ALIBABA CLOUD

Alibaba Cloud

Tablestore

CLI

Document Version: 20230209

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Download the Tablestore CLI

The Tablestore CLI provides simple and clear commands that you can run in Windows, Linux, and macOS.

The following table provides the download links of the Tablestore CLI for different operating systems.

Operating system	Download link
Windows	Windows10
Linux	• Linux (AMD64) • Linux (ARM64)
macOS	macOS

2.Start the Tablestore CLI and configure access information

This topic describes how to start the Tablestore CLI and configure the information that is required to access Tablestore, such as the AccessKey pair and Tablestore instance.

Prerequisites

An AccessKey pair that consists of an AccessKey ID and an AccessKey secret is obtained. For more information, see Obtain an AccessKey pair.

> **Note** If a RAM user is required to perform operations, make sure that the required permissions are granted to the RAM user. For more information, see Grant permissions to a RAM user.

Procedure

1. Decompress the Tablestore CLI package that you downloaded. Go to the root directory of the Tablestore CLI and select a method that is used to start the Tablestