeriodInSecond
Description	The interval at which the IoTDB service process periodically flushes memory data to disk.
Unit	Second
Type	Long

Data Status Monitoring

This module is the statistical monitoring method provided by IoTDB for users to store data information. We will record the statistical data in the system and store it in the database. The current 0.8.0 version of IoTDB provides statistics for writing data.

The user can choose to enable or disable the data statistics monitoring function (set the `enable_stat_monitor` item in the configuration file).

Writing Data Monitor

The current statistics of writing data by the system can be divided into two major modules: **Global Writing Data Statistics** and **Storage Group Writing Data Statistics**. **Global Writing Data Statistics** records the point number written by the user and the number of requests. **Storage Group Writing Data Statistics** records data of a certain storage group.

The system defaults to collect data every 5 seconds, and writes the statistics to the IoTDB and stores them in a system-specified locate. (If you need to change the statistic frequency, you can set The `back_loop_period_in_second entry` in the configuration file, see Section Engine Layer for details). After the system is refreshed or restarted, IoTDB does not recover the statistics, and the statistics data will restart from zero.

In order to avoid the excessive use of statistical information, we add a mechanism to periodically clear invalid data for statistical information. The system will delete invalid data at regular intervals. The user can set the trigger frequency (`stat_monitor_retain_interval_in_second` , default is 600s, see section Engine Layer for details) to set the frequency of deleting data. By setting the valid data duration (`stat_monitor_detect_freq_in_second entry` , the default is 600s, see section Engine Layer for details) to set the time period of valid data, that is, the data within the time of the clear operation trigger time is stat_monitor_detect_freq_in_second is valid data. In order to ensure the stability of the system, it is not allowed to delete the statistics frequently. Therefore, if the configuration parameter time is less than the default value (600s), the system will abort the configuration parameter and uses the default parameter.

It’s convenient for you to use `select` clause to get the writing data statistics the same as other timeseires.

Here are the writing data statistics:

- TOTAL_POINTS (GLOABAL)

Name	TOTAL_POINTS
Description	Calculate the global writing points number.
Type	Writing data statistics
Timeseries Name	root.stats.write.global.TOTAL_POINTS
Reset After Restarting System	yes

Example	select TOTAL_POINTS from root.stats.write.global

- TOTAL_REQ_SUCCESS (GLOABAL)

Name	TOTAL_REQ_SUCCESS
Description	Calculate the global successful requests number.
Type	Writing data statistics
Timeseries Name	root.stats.write.global.TOTAL_REQ_SUCCESS
Reset After Restarting System	yes
Example	select TOTAL_REQ_SUCCESS from root.stats.write.global

- TOTAL_REQ_FAIL (GLOABAL)

Name	TOTAL_REQ_FAIL
Description	Calculate the global failed requests number.
Type	Writing data statistics
Timeseries Name	root.stats.write.global.TOTAL_REQ_FAIL
Reset After Restarting System	yes
Example	select TOTAL_REQ_FAIL from root.stats.write.global

- TOTAL_POINTS_FAIL (GLOABAL)

Name	TOTAL_POINTS_FAIL
Description	Calculate the global failed writing points number.
Type	Writing data statistics
Timeseries Name	root.stats.write.global.TOTAL_POINTS_FAIL
Reset After Restarting System	yes
Example	select TOTAL_POINTS_FAIL from root.stats.write.global

- TOTAL_POINTS_SUCCESS (GLOABAL)

Name	TOTAL_POINTS_SUCCESS
Description	Calculate the c.
Type	Writing data statistics
Timeseries Name	root.stats.write.global.TOTAL_POINTS_SUCCESS
Reset After Restarting System	yes
Example	select TOTAL_POINTS_SUCCESS from root.stats.write.global

- TOTAL_REQ_SUCCESS (STORAGE GROUP)

Name	TOTAL_REQ_SUCCESS
Description	Calculate the successful requests number for specific storage group
Type	Writing data statistics
Timeseries Name	root.stats.write.<storage_group_name>.TOTAL_REQ_SUCCESS
Reset After Restarting System	yes
Example	select TOTAL_REQ_SUCCESS from root.stats.write.<storage_group_name>

- TOTAL_REQ_FAIL (STORAGE GROUP)

Name	TOTAL_REQ_FAIL
Description	Calculate the fail requests number for specific storage group
Type	Writing data statistics
Timeseries Name	root.stats.write.<storage_group_name>.TOTAL_REQ_FAIL
Reset After Restarting System	yes
Example	select TOTAL_REQ_FAIL from root.stats.write.<storage_group_name>

- TOTAL_POINTS_SUCCESS (STORAGE GROUP)

Name	TOTAL_POINTS_SUCCESS
Description	Calculate the successful writing points number for specific storage group.
Type	Writing data statistics
Timeseries Name	root.stats.write.<storage_group_name>.TOTAL_POINTS_SUCCESS
Reset After Restarting System	yes
Example	select TOTAL_POINTS_SUCCESS from root.stats.write.<storage_group_name>

- TOTAL_POINTS_FAIL (STORAGE GROUP)

Name	TOTAL_POINTS_FAIL
Description	Calculate the fail writing points number for specific storage group.
Type	Writing data statistics
Timeseries Name	root.stats.write.<storage_group_name>.TOTAL_POINTS_FAIL
Reset After Restarting System	yes

Example	select TOTAL_POINTS_FAIL from root.stats.write.<storage_group_name>

> Note:
>
> <storage_group_name> should be replaced by real storage group name, and the '.' in storage group need to be replaced by '_'. For example, the storage group name is 'root.a.b', when using in the statistics, it will change to 'root_a_b'

Example

Here we give some example of using writing data statistics.

If you want to know the global successful writing points number, you can use `select` clause to query it's value. The query statement is like this:

```
1. select TOTAL_POINTS_SUCCESS from root.stats.write.global
```

If you want to know the successfule writing points number of root.ln (storage group), here is the query statement:

```
1. select TOTAL_POINTS_SUCCESS from root.stats.write.root_ln
```

If you want to know the current timeseries point in the system, you can use `MAX_VALUE` function to query. Here is the query statement:

```
1. select MAX_VALUE(TOTAL_POINTS_SUCCESS) from root.stats.write.root_ln
```

File Size Monitor

Sometimes we are concerned about how the data file size of IoTDB is changing, maybe to help calculate how much disk space is left or the data ingestion speed. The File Size Monitor provides several statistics to show how different types of file-sizes change.

The file size monitor defaults to collect file size data every 5 seconds using the same shared parameter `back_loop_period_in_second` ,

Unlike Writing Data Monitor, currently File Size Monitor will not delete statistic data at regular intervals.

You can also use `select` clause to get the file size statistics like other time series.

Here are the file size statistics:

- DATA

Name	DATA
Description	Calculate the sum of all the files's sizes under the data

	directory (`data/data` by default) in byte.
Type	File size statistics
Timeseries Name	root.stats.file_size.DATA
Reset After Restarting System	No
Example	select DATA from root.stats.file_size.DATA

- SETTLED

Name	SETTLED
Description	Calculate the sum of all the `TsFile` size (under `data/data/settled` by default) in byte. If there are multiple `TsFile` directories like `{data/data/settled1, data/data/settled2}`, this statistic is the sum of their size.
Type	File size statistics
Timeseries Name	root.stats.file_size.SETTLED
Reset After Restarting System	No
Example	select SETTLED from root.stats.file_size.SETTLED

- OVERFLOW

Name	OVERFLOW
Description	Calculate the sum of all the `out-of-order data file` size (under `data/data/unsequence` by default) in byte.
Type	File size statistics
Timeseries Name	root.stats.file_size.OVERFLOW
Reset After Restarting System	No
Example	select OVERFLOW from root.stats.file_size.OVERFLOW

- WAL

Name	WAL
Description	Calculate the sum of all the `Write-Ahead-Log file` size (under `data/wal` by default) in byte.
Type	File size statistics
Timeseries Name	root.stats.file_size.WAL
Reset After Restarting System	No
Example	select WAL from root.stats.file_size.WAL

- INFO

Name	INFO
Description	Calculate the sum of all the `.restore`, etc. file size (under `data/system/info`) in byte.
Type	File size statistics
Timeseries Name	root.stats.file_size.INFO
Reset After Restarting System	No
Example	select INFO from root.stats.file_size.INFO

- SCHEMA

Name	SCHEMA
Description	Calculate the sum of all the `metadata file` size (under `data/system/metadata`) in byte.
Type	File size statistics
Timeseries Name	root.stats.file_size.SCHEMA
Reset After Restarting System	No
Example	select SCHEMA from root.stats.file_size.SCHEMA

Performance Monitor

Introduction

In order to grasp the performance of iotdb, we add this module to count the time-consumption of each operation. This module can compute the statistics of the avg time-consuming of each operation and the proportion of each operation whose time consumption falls into a time range. The output is in log_measure.log file. An output example is below.

```
OPERATION                                COUNT                TOTAL_TIME           AVG_TIME

EXECUTE_BATCH                            100004               2946053              29.4594

EXECUTE_ONE_SQL_IN_BATCH                 10010010             2943644              0.2941

EXECUTE_QUERY                            0                    0                    0.0000

========================================OPERATION HISTOGRAM=================================================

OPERATION                                1ms      4ms      16ms     64ms     256ms    1024ms   2147483647ms
EXECUTE_BATCH                            0.00%    0.00%    0.00%    98.79%   0.63%    0.41%    0.16%
EXECUTE_ONE_SQL_IN_BATCH                 99.94%   0.01%    0.02%    0.01%    0.00%    0.00%    0.00%
EXECUTE_QUERY                            0.00%    0.00%    0.00%    0.00%    0.00%    0.00%    0.00%
============================================================================================================
```

Configuration parameter

location: conf/iotdb-engine.properties

Table -parameter and description

Parameter	Default Value	Description
enable_performance_stat	false	Is stat performance of sub-module enable.
performance_stat_display_interval	60000	The interval of display statistic result in ms.
performance_stat_memory_in_kb	20	The memory used for performance_stat in kb.

JMX MBean

Connect to jconsole with port 31999, and choose ‘MBean’in menu bar. Expand the sidebar and choose ‘org.apache.iotdb.db.cost.statistic’. You can Find:

Attribute

1. EnableStat: Whether the statistics are enabled or not, if it is true, the module records the time-consuming of each operation and prints the results; It is non-editable but can be changed by the function below.

2. DisplayIntervalInMs: The interval between print results. The changes will not take effect instantly. To make the changes effective, you should call startContinuousStatistics() or startOneTimeStatistics().

3. OperationSwitch: It’s a map to indicate whether the statistics of one kind of operation should be computed, the key is operation name and the value is true means the statistics of the operation are enabled, otherwise disabled. This parameter cannot be changed directly, it’s changed by operation ‘changeOperationSwitch()’.

Operation

1. startContinuousStatistics: Start the statistics and output at interval of ‘DisplayIntervalInMs’.
2. startOneTimeStatistics: Start the statistics and output in delay of ‘DisplayIntervalInMs’.
3. stopStatistic: Stop the statistics.
4. clearStatisticalState(): clear current stat result, reset statistical result.
5. changeOperationSwitch(String operationName, Boolean operationState):set whether to monitor a kind of operation. The param ‘operationName’ is the name of operation, defined in attribute operationSwitch. The param operationState is whether to enable the statistics or not. If the state is switched successfully, the function will return true, else return false.

Adding Custom Monitoring Items for contributors of IOTDB

Add Operation

Add an enumeration in org.apache.iotdb.db.cost.statistic.Operation.

Add Timing Code in Monitoring Area

Add timing code in the monitoring start area:

```
1. long t0 = System. currentTimeMillis();
```

Add timing code in the monitoring stop area:

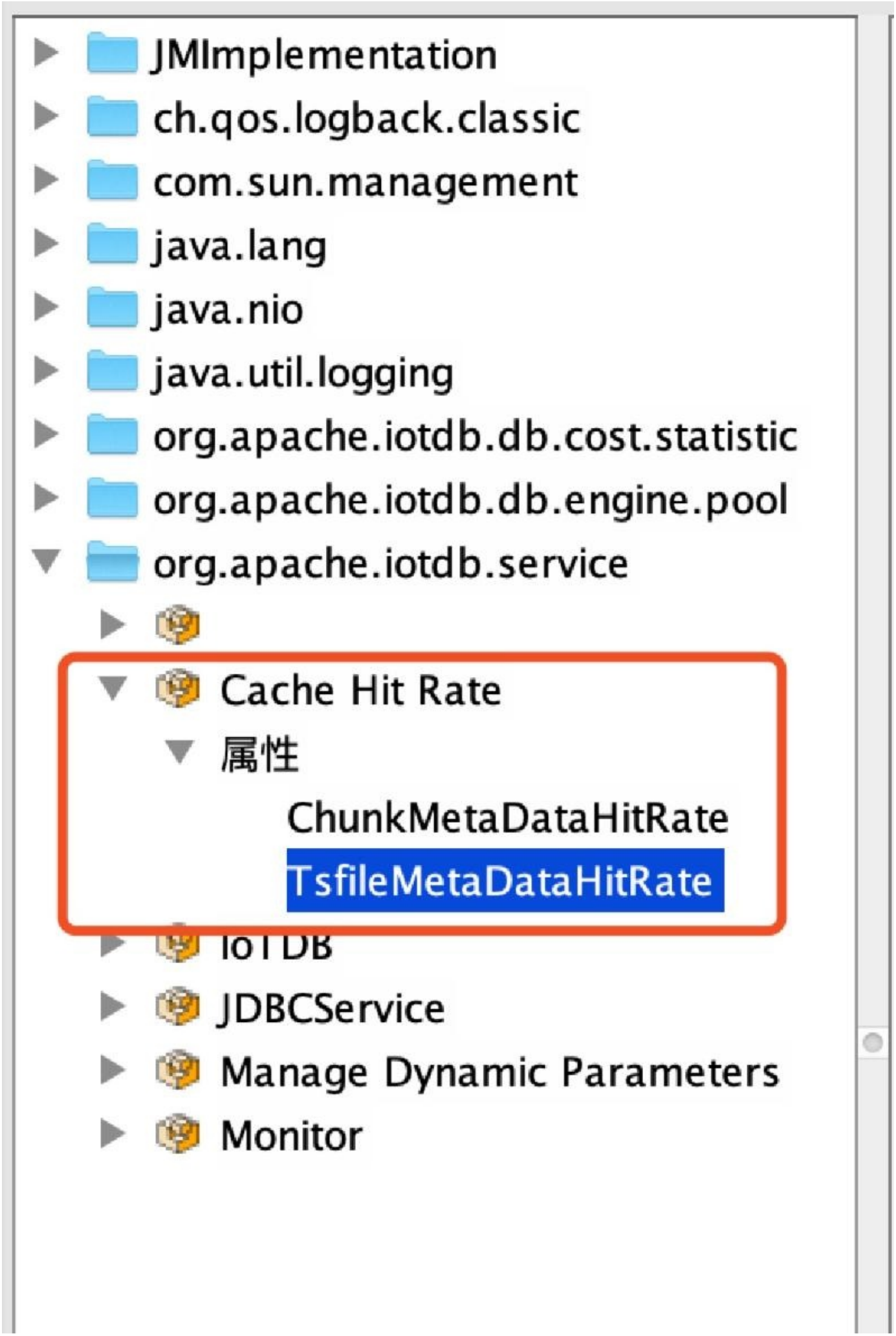
```
1. Measurement.INSTANCE.addOperationLatency(Operation, t0);
```

Cache Hit Ratio Statistics

Overview

To improve query performance, IOTDB caches ChunkMetaData and TsFileMetaData. Users can view the cache hit ratio through debug level log and MXBean, and adjust the memory occupied by the cache according to the cache hit ratio and system memory. The method of using MXBean to view cache hit ratio is as follows:

1. Connect to jconsole with port 31999 and select ‘MBean’ in the menu item above.
2. Expand the sidebar and select ‘org.apache.iotdb.db.service’. You will get the results shown in the following figure:


JMImplementation
ch.qos.logback.classic
com.sun.management
java.lang
java.nio
java.util.logging
org.apache.iotdb.db.cost.statistic
org.apache.iotdb.db.engine.pool
org.apache.iotdb.service
Cache Hit Rate
属性
ChunkMetaDataHitRate
TsfileMetaDataHitRate
IoTDB
JDBCService
Manage Dynamic Parameters
Monitor

System log

IoTDB allows users to configure IoTDB system logs (such as log output level) by modifying the log configuration file. The default location of the system log configuration file is in $IOTDB_HOME/conf folder.

The default log configuration file is named logback.xml. The user can modify the configuration of the system running log by adding or changing the xml tree node parameters. It should be noted that the configuration of the system log using the log configuration file does not take effect immediately after the modification, instead, it will take effect after restarting the system. The usage of logback.xml is just as usual.

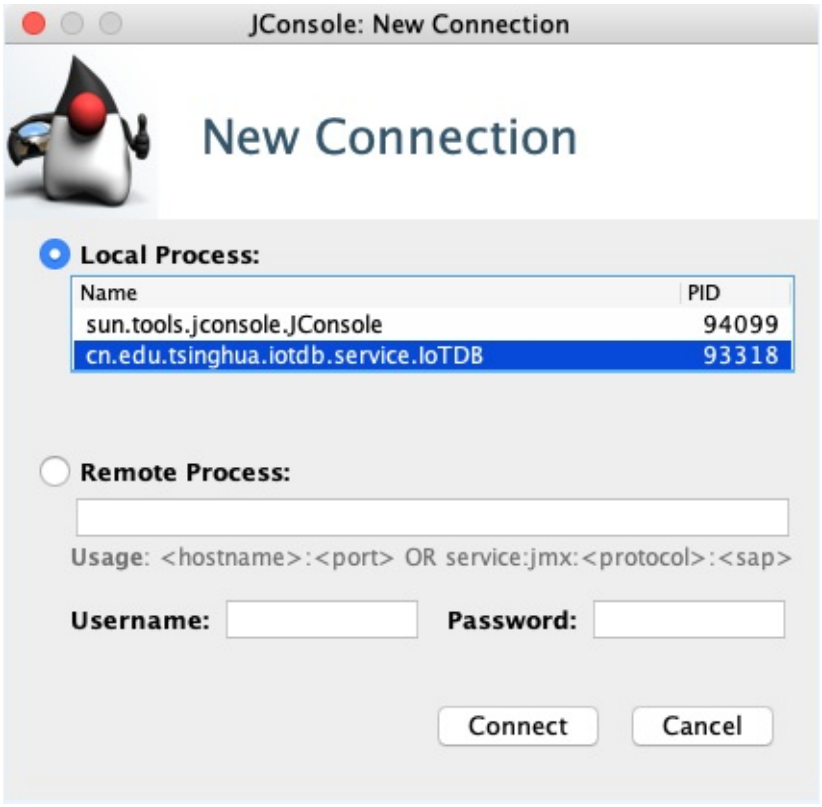
At the same time, in order to facilitate the debugging of the system by the developers and DBAs, we provide several JMX interfaces to dynamically modify the log configuration, and configure the Log module of the system in real time without restarting the system.

Dynamic System Log Configuration

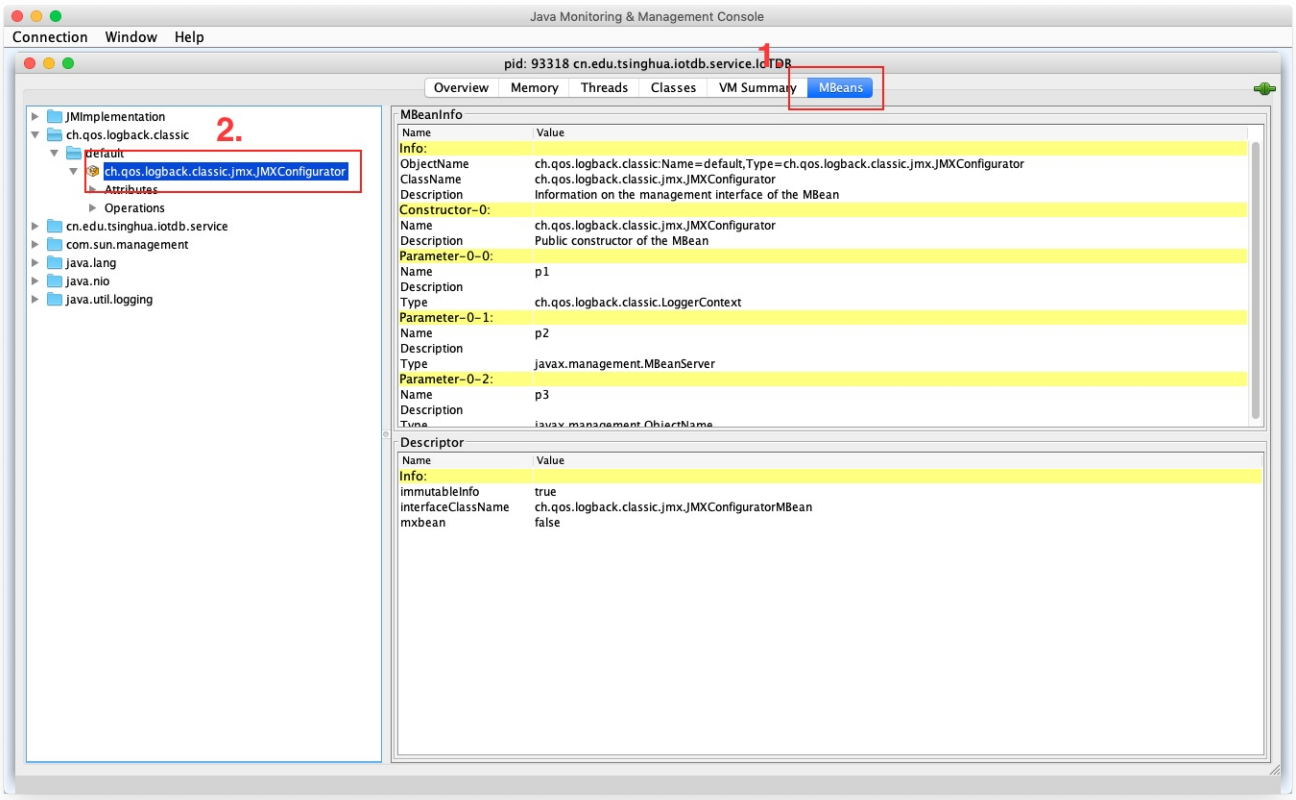
Connect JMX

Here we use JConsole to connect with JMX.

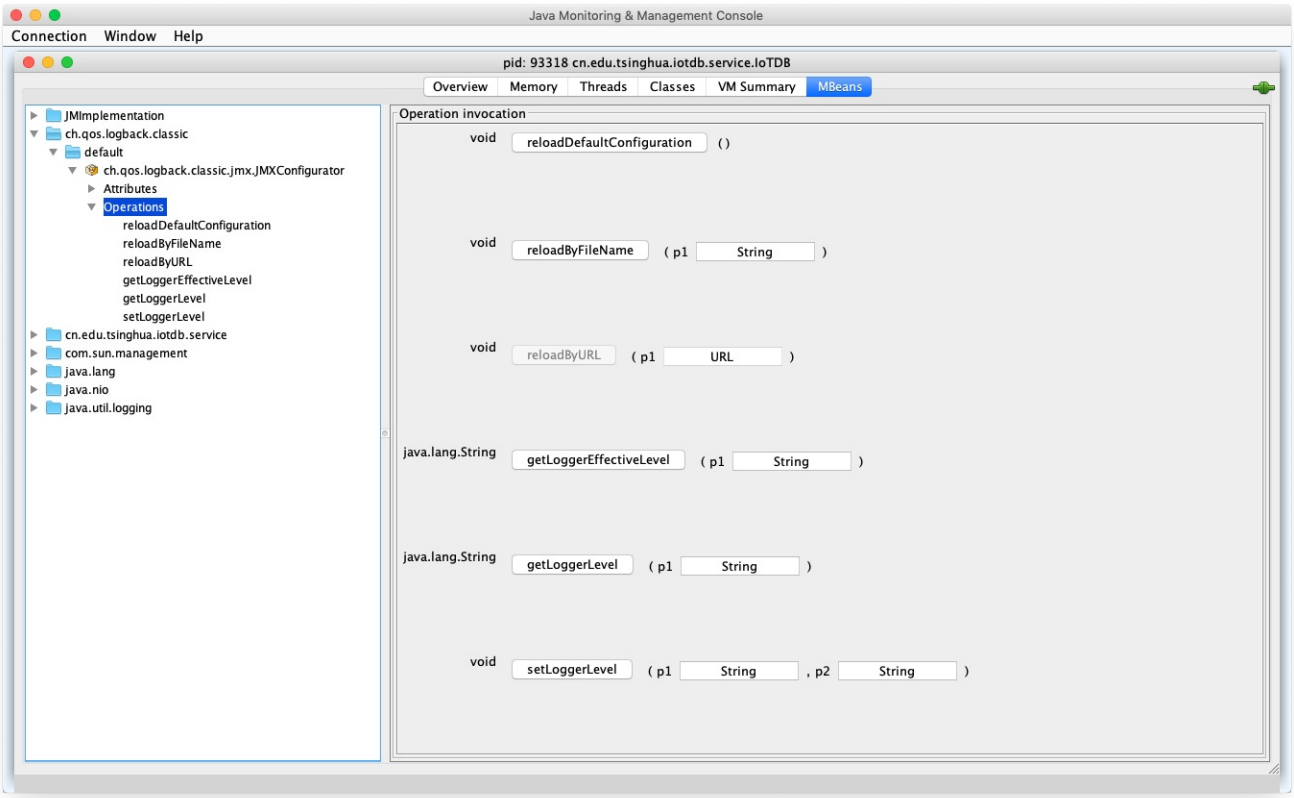
Start the JConsole, establish a new JMX connection with the IoTDB Server (you can select the local process or input the IP and PORT for remote connection, the default operation port of the IoTDB JMX service is 31999). Fig 4.1 shows the connection GUI of JConsole.



After connected, click `MBean` and find `ch.qos.logback.classic.default.ch.qos.logback.classic.jmx.JMXConfigurator` (As shown in fig 4.2).



In the JMXConfigurator Window, there are 6 operations provided for you, as shown in fig 4.3. You can use there interfaces to perform operation.



Interface Instruction

- reloadDefaultConfiguration

This method is to reload the default logback configuration file. The user can modify the default configuration file first, and then call this method to reload the modified configuration file into the system to take effect.

- reloadByFileName

This method loads a logback configuration file with the specified path and name, and then makes it take effect. This method accepts a parameter of type String named p1, which is the path to the configuration file that needs to be specified for loading.

- getLoggerEffectiveLevel

This method is to obtain the current log level of the specified Logger. This method accepts a String type parameter named p1, which is the name of the specified Logger. This method returns the log level currently in effect for the specified Logger.

- getLoggerLevel

This method is to obtain the log level of the specified Logger. This method accepts a String type parameter named p1, which is the name of the specified Logger. This method returns the log level of the specified Logger. It should be noted that the difference between this method and the `getLoggerEffectiveLevel` method is that the method returns the

log level that the specified Logger is set in the configuration file. If the user does not set the log level for the Logger. , then return empty. According to Logger’s log-level inheritance mechanism, f a Logger’s level is not explicitly set, it will inherit the log level settings from its nearest ancestor. At this point, calling the `getLoggerEffectiveLevel` method will return the log level in which the Logger is in effect; calling `getLoggerLevel` will return null.

Load External TsFile Tool

Introduction

The load external tsfile tool allows users to load tsfiles, delete a tsfile, or move a tsfile to target directory from the running Apache IoTDB instance.

Usage

The user sends specified commands to the Apache IoTDB system through the Cli tool or JDBC to use the tool.

load tsfiles

The command to load tsfiles is `load "<path/dir>" [true/false] [storage group level]` .

This command has two usages:

1. Load a single tsfile by specifying a file path (absolute path).

The second parameter indicates the path of the tsfile to be loaded and the name of the tsfile needs to conform to the tsfile naming convention, that is, `{systemTime}-{versionNum}-{mergeNum} .tsfile` . The third and fourth parameters are optional. When the metadata corresponding to the timeseries in the tsfile to be loaded does not exist, you can choose whether to create the schema automatically. If the third parameter is true, the schema will be created automatically. If the thrid parameter is false, the schema will not be created. By default, the schema will be created. When the storage group corresponding to the tsfile does not exist, the user can set the level of storage group through the fourth parameter. By default, it'll use the storage group level which is set in `iotdb-engine.properties` . If the `.resource` file corresponding to the file exists, it will be loaded into the data directory and engine of the Apache IoTDB. Otherwise, the corresponding `.resource` file will be regenerated from the tsfile file.

Examples:

- `load "/Users/Desktop/data/1575028885956-101-0.tsfile"`
- `load "/Users/Desktop/data/1575028885956-101-0.tsfile" false`
- `load "/Users/Desktop/data/1575028885956-101-0.tsfile" true`
- `load "/Users/Desktop/data/1575028885956-101-0.tsfile" true 1`

1. Load a batch of files by specifying a folder path (absolute path).

The second parameter indicates the path of the tsfile to be loaded and the name of the tsfiles need to conform to the tsfile naming convention, that is, `{systemTime}-{versionNum}-`

`{mergeNum} .tsfile` . The third and fourth parameters are optional. When the metadata corresponding to the time series in the tsfile to be loaded does not exist, you can choose whether to create the schema automatically. If the third parameter is true, the schema will be created automatically. If the third parameter is false, the schema will not be created. By default, the schema will be created. When the storage group corresponding to tsfile does not exist, the user can set the level of storage group through the fourth parameter. By default, it'll use the storage group level which is set in `iotdb-engine.properties` . If the `.resource` file corresponding to the file exists, they will be loaded into the data directory and engine of the Apache IoTDB. Otherwise, the corresponding `.resource` files will be regenerated from the tsfile sfile.

Examples:

- `load "/Users/Desktop/data"`
- `load "/Users/Desktop/data" false`
- `load "/Users/Desktop/data" true`
- `load "/Users/Desktop/data" true 1`

remove a tsfile

The command to delete a tsfile is: `remove "<path>"` .

This command deletes a tsfile by specifying the file path. The specific implementation is to delete the tsfile and its corresponding `.resource` and `.modification` files.

Examples:

- `remove "root.vehicle/1575028885956-101-0.tsfile"`
- `remove "1575028885956-101-0.tsfile"`

move a tsfile to a target directory

The command to move a tsfile to ta arget directory is: `move "<path>" "<dir>"` .

This command moves a tsfile to a target directory by specifying tsfile path and the target directory(absolute path). The specific implementation is to remove the tsfile from the engine and move the tsfile file and its corresponding `.resource` file to the target directory.

Examples:

- `move "root.vehicle/1575029224130-101-0.tsfile" "/data/data/tmp"`
- `move "1575029224130-101-0.tsfile" "/data/data/tmp"`

Performance Tracing Tool

IoTDB supports the use of `TRACING` statements to enable and disable performance tracing of query statements, which is disabled by default. Users can use performance tracking tool to analyze potential performance problems in some queries. By default, the log files for performance tracing are stored in the directory `./data/tracing` .

Turn on Tracing:

```
IoTDB> TRACING ON
```

Turn off Tracing:

```
IoTDB> TRACING OFF
```

Since the cost of an IoTDB query mainly depends on the number of time series queried, the number of tsfile files accessed, the total number of chunks to be scanned, and the average size of each chunk (the number of data points contained in the chunk). Therefore, the current performance analysis includes the following contents:

- Start time
- Query statement
- Number of series paths
- Number of sequence files
- Statistics of each sequence file
- Number of unSequence files
- Statistics of each unSequence file
- Number of chunks
- Average size of chunks
- Total cost time

Example

For example, execute `select * from root` , the contents of the tracing log file will include the following contents:

```
1. Query Id: 2 - Start time: 2020-06-28 10:53:54.727
2. Query Id: 2 - Query Statement: select * from root
3. Query Id: 2 - Number of series paths: 3
4. Query Id: 2 - Number of sequence files: 1
5. Query Id: 2 - SeqFile_1603336100446-1-0.tsfile root.sg.d1[1, 10000]
6. Query Id: 2 - Number of unsequence files: 1
7. Query Id: 2 - UnSeqFile_1603354798303-2-0.tsfile root.sg.d1[9, 1000]
8. Query Id: 2 - Number of chunks: 3
9. Query Id: 2 - Average size of chunks: 4113
10. Query Id: 2 - Total cost time: 11ms
```

In order to avoid disordered output information caused by multiple queries being executed at the same time, the Query ID is added before each output information. Users can use `grep "Query ID: 2" tracing.txt` to extract all tracing information of one query.

- Grafana
- MapReduce TsFile
- Spark TsFile
- Spark IoTDB
- Hive TsFile

IoTDB-Grafana

Outline

- IoTDB-Grafana
  - Grafana installation
    - Install Grafana
    - Install data source plugin
    - Start Grafana
  - IoTDB installation
  - IoTDB-Grafana installation
    - Start IoTDB-Grafana
  - Explore in Grafana
    - Add data source
    - Design in dashboard

IoTDB-Grafana

Grafana is an open source volume metrics monitoring and visualization tool, which can be used to display time series data and application runtime analysis. Grafana supports Graphite, InfluxDB and other major time series databases as data sources. IoTDB-Grafana is a connector which we developed to show time series data in IoTDB by reading data from IoTDB and sends to Grafana(https://grafana.com/). Before using this tool, make sure Grafana and IoTDB are correctly installed and started.

Grafana installation

Install Grafana

- Download url: https://grafana.com/grafana/download
- version >= 4.4.1

Install data source plugin

- plugin name: simple-json-datasource
- Download url: https://github.com/grafana/simple-json-datasource

After downloading this plugin, use the grafana-cli tool to install SimpleJson from the commandline:

```
grafana-cli plugins install grafana-simple-json-datasource
```

Alternatively, manually download the .zip file and unpack it into grafana plugins directory.

- `{grafana-install-directory}\data\plugins\` (Windows)
- `/var/lib/grafana/plugins` (Linux)
- `/usr/local/var/lib/grafana/plugins` (Mac)

Start Grafana

If Unix is used, Grafana will start automatically after installing, or you can run `sudo service grafana-server start` command. See more information here (opens new window).

If Mac and `homebrew` are used to install Grafana, you can use `homebrew` to start Grafana. First make sure homebrew/services is installed by running `brew tap homebrew/services`, then start Grafana using: `brew services start grafana`. See more information here (opens new window).

If Windows is used, start Grafana by executing grafana-server.exe, located in the bin directory, preferably from the command line. See more information here (opens new window).

IoTDB installation

See https://github.com/apache/iotdb

IoTDB-Grafana installation

```
1. git clone https://github.com/apache/iotdb.git
```

Start IoTDB-Grafana

Option one

Import the entire project, after the maven dependency is installed, directly run `iotdb/grafana/rc/main/java/org/apache/iotdb/web/grafana` directory `TsfileWebDemoApplication.java`, this grafana connector is developed by springboot

Option two

In `/grafana/target/` directory

```
1. cd iotdb
2. mvn clean package -pl grafana -am -Dmaven.test.skip=true
```

```
cd grafana/target
java -jar iotdb-grafana-{version}.war
```

If following output is displayed, then iotdb-grafana connector is successfully activated.

```
$ java -jar iotdb-grafana-{version}.war

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.4.RELEASE)
...
```

To configure properties, move the `grafana/src/main/resources/application.properties` to the same directory as the war package (`grafana/target`)

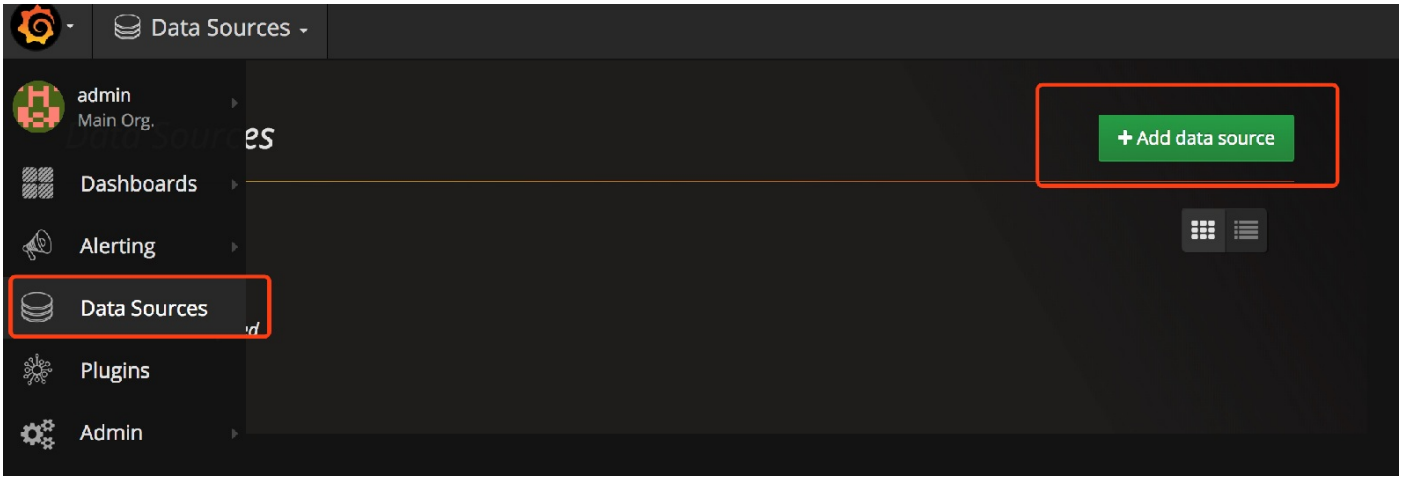
Explore in Grafana

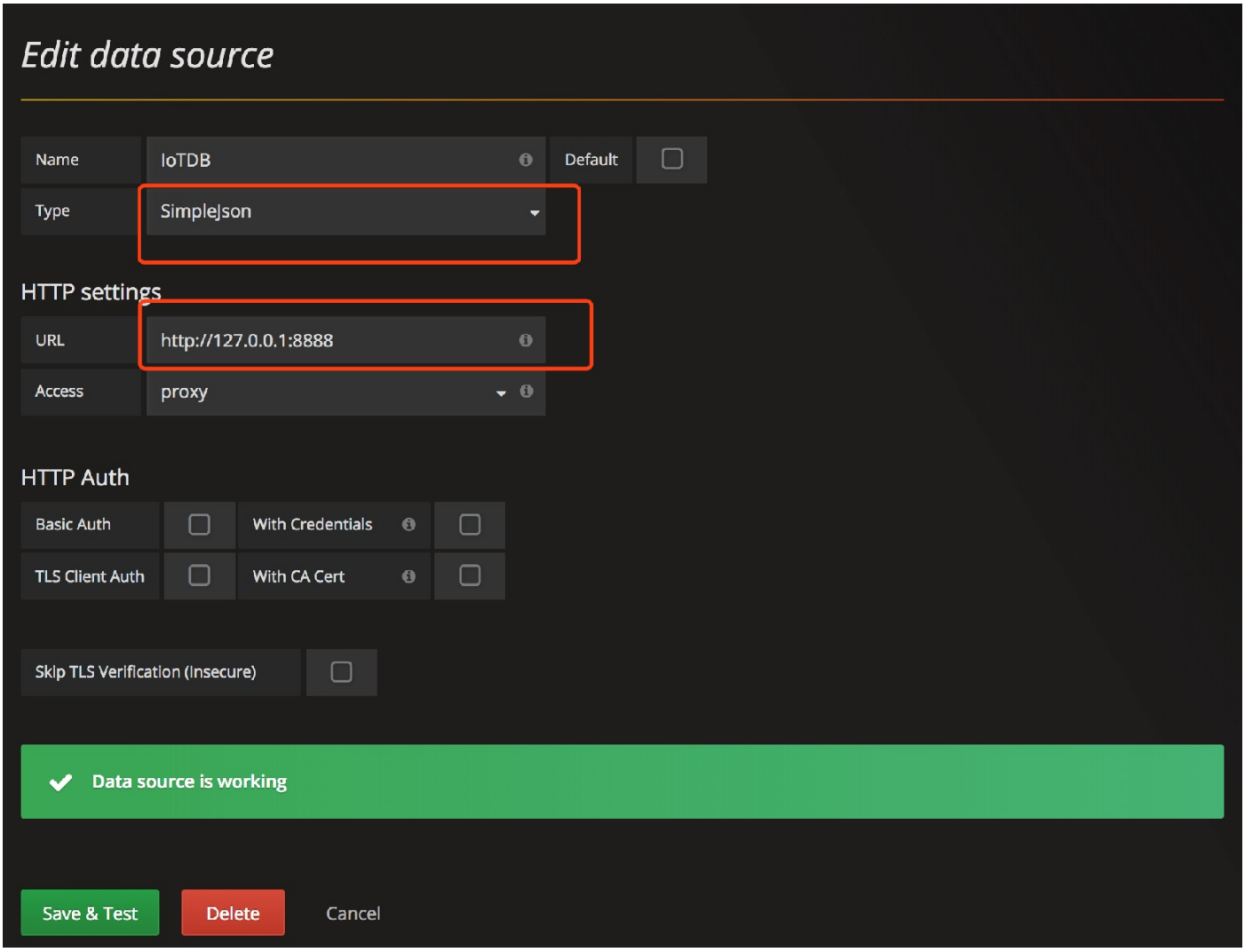
The default port of Grafana is 3000, see http://localhost:3000/

Username and password are both “admin” by default.

Add data source

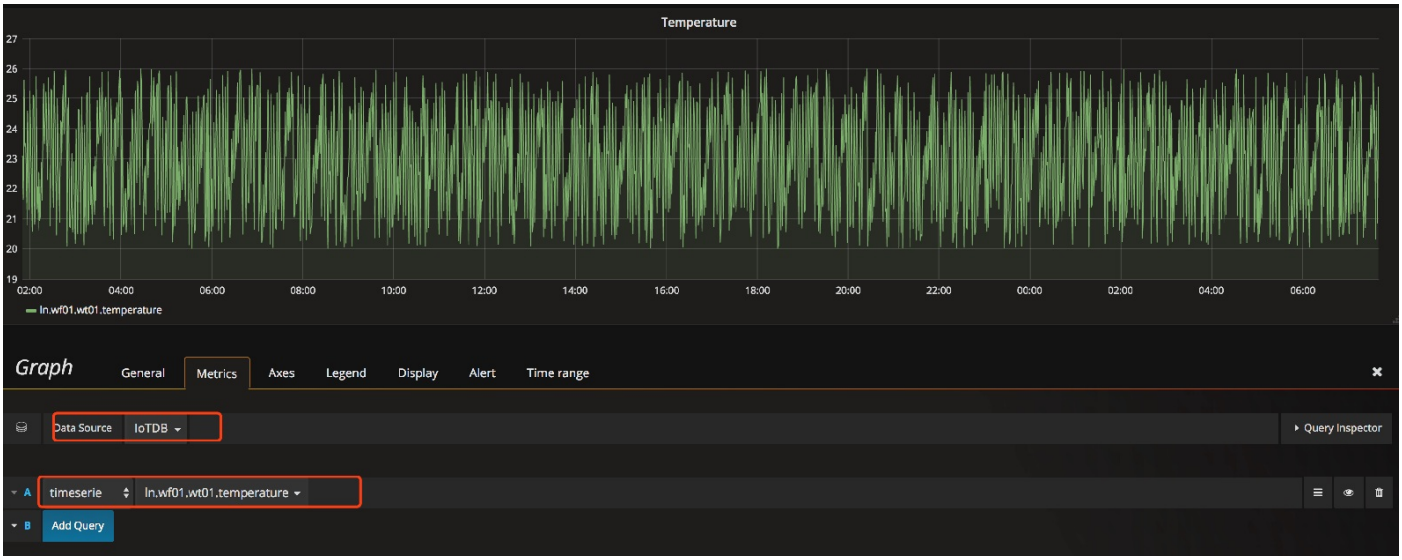
Select `Data Sources` and then `Add data source` , select `SimpleJson` in `Type` and `URL` is http://localhost:8888. After that, make sure IoTDB has been started, click “Save & Test”, and “Data Source is working” will be shown to indicate successful configuration.





Design in dashboard

Add diagrams in dashboard and customize your query. See http://docs.grafana.org/guides/getting_started/



config grafana

```
# ip and port of IoTDB
spring.datasource.url=jdbc:iotdb://127.0.0.1:6667/
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=org.apache.iotdb.jdbc.IoTDBDriver
server.port=8888
# Use this value to set timestamp precision as "ms", "us" or "ns", which must to be same with the timestamp
# precision of Apache IoTDB engine.
timestamp_precision=ms

# Use this value to set down sampling true/false
isDownSampling=true
# defaut sampling intervals
interval=1m
# aggregation function to use to downsampling the data (int, long, float, double)
# COUNT, FIRST_VALUE, LAST_VALUE, MAX_TIME, MAX_VALUE, AVG, MIN_TIME, MIN_VALUE, NOW, SUM
continuous_data_function=AVG
# aggregation function to use to downsampling the data (boolean, string)
# COUNT, FIRST_VALUE, LAST_VALUE, MAX_TIME, MIN_TIME, NOW
discrete_data_function=LAST_VALUE
```

The specific configuration information of interval is as follows

<1h: no sampling

1h~1d : intervals = 1m

1d~30d:intervals = 1h

>30d: intervals = 1d

After configuration, please re-run war package

```
java -jar iotdb-grafana-{version}.war
```

MapReduce TsFile

Outline

- TsFile-Hadoop-Connector User Guide
  - About TsFile-Hadoop-Connector
  - System Requirements
  - Data Type Correspondence
  - TSFInputFormat Explanation
  - Examples
    - Read Example: calculate the sum
    - Write Example: write the average into Tsfile

TsFile-Hadoop-Connector User Guide

About TsFile-Hadoop-Connector

TsFile-Hadoop-Connector implements the support of Hadoop for external data sources of Tsfile type. This enables users to read, write and query Tsfile by Hadoop.

With this connector, you can

- load a single TsFile, from either the local file system or hdfs, into Hadoop
- load all files in a specific directory, from either the local file system or hdfs, into hadoop
- write data from Hadoop into TsFile

System Requirements

Hadoop Version	Java Version	TsFile Version
`2.7.3`	`1.8`	`0.11.1`

> Note: For more information about how to download and use TsFile, please see the following link: https://github.com/apache/iotdb/tree/master/tsfile.

Data Type Correspondence

TsFile data type	Hadoop writable
BOOLEAN	BooleanWritable
INT32	IntWritable

INT64	LongWritable
FLOAT	FloatWritable
DOUBLE	DoubleWritable
TEXT	Text

TSFInputFormat Explanation

TSFInputFormat extract data from tsfile and format them into records of `MapWritable` .

Suppose that we want to extract data of the device named `d1` which has three sensors named `s1` , `s2` , `s3` .

`s1` 's type is `BOOLEAN` , `s2` 's type is `DOUBLE` , `s3` 's type is `TEXT` .

The `MapWritable` struct will be like:

```
{
    "time_stamp": 10000000,
    "device_id":  d1,
    "s1":         true,
    "s2":         3.14,
    "s3":         "middle"
}
```

In the Map job of Hadoop, you can get any value you want by key as following:

`mapwritable.get(new Text("s1"))`

> Note: All keys in `MapWritable` are `Text` type.

Examples

Read Example: calculate the sum

First of all, we should tell InputFormat what kind of data we want from tsfile.

```
    // configure reading time enable
    TSFInputFormat.setReadTime(job, true);
    // configure reading deviceId enable
    TSFInputFormat.setReadDeviceId(job, true);
    // configure reading which deltaObjectIds
    String[] deviceIds = {"device_1"};
    TSFInputFormat.setReadDeviceIds(job, deltaObjectIds);
    // configure reading which measurementIds
    String[] measurementIds = {"sensor_1", "sensor_2", "sensor_3"};
    TSFInputFormat.setReadMeasurementIds(job, measurementIds);
```

And then,the output key and value of mapper and reducer should be specified

```
    // set inputformat and outputformat
    job.setInputFormatClass(TSFInputFormat.class);
    // set mapper output key and value
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(DoubleWritable.class);
    // set reducer output key and value
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(DoubleWritable.class);
```

Then, the `mapper` and `reducer` class is how you deal with the `MapWritable` produced by `TSFInputFormat` class.

```
  public static class TSMapper extends Mapper<NullWritable, MapWritable, Text, DoubleWritable> {

    @Override
    protected void map(NullWritable key, MapWritable value,
        Mapper<NullWritable, MapWritable, Text, DoubleWritable>.Context context)
        throws IOException, InterruptedException {

      Text deltaObjectId = (Text) value.get(new Text("device_id"));
      context.write(deltaObjectId, (DoubleWritable) value.get(new Text("sensor_3")));
    }
  }

  public static class TSReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    @Override
    protected void reduce(Text key, Iterable<DoubleWritable> values,
        Reducer<Text, DoubleWritable, Text, DoubleWritable>.Context context)
        throws IOException, InterruptedException {

      double sum = 0;
      for (DoubleWritable value : values) {
        sum = sum + value.get();
      }
      context.write(key, new DoubleWritable(sum));
    }
  }
```

> Note: For the complete code, please see the following link: https://github.com/apache/iotdb/blob/master/example/hadoop/src/main/java/org/apache/iotdb//hadoop/tsfile/TSFMRReadExample.java

Write Example: write the average into Tsfile

Except for the `OutputFormatClass`, the rest of configuration code for hadoop map-reduce job is almost same as above.

```
    job.setOutputFormatClass(TSFOutputFormat.class);
    // set reducer output key and value
    job.setOutputKeyClass(NullWritable.class);
    job.setOutputValueClass(HDFSTSRecord.class);
```

Then, the `mapper` and `reducer` class is how you deal with the `MapWritable` produced by `TSFInputFormat` class.

```
    public static class TSMapper extends Mapper<NullWritable, MapWritable, Text, MapWritable> {
        @Override
        protected void map(NullWritable key, MapWritable value,
                           Mapper<NullWritable, MapWritable, Text, MapWritable>.Context context)
                throws IOException, InterruptedException {

            Text deltaObjectId = (Text) value.get(new Text("device_id"));
            long timestamp = ((LongWritable)value.get(new Text("timestamp"))).get();
            if (timestamp % 100000 == 0) {
                context.write(deltaObjectId, new MapWritable(value));
            }
        }
    }

    /**
     * This reducer calculate the average value.
     */
    public static class TSReducer extends Reducer<Text, MapWritable, NullWritable, HDFSTSRecord> {

        @Override
        protected void reduce(Text key, Iterable<MapWritable> values,
                              Reducer<Text, MapWritable, NullWritable, HDFSTSRecord>.Context context) throws
IOException, InterruptedException {
            long sensor1_value_sum = 0;
            long sensor2_value_sum = 0;
            double sensor3_value_sum = 0;
            long num = 0;
            for (MapWritable value : values) {
                num++;
                sensor1_value_sum += ((LongWritable)value.get(new Text("sensor_1"))).get();
                sensor2_value_sum += ((LongWritable)value.get(new Text("sensor_2"))).get();
                sensor3_value_sum += ((DoubleWritable)value.get(new Text("sensor_3"))).get();
            }
            HDFSTSRecord tsRecord = new HDFSTSRecord(1L, key.toString());
            DataPoint dPoint1 = new LongDataPoint("sensor_1", sensor1_value_sum / num);
            DataPoint dPoint2 = new LongDataPoint("sensor_2", sensor2_value_sum / num);
            DataPoint dPoint3 = new DoubleDataPoint("sensor_3", sensor3_value_sum / num);
            tsRecord.addTuple(dPoint1);
            tsRecord.addTuple(dPoint2);
            tsRecord.addTuple(dPoint3);
            context.write(NullWritable.get(), tsRecord);
        }
    }
```

> Note: For the complete code, please see the following link:
> https://github.com/apache/iotdb/blob/master/example/hadoop/src/main/java/org/apache/iotdb//hadoop/tsfile/TSMRWriteExample.java

TsFile-Spark-Connector User Guide

1. About TsFile-Spark-Connector

TsFile-Spark-Connector implements the support of Spark for external data sources of Tsfile type. This enables users to read, write and query Tsfile by Spark.

With this connector, you can

- load a single TsFile, from either the local file system or hdfs, into Spark
- load all files in a specific directory, from either the local file system or hdfs, into Spark
- write data from Spark into TsFile

2. System Requirements

Spark Version	Scala Version	Java Version	TsFile
`2.4.3`	`2.11.8`	`1.8`	`0.11.1`

> Note: For more information about how to download and use TsFile, please see the following link: https://github.com/apache/iotdb/tree/master/tsfile.

3. Quick Start

Local Mode

Start Spark with TsFile-Spark-Connector in local mode:

```
1. ./<spark-shell-path>  --jars  tsfile-spark-connector.jar,tsfile-0.11.1-jar-with-dependencies.jar
```

Note:

- <spark-shell-path> is the real path of your spark-shell.
- Multiple jar packages are separated by commas without any spaces.
- See https://github.com/apache/iotdb/tree/master/tsfile for how to get TsFile.

Distributed Mode

Start Spark with TsFile-Spark-Connector in distributed mode (That is, the spark cluster is connected by spark-shell):

```
    . /<spark-shell-path>   --jars  tsfile-spark-connector.jar,tsfile-{version}-jar-with-dependencies.jar  --
1. master spark://ip:7077
```

Note:

- <spark-shell-path> is the real path of your spark-shell.
- Multiple jar packages are separated by commas without any spaces.
- See https://github.com/apache/iotdb/tree/master/tsfile for how to get TsFile.

4. Data Type Correspondence

TsFile data type	SparkSQL data type
BOOLEAN	BooleanType
INT32	IntegerType
INT64	LongType
FLOAT	FloatType
DOUBLE	DoubleType
TEXT	StringType

5. Schema Inference

The way to display TsFile is dependent on the schema. Take the following TsFile structure as an example: There are three measurements in the TsFile schema: status, temperature, and hardware. The basic information of these three measurements is listed:

Name	Type	Encode
status	Boolean	PLAIN
temperature	Float	RLE
hardware	Text	PLAIN

The existing data in the TsFile are:

device:root.ln.wf01.wt01				device:root.ln.wf02.wt02			
status		temperature		hardware		status	
time	value	time	value	time	value	time	value
1	True	1	2.2	2	“aaa”	1	True
3	True	2	2.2	4	“bbb”	2	False
5	False	3	2.1	6	“ccc”	4	True

The corresponding SparkSQL table is:

time	root.ln.wf02.wt02.temperature	root.ln.wf02.wt02.status	root.ln.wf02.wt02.
1	null	true	null

2	null	false	aaa
3	null	null	null
4	null	true	bbb
5	null	null	null
6	null	null	ccc

You can also use narrow table form which as follows: (You can see part 6 about how to use narrow form)

time	device_name	status	hardware	temperature
1	root.ln.wf02.wt01	true	null	2.2
1	root.ln.wf02.wt02	true	null	null
2	root.ln.wf02.wt01	null	null	2.2
2	root.ln.wf02.wt02	false	aaa	null
3	root.ln.wf02.wt01	true	null	2.1
4	root.ln.wf02.wt02	true	bbb	null
5	root.ln.wf02.wt01	false	null	null
6	root.ln.wf02.wt02	null	ccc	null

6. Scala API

NOTE: Remember to assign necessary read and write permissions in advance.

Example 1: read from the local file system

```
import org.apache.iotdb.spark.tsfile._
val wide_df = spark.read.tsfile("test.tsfile")
wide_df.show

val narrow_df = spark.read.tsfile("test.tsfile", true)
narrow_df.show
```

Example 2: read from the hadoop file system

```
import org.apache.iotdb.spark.tsfile._
val wide_df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile")
wide_df.show

val narrow_df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile", true)
narrow_df.show
```

Example 3: read from a specific directory

```
import org.apache.iotdb.spark.tsfile._
val df = spark.read.tsfile("hdfs://localhost:9000/usr/hadoop")
df.show
```

Note 1: Global time ordering of all TsFiles in a directory is not supported now.

Note 2: Measurements of the same name should have the same schema.

Example 4: query in wide form

```
import org.apache.iotdb.spark.tsfile._
val df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile")
df.createOrReplaceTempView("tsfile_table")
val newDf = spark.sql("select * from tsfile_table where `device_1.sensor_1`>0 and `device_1.sensor_2` < 22")
newDf.show
```

```
import org.apache.iotdb.spark.tsfile._
val df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile")
df.createOrReplaceTempView("tsfile_table")
val newDf = spark.sql("select count(*) from tsfile_table")
newDf.show
```

Example 5: query in narrow form

```
import org.apache.iotdb.spark.tsfile._
val df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile", true)
df.createOrReplaceTempView("tsfile_table")
val newDf = spark.sql("select * from tsfile_table where device_name = 'root.ln.wf02.wt02' and temperature > 5")
newDf.show
```

```
import org.apache.iotdb.spark.tsfile._
val df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile", true)
df.createOrReplaceTempView("tsfile_table")
val newDf = spark.sql("select count(*) from tsfile_table")
newDf.show
```

Example 6: write in wide form

```
// we only support wide_form table to write
import org.apache.iotdb.spark.tsfile._

val df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile")
df.show
df.write.tsfile("hdfs://localhost:9000/output")

```

```
val newDf = spark.read.tsfile("hdfs://localhost:9000/output")
newDf.show
```

Example 6: write in narrow form

```
// we only support wide_form table to write
import org.apache.iotdb.spark.tsfile._

val df = spark.read.tsfile("hdfs://localhost:9000/test.tsfile", true)
df.show
df.write.tsfile("hdfs://localhost:9000/output", true)

val newDf = spark.read.tsfile("hdfs://localhost:9000/output", true)
newDf.show
```

Appendix A: Old Design of Schema Inference

The way to display TsFile is related to TsFile Schema. Take the following TsFile structure as an example: There are three measurements in the Schema of TsFile: status, temperature, and hardware. The basic info of these three Measurements is:

Name	Type	Encode
status	Boolean	PLAIN
temperature	Float	RLE
hardware	Text	PLAIN

Basic info of Measurements

The existing data in the file are:

delta_object:root.ln.wf01.wt01				delta_object:root.ln.wf02.wt02				delta_object:root.s		
status		temperature		hardware		status		status		
time	value	time	value	time	value	time	value	time	value	t
1	True	1	2.2	2	“aaa”	1	True	2	True	3
3	True	2	2.2	4	“bbb”	2	False	3	True	6
5	False	3	2.1	6	“ccc”	4	True	4	True	8
7	True	4	2.0	8	“ddd”	5	False	6	True	9

A set of time-series data

There are two ways to show a set of time-series data:

the default way

Two columns are created to store the full path of the device: time(LongType) and delta_object(StringType).

- `time` : Timestamp, LongType
- `delta_object` : Delta_object ID, StringType

Next, a column is created for each Measurement to store the specific data. The SparkSQL table structure is:

time(LongType)	delta_object(StringType)	status(BooleanType)	temperature(FloatT
1	root.ln.wf01.wt01	True	2.2
1	root.ln.wf02.wt02	True	null
2	root.ln.wf01.wt01	null	2.2
2	root.ln.wf02.wt02	False	null
2	root.sgcc.wf03.wt01	True	null
3	root.ln.wf01.wt01	True	2.1
3	root.sgcc.wf03.wt01	True	3.3
4	root.ln.wf01.wt01	null	2.0
4	root.ln.wf02.wt02	True	null
4	root.sgcc.wf03.wt01	True	null
5	root.ln.wf01.wt01	False	null
5	root.ln.wf02.wt02	False	null
5	root.sgcc.wf03.wt01	True	null
6	root.ln.wf02.wt02	null	null
6	root.sgcc.wf03.wt01	null	6.6
7	root.ln.wf01.wt01	True	null
8	root.ln.wf02.wt02	null	null
8	root.sgcc.wf03.wt01	null	8.8
9	root.sgcc.wf03.wt01	null	9.9

unfold delta_object column

Expand the device column by “.” into multiple columns, ignoring the root directory “root”. Convenient for richer aggregation operations. To use this display way, the parameter “delta_object_name” is set in the table creation statement (refer to Example 5 in Section 5.1 of this manual), as in this example, parameter “delta_object_name” is set to “root.device.turbine”. The number of path layers needs to be one-to-one. At this point, one column is created for each layer of the device path except the “root” layer. The column name is the name in the parameter and the value is the name of the corresponding layer of the device. Next, one column is created for each Measurement to store the specific data.

Then SparkSQL Table Structure is as follows:

time(LongType)	group(StringType)	field(StringType)	device(StringType)	status
1	ln	wf01	wt01	True
1	ln	wf02	wt02	True
2	ln	wf01	wt01	null
2	ln	wf02	wt02	False
2	sgcc	wf03	wt01	True
3	ln	wf01	wt01	True
3	sgcc	wf03	wt01	True
4	ln	wf01	wt01	null
4	ln	wf02	wt02	True
4	sgcc	wf03	wt01	True
5	ln	wf01	wt01	False
5	ln	wf02	wt02	False
5	sgcc	wf03	wt01	True
6	ln	wf02	wt02	null
6	sgcc	wf03	wt01	null
7	ln	wf01	wt01	True
8	ln	wf02	wt02	null
8	sgcc	wf03	wt01	null
9	sgcc	wf03	wt01	null

TsFile-Spark-Connector displays one or more TsFiles as a table in SparkSQL By SparkSQL. It also allows users to specify a single directory or use wildcards to match multiple directories. If there are multiple TsFiles, the union of the measurements in all TsFiles will be retained in the table, and the measurement with the same name have the same data type by default. Note that if a situation with the same name but different data types exists, TsFile-Spark-Connector does not guarantee the correctness of the results.

The writing process is to write a DataFrame as one or more TsFiles. By default, two columns need to be included: time and delta_object. The rest of the columns are used as Measurement. If user wants to write the second table structure back to TsFile, user can set the “delta_object_name” parameter(refer to Section 5.1 of Section 5.1 of this manual).

Appendix B: Old Note

NOTE: Check the jar packages in the root directory of your Spark and replace libthrift-0.9.2.jar and libfb303-0.9.2.jar with libthrift-0.9.1.jar and libfb303-

```
0.9.1.jar respectively.
```

Spark IoTDB Connecter

version

The versions required for Spark and Java are as follow:

Spark Version	Scala Version	Java Version	TsFile
`2.4.3`	`2.11`	`1.8`	`0.11.1`

install

mvn clean scala:compile compile install

1. maven dependency

```
    <dependency>
      <groupId>org.apache.iotdb</groupId>
      <artifactId>spark-iotdb-connector</artifactId>
      <version>0.11.1</version>
    </dependency>
```

2. spark-shell user guide

```
spark-shell --jars spark-iotdb-connector-0.11.1.jar,iotdb-jdbc-0.11.1-jar-with-dependencies.jar

import org.apache.iotdb.spark.db._

val df = spark.read.format("org.apache.iotdb.spark.db").option("url","jdbc:iotdb://127.0.0.1:6667/").option("sql","select * from root").load

df.printSchema()

df.show()
```

To partition rdd:

```
spark-shell --jars spark-iotdb-connector-0.11.1.jar,iotdb-jdbc-0.11.1-jar-with-dependencies.jar

import org.apache.iotdb.spark.db._

```

```
val df =
spark.read.format("org.apache.iotdb.spark.db").option("url","jdbc:iotdb://127.0.0.1:6667/").option("sql","select * from root").
                        option("lowerBound", [lower bound of time that you want query(include)]).option("upperBound", [upper bound of time that you want query(include)]).
                        option("numPartition", [the partition number you want]).load

df.printSchema()

df.show()
```

3. Schema Inference

Take the following TsFile structure as an example: There are three Measurements in the TsFile schema: status, temperature, and hardware. The basic information of these three measurements is as follows:

Name	Type	Encode
status	Boolean	PLAIN
temperature	Float	RLE
hardware	Text	PLAIN

The existing data in the TsFile is as follows:

device:root.ln.wf01.wt01				device:root.ln.wf02.wt02			
status		temperature		hardware		status	
time	value	time	value	time	value	time	value
1	True	1	2.2	2	“aaa”	1	True
3	True	2	2.2	4	“bbb”	2	False
5	False	3	2.1	6	“ccc”	4	True

The wide(default) table form is as follows:

time	root.ln.wf02.wt02.temperature	root.ln.wf02.wt02.status	root.ln.wf02.wt02.
1	null	true	null
2	null	false	aaa
3	null	null	null
4	null	true	bbb
5	null	null	null
6	null	null	ccc

You can also use narrow table form which as follows: (You can see part 4 about how to use narrow form)

time	device_name	status	hardware	temperature
1	root.ln.wf02.wt01	true	null	2.2
1	root.ln.wf02.wt02	true	null	null
2	root.ln.wf02.wt01	null	null	2.2
2	root.ln.wf02.wt02	false	aaa	null
3	root.ln.wf02.wt01	true	null	2.1
4	root.ln.wf02.wt02	true	bbb	null
5	root.ln.wf02.wt01	false	null	null
6	root.ln.wf02.wt02	null	ccc	null

4. Transform between wide and narrow table

from wide to narrow

```
import org.apache.iotdb.spark.db._

val wide_df = spark.read.format("org.apache.iotdb.spark.db").option("url",
"jdbc:iotdb://127.0.0.1:6667/").option("sql", "select * from root where time < 1100 and time > 1000").load
val narrow_df = Transformer.toNarrowForm(spark, wide_df)
```

from narrow to wide

```
import org.apache.iotdb.spark.db._

val wide_df = Transformer.toWideForm(spark, narrow_df)
```

5. Java user guide

```
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.apache.iotdb.spark.db.*

public class Example {

  public static void main(String[] args) {
    SparkSession spark = SparkSession
        .builder()
        .appName("Build a DataFrame from Scratch")
        .master("local[*]")
        .getOrCreate();

```

```
15.     Dataset<Row> df = spark.read().format("org.apache.iotdb.spark.db")
16.         .option("url","jdbc:iotdb://127.0.0.1:6667/")
17.         .option("sql","select * from root").load();
18.
19.     df.printSchema();
20.
21.     df.show();
22.
23.     Dataset<Row> narrowTable = Transformer.toNarrowForm(spark, df)
24.     narrowTable.show()
25.   }
26. }
```

TsFile-Hive-Connector User Guide

Outline

- TsFile-Hive-Connector User Guide
  - About TsFile-Hive-Connector
  - System Requirements
  - Data Type Correspondence
  - Add Dependency For Hive
  - Create Tsfile-backed Hive tables
  - Query from Tsfile-backed Hive tables
    - Select Clause Example
    - Aggregate Clause Example
  - What’s Next

About TsFile-Hive-Connector

TsFile-Hive-Connector implements the support of Hive for external data sources of Tsfile type. This enables users to operate Tsfile by Hive.

With this connector, you can

- Load a single TsFile, from either the local file system or hdfs, into hive
- Load all files in a specific directory, from either the local file system or hdfs, into hive
- Query the tsfile through HQL.
- As of now, the write operation is not supported in hive-connector. So, insert operation in HQL is not allowed while operating tsfile through hive.

System Requirements

Hadoop Version	Hive Version	Java Version	TsFile
`2.7.3` or `3.2.1`	`2.3.6` or `3.1.2`	`1.8`	`0.11.1`

> Note: For more information about how to download and use TsFile, please see the following link: https://github.com/apache/iotdb/tree/master/tsfile.

Data Type Correspondence

TsFile data type	Hive field type
BOOLEAN	Boolean
INT32	INT

INT64	BIGINT
FLOAT	Float
DOUBLE	Double
TEXT	STRING

Add Dependency For Hive

To use hive-connector in hive, we should add the hive-connector jar into hive.

After downloading the code of iotdb from https://github.com/apache/iotdb (opens new window), you can use the command of `mvn clean package -pl hive-connector -am -Dmaven.test.skip=true` to get a `hive-connector-X.X.X-jar-with-dependencies.jar` .

Then in hive, use the command of `add jar XXX` to add the dependency. For example:

```
hive> add jar /Users/hive/iotdb/hive-connector/target/hive-connector-0.11.1-jar-with-dependencies.jar;

Added [/Users/hive/iotdb/hive-connector/target/hive-connector-0.11.1-jar-with-dependencies.jar] to class path
Added resources: [/Users/hive/iotdb/hive-connector/target/hive-connector-0.11.1-jar-with-dependencies.jar]
```

Create Tsfile-backed Hive tables

To create a Tsfile-backed table, specify the `serde` as `org.apache.iotdb.hive.TsFileSerDe` , specify the `inputformat` as `org.apache.iotdb.hive.TSFHiveInputFormat` , and the `outputformat` as `org.apache.iotdb.hive.TSFHiveOutputFormat` .

Also provide a schema which only contains two fields: `time_stamp` and `sensor_id` for the table. `time_stamp` is the time value of the time series and `sensor_id` is the sensor name to extract from the tsfile to hive such as `sensor_1` . The name of the table can be any valid table names in hive.

Also a location provided for hive-connector to pull the most current data for the table.

The location should be a specific directory on your local file system or HDFS to set up Hadoop. If it is in your local file system, the location should look like `file:///data/data/sequence/root.baic2.WWS.leftfrontdoor/`

Last, set the `device_id` in `TBLPROPERTIES` to the device name you want to analyze.

For example:

```
CREATE EXTERNAL TABLE IF NOT EXISTS only_sensor_1(
  time_stamp TIMESTAMP,
  sensor_1 BIGINT)
ROW FORMAT SERDE 'org.apache.iotdb.hive.TsFileSerDe'
```

```
STORED AS
  INPUTFORMAT 'org.apache.iotdb.hive.TSFHiveInputFormat'
  OUTPUTFORMAT 'org.apache.iotdb.hive.TSFHiveOutputFormat'
LOCATION '/data/data/sequence/root.baic2.WWS.leftfrontdoor/'
TBLPROPERTIES ('device_id'='root.baic2.WWS.leftfrontdoor.plc1');
```

In this example, the data of `root.baic2.WWS.leftfrontdoor.plc1.sensor_1` is pulled from the directory of `/data/data/sequence/root.baic2.WWS.leftfrontdoor/`. This table results in a description as below:

```
hive> describe only_sensor_1;
OK
time_stamp              timestamp                   from deserializer
sensor_1                bigint                  from deserializer
Time taken: 0.053 seconds, Fetched: 2 row(s)
```

At this point, the Tsfile-backed table can be worked with in Hive like any other table.

Query from Tsfile-backed Hive tables

Before we do any queries, we should set the `hive.input.format` in hive by executing the following command.

```
hive> set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
```

Now, we already have an external table named `only_sensor_1` in hive. We can use any query operations through HQL to analyse it.

For example:

Select Clause Example

```
hive> select * from only_sensor_1 limit 10;
OK
1    1000000
2    1000001
3    1000002
4    1000003
5    1000004
6    1000005
7    1000006
8    1000007
9    1000008
10    1000009
Time taken: 1.464 seconds, Fetched: 10 row(s)
```

Aggregate Clause Example

```
hive> select count(*) from only_sensor_1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a
different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = jackietien_20191016202416_d1e3e233-d367-4453-b39a-2aac9327a3b6
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2019-10-16 20:24:18,305 Stage-1 map = 0%,  reduce = 0%
2019-10-16 20:24:27,443 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local867757288_0002
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
1000000
Time taken: 11.334 seconds, Fetched: 1 row(s)
```

What’s Next

Only read operation is currently supported. Write operation is under development.

- Files
- Writing Data on HDFS
- Shared Nothing Cluster

Files

In IoTDB, there are many kinds of data needed to be stored. This section introduces IoTDB's data storage strategy to provide you an explicit understanding of IoTDB's data management.

The data in IoTDB is divided into three categories, namely data files, system files, and pre-write log files.

Data Files

Data files store all the data that the user wrote to IoTDB, which contains TsFile and other files. TsFile storage directory can be configured with the `data_dirs` configuration item (see file layer for details). Other files can be configured through data_dirs configuration item (see Engine Layer for details).

In order to support users' storage requirements such as disk space expansion better, IoTDB supports multiple file directories storage methods for TsFile storage configuration. Users can set multiple storage paths as data storage locations(see data_dirs configuration item), and you can specify or customize the directory selection strategy (see multi_dir_strategy configuration item for details).

System files

System files include schema files, which store metadata information of data in IoTDB. It can be configured through the `base_dir` configuration item (see System Layer for details).

Pre-write Log Files

Pre-write log files store WAL files. It can be configured through the `wal_dir` configuration item (see System Layer for details).

Example of Setting Data storage Directory

For a clearer understanding of configuring the data storage directory, we will give an example in this section.

The data directory path included in storage directory setting are: base_dir, data_dirs, multi_dir_strategy, and wal_dir, which refer to system files, data folders, storage strategy, and pre-write log files.

An example of the configuration items are as follows:

```
base_dir = $IOTDB_HOME/data
data_dirs = /data1/data, /data2/data, /data3/data
multi_dir_strategy=MaxDiskUsableSpaceFirstStrategy
wal_dir= $IOTDB_HOME/data/wal
```

After setting the configuration, the system will:

- Save all system files in $IOTDB_HOME/data
- Save TsFile in /data1/data, /data2/data, /data3/data. And the choosing strategy is `MaxDiskUsableSpaceFirstStrategy`, when data writes to the disk, the system will automatically select a directory with the largest remaining disk space to write data.
- Save WAL data in $IOTDB_HOME/data/wal

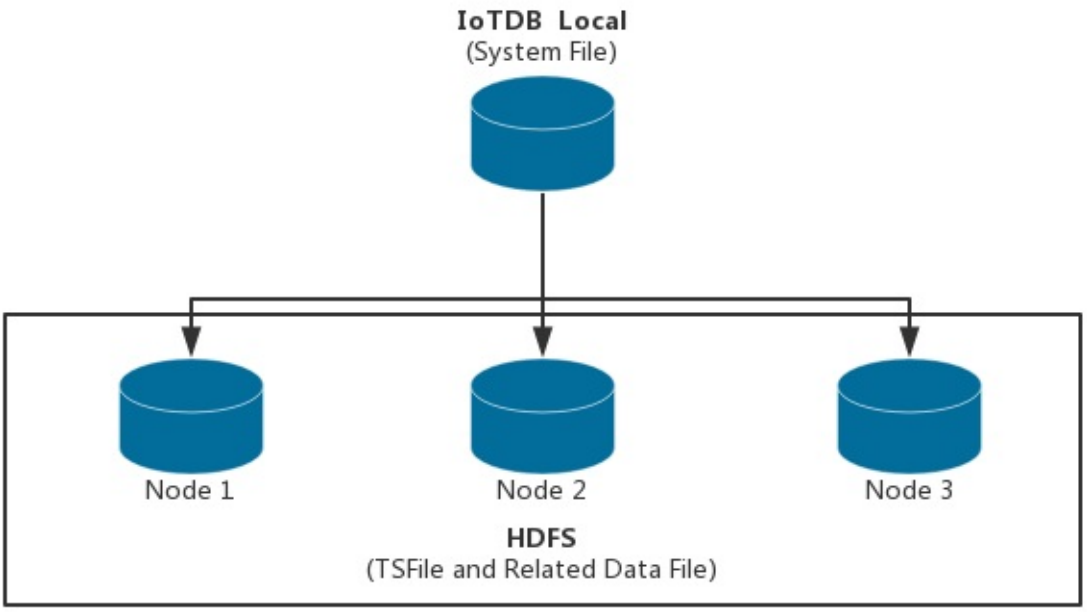
Writing Data on HDFS

Shared Storage Architecture

Currently, TSFiles(including both TSFile and related data files) are supported to be stored in local file system and hadoop distributed file system(HDFS). It is very easy to config the storage file system of TSFile.

System architecture

When you config to store TSFile on HDFS, your data files will be in distributed storage. The system architecture is as below:



Config and usage

To store TSFile and related data files in HDFS, here are the steps:

First, download the source release from website or git clone the repository, the tag of a released version is release/x.x.x

Build server and Hadoop module by: `mvn clean package -pl server,hadoop -am -Dmaven.test.skip=true`

Then, copy the target jar of Hadoop module `hadoop-tsfile-0.11.1-jar-with-dependencies.jar` into server target lib folder `.../server/target/iotdb-server-0.11.1/lib` .

Edit user config in `iotdb-engine.properties` . Related configurations are:

- tsfile_storage_fs

Name	tsfile_storage_fs
Description	The storage file system of Tsfile and related data files. Currently LOCAL file system and HDFS are supported.
Type	String
Default	LOCAL
Effective	Only allowed to be modified in first start up

- core_site_path

Name	core_site_path
Description	Absolute file path of core-site.xml if Tsfile and related data files are stored in HDFS.
Type	String
Default	/etc/hadoop/conf/core-site.xml
Effective	After restart system

- hdfs_site_path

Name	hdfs_site_path
Description	Absolute file path of hdfs-site.xml if Tsfile and related data files are stored in HDFS.
Type	String
Default	/etc/hadoop/conf/hdfs-site.xml
Effective	After restart system

- hdfs_ip

Name	hdfs_ip
Description	IP of HDFS if Tsfile and related data files are stored in HDFS. **If there are more than one hdfs_ip in configuration, Hadoop HA is used.**
Type	String
Default	localhost
Effective	After restart system

- hdfs_port

Name	hdfs_port
Description	Port of HDFS if Tsfile and related data files are stored in HDFS
Type	String
Default	9000
Effective	After restart system

- dfs_nameservices

Name	hdfs_nameservices
Description	Nameservices of HDFS HA if using Hadoop HA
Type	String
Default	hdfsnamespace
Effective	After restart system

- dfs_ha_namenodes

Name	hdfs_ha_namenodes
Description	Namenodes under DFS nameservices of HDFS HA if using Hadoop HA
Type	String
Default	nn1,nn2
Effective	After restart system

- dfs_ha_automatic_failover_enabled

Name	dfs_ha_automatic_failover_enabled
Description	Whether using automatic failover if using Hadoop HA
Type	Boolean
Default	true
Effective	After restart system

- dfs_client_failover_proxy_provider

Name	dfs_client_failover_proxy_provider
Description	Proxy provider if using Hadoop HA and enabling automatic failover
Type	String
Default	org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvic
Effective	After restart system

- hdfs_use_kerberos

Name	hdfs_use_kerberos
Description	Whether use kerberos to authenticate hdfs
Type	String
Default	false
Effective	After restart system

- kerberos_keytab_file_path

Name	kerberos_keytab_file_path
Description	Full path of kerberos keytab file

Type	String
Default	/path
Effective	After restart system

- kerberos_principal

Name	kerberos_principal
Description	Kerberos pricipal
Type	String
Default	your principal
Effective	After restart system

Start server, and Tsfile will be stored on HDFS.

To reset storage file system to local, just edit configuration `tsfile_storage_fs` to `LOCAL`. In this situation, if data files are already on HDFS, you should either download them to local and move them to your config data file folder (`../server/target/iotdb-server-0.11.1/data/data` by default), or restart your process and import data to IoTDB.

Frequent questions

1. What Hadoop version does it support?

A: Both Hadoop 2.x and Hadoop 3.x can be supported.

1. When starting the server or trying to create timeseries, I encounter the error below:

```
ERROR org.apache.iotdb.tsfile.fileSystem.fsFactory.HDFSFactory:62 - Failed to get Hadoop file system. Please
1. check your dependency of Hadoop module.
```

A: It indicates that you forget to put Hadoop module dependency in IoTDB server. You can solve it by:

- Build Hadoop module: `mvn clean package -pl hadoop -am -Dmaven.test.skip=true`
- Copy the target jar of Hadoop module `hadoop-tsfile-0.11.1-jar-with-dependencies.jar` into server target lib folder `.../server/target/iotdb-server-0.11.1/lib`.

Shared Nothing Architecture

Shared Nothing Architecture is under development. Please wait patiently and look forward to it.

- Comparison

Comparison

Known Time Series Database

As the time series data becomes more and more important, several open sourced time series databases are introduced to the world. However, few of them are developed for IoT or IIoT (Industrial IoT) scenario in particular.

3 kinds of TSDBs are compared here.

- InfluxDB - Native Time series database

  InfluxDB is one of the most popular TSDBs.

  Interface: InfluxQL and HTTP API

- OpenTSDB and KairosDB - Time series database based on NoSQL

  These two DBs are similar, while the first is based on HBase and the second is based on Cassandra. Both of them provides RESTful style API.

  Interface: Restful API

- TimesacleDB - Time series database based on Relational Database

  Interface: SQL

Prometheus and Druid are also famous for time series data management. However, Prometheus focuses data collection, data visualization and alert warnings. Druid focuses on data analysis with OLAP workload. We omit them here.

Comparison

The above time series databases are compared from two aspects: the feature comparison and the performance comparison.

Feature Comparison

I list the basic features comparison of these databases.

Legend:

- O: big support greatly
- o: support
- x: not support
- ☹ : support but not very good

- ?: unknown

Basic Features

TSDB	IoTDB	InfluxDB	OpenTSDB	KairosDB	TimescaleDB
OpenSource	**o**	o	o	**o**	o
SQL-like	o	o	x	x	**o**
Schema	“Tree-based, tag-based”	tag-based	tag-based	tag-based	Relational
Writing out-of-order data	o	o	o	o	o
Schema-less	o	o	o	o	o
Batch insertion	o	o	o	o	o
Time range filter	o	o	o	o	o
Order by time	**o**	o	x	x	o
Value filter	o	o	x	x	o
Downsampling	**o**	o	o	o	o
Fill	**o**	o	o	x	o
LIMIT	o	o	o	o	o
SLIMIT	o	o	x	x	?
Latest value	O	o	o	x	o

Details

- OpenSource:
  - IoTDB uses Apache License 2.0.
  - InfluxDB uses MIT license. However, **the cluster version is not open sourced**.
  - OpenTSDB uses LGPL2.1, which **is not compatible with Apache License**.
  - KairosDB uses Apache License 2.0.
  - TimescaleDB uses Timescale License, which is not free for enterprise.
- SQL like:
  - IoTDB and InfluxDB support SQL like language. In addition, the integration of IoTDB and Calcite is almost done (a PR has been submitted), which means IoTDB will support Standard SQL soon.
  - OpenTSDB and KairosDB only support Rest API, while IoTDB also supports Rest API (a PR has been submitted).
  - TimescaleDB uses the SQL the same as PG.
- Schema:
  - IoTDB: IoTDB proposes a Tree based schema (opens new window). It is quite

different from other TSDBs. However, the kind of schema has the following advantages:

- In many industrial scenarios, the management of devices are hierarchical, rather than flat. That is why we think a tree based schema is better than tag-value based schema.
- In many real world applications, tag names are constant. For example, a wind turbine manufacturer always identify their wind turbines by which country it locates, the farm name it belongs to, and its ID in the farm. So, a 4-depth tree (“root.the-country-name.the-farm-name.the-id”) is fine. You do not need to repeat to tell IoTDB the 2nd level of the tree is for country name, the 3rd level is for farm id, etc.
- A path based time series ID definition also supports flexible queries, like “root.*.a.b.*“, where * is wildcard character.

- InfluxDB, KairosDB, OpenTSDB are tag-value based, which is more popular currently.
- TimescaleDB uses relational table.

- Order by time:

  Order by time seems quite trivial for time series database. But… if we consider another feature, called align by time, something becomes interesting. And, that is why we mark OpenTSDB and KairosDB unsupported.

  Actually, in each time series, all these TSDBs support order data by timestamps.

  However, OpenTSDB and KairosDB do not support order data from different timeseries in the time order.

  Ok, consider a new case: I have two time series, one is for the wind speed in wind farm1, another is for the generated energy of wind turbine1 in farm1. If we want to analyze the relation between the wind speed and the generated energy, we have to know the values of both at the same time. That is to say, we have to align the two time series in the time dimension.

  So, the result should be:

timestamp	wind speed	generated energy
1	5.0	13.1
2	6.0	13.3
3	null	13.1

  or,

timestamp	series name	value

1	wind speed	5.0
1	generated energy	13.1
2	wind speed	6.0
2	generated energy	13.3
3	generated energy	13.1

Though the second table format does not align data by the time dimension, it is easy to be implemented in the client-side, by just scanning data row by row.

IoTDB supports the first table format (called align by time), InfluxDB supports the second table format.

- Downsampling:

  Downsampling is for changing the granularity of timeseries, e.g., from 10Hz to 1Hz, or 1 point per day.

  Different from other systems, IoTDB downsamples data in real time, while others serialized downsampled data on disk.
  That is to say,

  - IoTDB supports **adhoc** downsampling data in **arbitrary time**. e.g., a SQL returns 1 point per 5 minutes and start with 2020-04-27 08:00:00 while another SQL returns 1 point per 5 minutes + 10 seconds and start with 2020-04-27 08:00:01. (InfluxDB also supports adhoc downsampling but the performance is ….. hm)
  - There is no disk loss for IoTDB.

- Fill:

  Sometimes we thought the data is collected in some fixed frequency, e.g., 1Hz (1 point per second). But usually, we may lost some data points, because the network is unstable, the machine is busy, or the machine is down for several minutes.

  In this case, filling these holes is important. Data scientists can avoid to many so called dirty work, e.g., data clean.

  InfluxDB and OpenTSDB only support using fill in a group by statement, while IoTDB supports to fill data when just given a particular timestamp. Besides, IoTDB supports several strategies for filling data.

- Slimit:

  Slimit means return limited number of measurements (or, fields in InfluxDB). For example, a wind turbine may have 1000 measurements (speed, voltage, etc..), using slimit and soffset can just return a part of them.

- Latest value:

> As one of the most basic timeseries based applications is monitoring the latest data. Therefore, a query to return the latest value of a time series is very important. IoTDB and OpenTSDB support that with a special SQL or API, while InfluxDB supports that using an aggregation function. (the reason why IoTDB provides a special SQL is IoTDB optimizes the query expressly.)

Conclusion:

Well, if we compare the basic features, we can find that OpenTSDB and KairosDB somehow lack some important query features. TimescaleDB can not be freely used in business. IoTDB and InfluxDB can meet most requirements of time series data management, while they have some difference.

Advanced Features

I listed some interesting features that these systems may differ.

TSDB	IoTDB	InfluxDB	OpenTSDB	KairosDB	TimescaleDB
Align by time	**O**	o	x	x	o
Compression	**O**	☹	☹	☹	☹
MQTT support	**O**	o	x	x	☹
Run on Edge-side Device	**O**	o	x	☹	o
Multi-instance Sync	**O**	x	x	x	x
JDBC Driver	**o**	x	x	x	x
Standard SQL	o	x	x	x	**O**
Spark integration	**O**	x	x	x	x
Hive integration	**O**	x	x	x	x
Writing data to NFS (HDFS)	**O**	x	o	x	x
Flink integration	**O**	x	x	x	x

- Align by time: have been introduced. Let’s skip it..
- Compression:
  - IoTDB supports many encoding and compression for time series, like RLE, 2DIFF, Gorilla, etc.. and Snappy compression. In IoTDB, you can choose which encoding method you want, according to the data distribution. For more info, see here (opens new window).
  - InfluxDB also supports encoding and compression, but you can not define which encoding method you want. It just depends on the data type. For more info, see here (opens new window).
  - OpenTSDB and KairosDB use HBase and Cassandra in backend, and have no special encoding for time series.

- MQTT protocol support:

  MQTT protocol is an international standard and widely known in industrial users. only IoTDB and InfluxDB support user using MQTT client to write data.

- Running on Edge-side Device:

  Nowdays, edge computing is more and more popular, which means the edge device has more powerful computational resources. Deploying a TSDB on the edge side is useful for managing data on the edge side and serve for edge computing. As OpenTSDB and KairosDB rely another DB, the architecture is heavy. Especially, it is hard to run Hadoop on the edge side.

- Multi-instance Sync:

  Ok, now we have many TSDB instances on the edge-side. Then, how to upload their data to the data center, to form a … data lake (or ocean, river,…, whatever). One solution is to read data from these instances and write the data point by point to the data center instance. IoTDB provides another choice, which is just uploading the data file into the data center incrementally, then the data center can support service on the data.

- JDBC driver:

  Now only IoTDB supports a JDBC driver (though not all interfaces are implemented), and makes it possible to integrate many other JDBC driver based softwares.

- Standard SQL:

  As mentioned before, the integration of IoTDB and Calcite is almost done (a PR has been submitted), which means IoTDB will support Standard SQL.

- Spark and Hive integration:

  It is very important that letting big data analysis software to access the data in database for more complex data analysis. IoTDB supports Hive-connector and Spark connector for better integration.

- Writing data to NFS (HDFS): Sharing nothing architecture is good, but sometimes you have to add new servers even your CPU and memory is idle but the disk is full… Besides, if we can save the data file directly to HDFS, it will be more easy to use Spark and other softwares to analyze data, without ETL.

- IoTDB supports writing data locally or on HDFS directly. IoTDB also allows user to extend to store data on other NFS.

- InfluxDB, KairosDB have to write data locally.

- OpenTSDB has to write data on HDFS.

Conclusion:

We can find that IoTDB has many powerful features that other TSDBs do not support.

Performance Comparison

Ok… If you say, “well, I just want the basic features. IoTDB has little difference from others.”. It is somehow right. But, if you consider the performance, you may change your mind.

quick review

Given a workload:

- Write:

10 clients write data concurrently. The number of storage group is 50. There are 1000 devices and each device has 100 measurements (i.e.,, 100K time series totally). The data type is float and IoTDB uses RLE encoding and Snappy compression. IoTDB uses batch insertion API and the batch size is 100 (write 100 data points per write API call).

- Read:

50 clients read data concurrently. Each client just read data from 1 device with 10 measurements in one storage group.

IoTDB is v0.9.0.

Write performance:

We write 112GB data totally.

The write throughput (points/second) is:

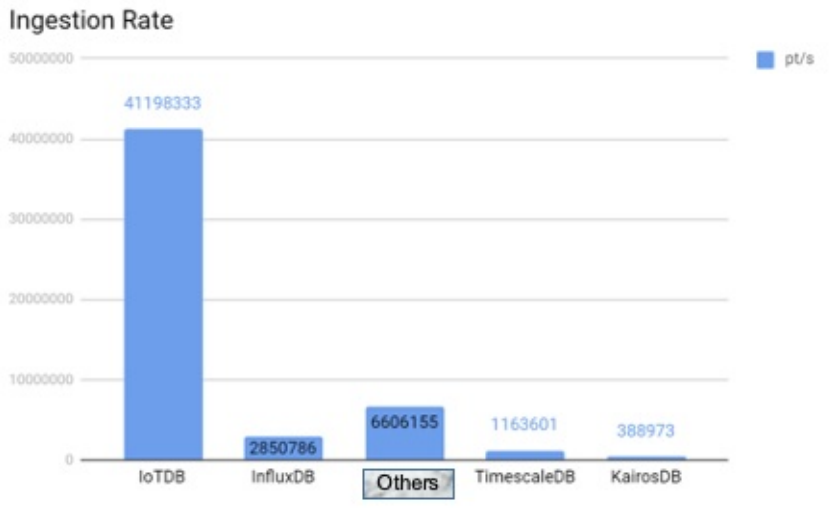


Figure 1. Write throughput (points/second) IoTDB v0.9

The disk occupation is:

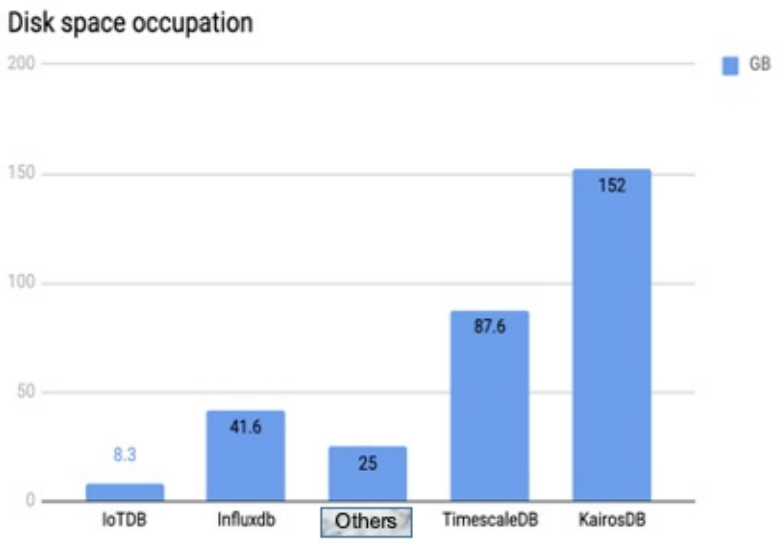


Figure 2. Disk occupation(GB) IoTDB v0.9

Query performance

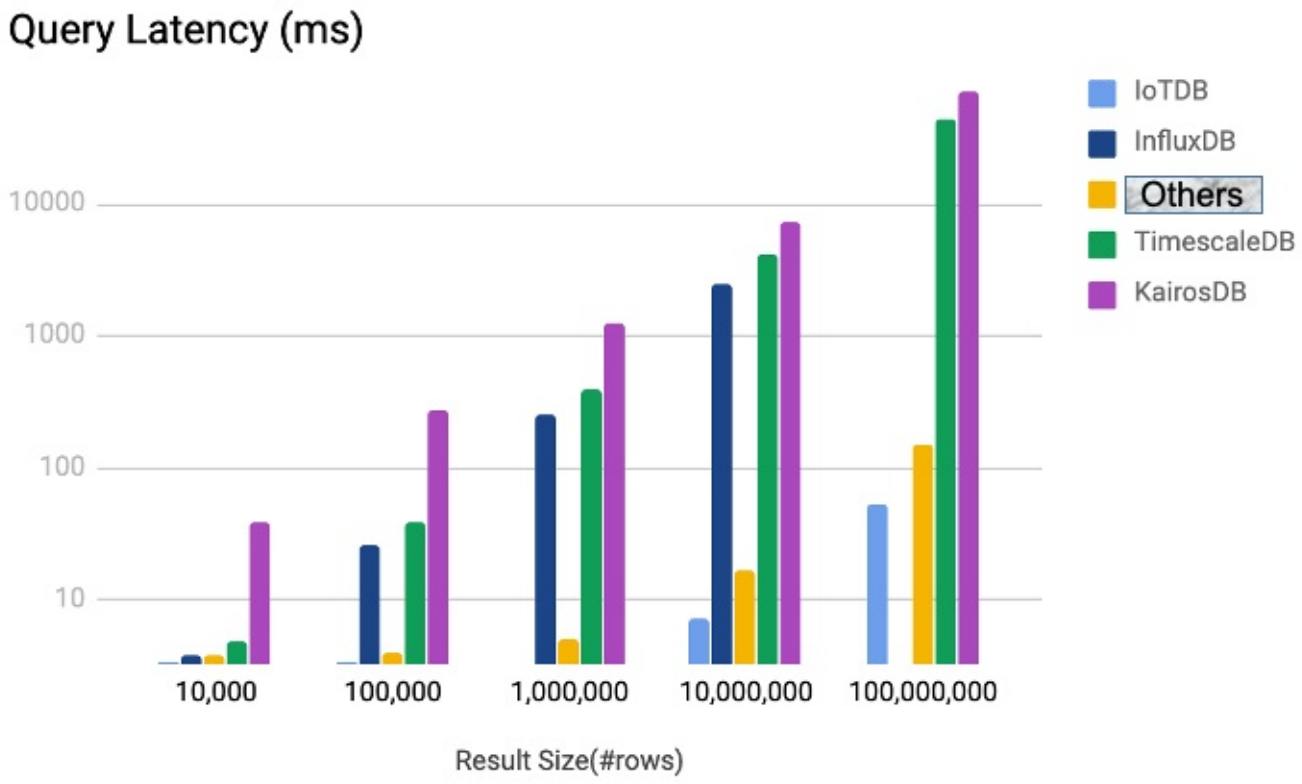


Figure 3. Aggregation query time cost(ms) IoTDB v0.9

We can see that IoTDB outperforms others.

More details

We provide a benchmarking tool, called IoTDB-benchamrk (https://github.com/thulab/iotdb-benchmark, you may have to use the dev branch to compile it), it supports IoTDB, InfluxDB, KairosDB, TimescaleDB, OpenTSDB. We have a article (opens new window) for comparing these systems using the benchmark tool. When we publish the article, IoTDB just entered Apache incubator, so we deleted the performance of IoTDB in that article. But after comparison, some results are presented here.

- **IoTDB: 0.8.0**. (notice: **IoTDB v0.9 outperforms than v0.8**, the result will be updated once experiments on v0.9 are finished)
- InfluxDB: 1.5.1.
- OpenTSDB: 2.3.1 (HBase 1.2.8)
- KairosDB: 1.2.1 (Cassandra 3.11.3)
- TimescaleDB: 1.0.0 (PostgreSQL 10.5)

All TSDB run on the same server one by one.

- For InfluxDB, we set the cache-max-memory-size and max-series-perbase as unlimited (otherwise it will be timeout quickly)
- For OpenTSDB, we modified tsd.http.request.enable_chunked, tsd.http.request.max_chunk and tsd.storage.fix_duplicates for supporting write data in batch and write out-of-order data.
- For KairosDB, we set Cassandra’s read_repair_chance as 0.1 (However it has no effect because we just have one node).
- For TimescaleDB, we use PGTune tool to optimize PostgreSQL.

All TSDBs run on a server with Intel Xeon CPU E5-2697 v4 @2.3GHz, 256GB memory and 10 HDD disks with RAID-5. The OS is Ubuntu 16.04.2 LTS, 64bits.

Another server run IoTDB benchmark tool.

I omit the detailed workload here, let’s see the result:

Legend:

- I: InfluxDB
- O: OpenTSDB
- T: TimescaleDB
- K: KairosDB
- **D: IoTDB**

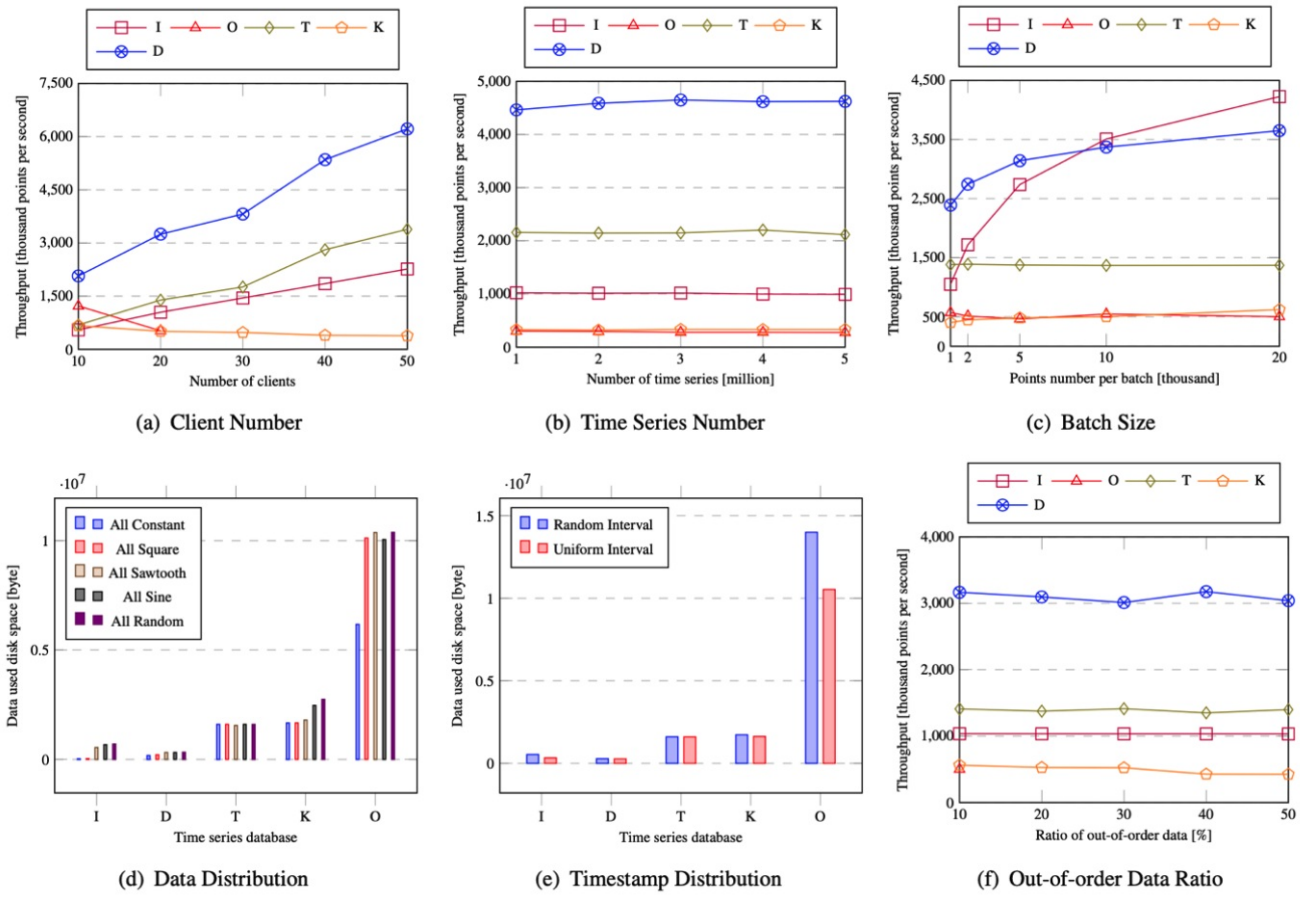


Figure 4. Write experiments IoTDB v0.8.0

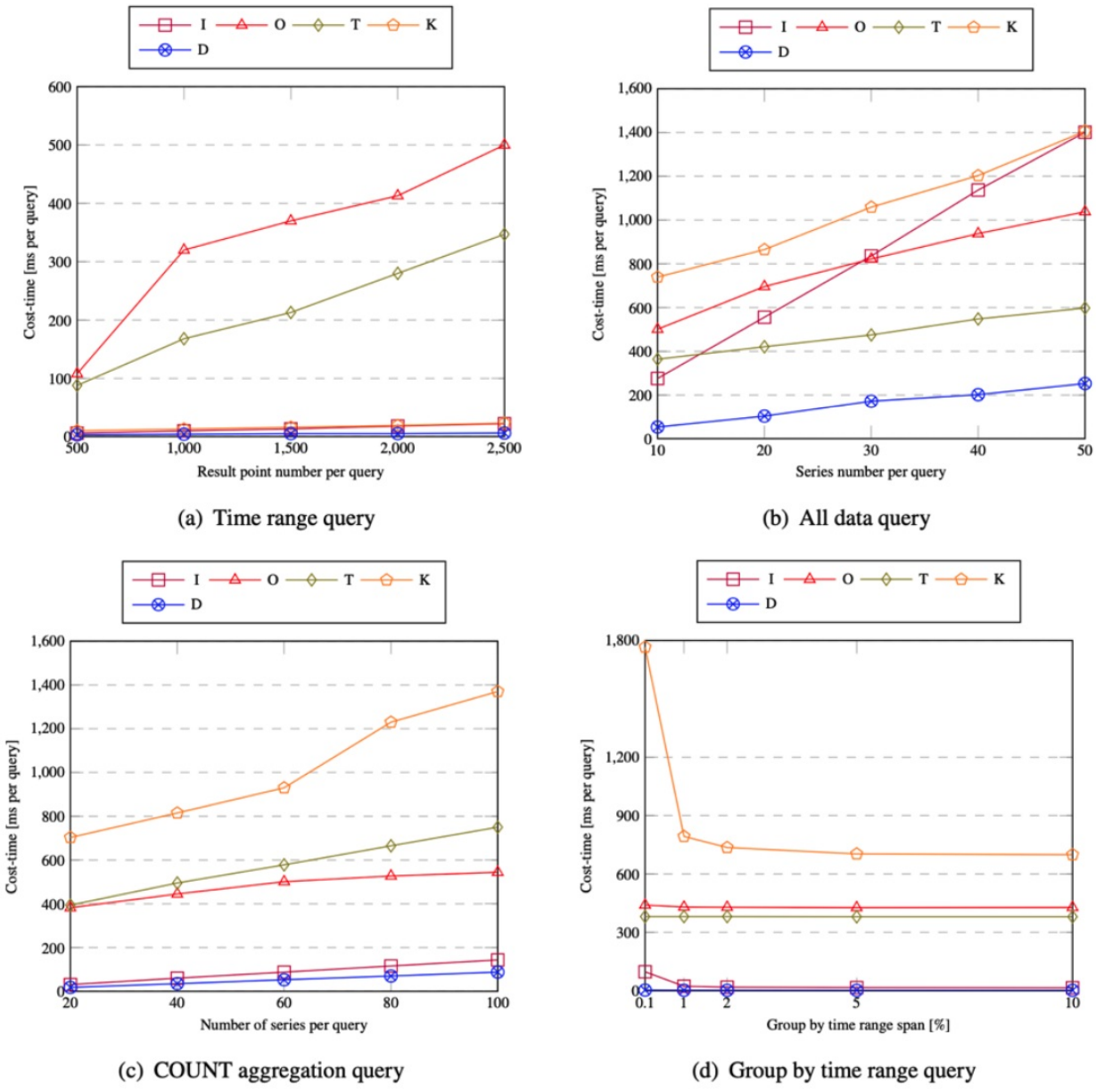


Figure 5. Query experiments IoTDB v0.8.0

We can see that IoTDB outperforms others hugely.

In Figure. 4(c), when the batch size reaches to 10000 points, InfluxDB is better than IoTDB v0.8. It is because in IoTDB v0.8, batch insert API is not optimized.

From IoTDB v0.9 on, using batch insert API can obtain 8 to 10 times write performance improvement.

For example, using IoTDB v0.8, the write throughput can only reach to 6 million data points per second. But using IoTDB v0.9, the write throughput can reach to 40 million data points per second on the same server with the same workload. (see Figure. 4(a) vs Figure. 1).

Conclusion

If you are considering a TSDB for your IIoT application, Apache IoTDB, a new time series, is your best choice.

We will update this page once we release new version and finish the experiments. We also welcome more contributors correct this article and contribute IoTDB and reproduce experiments.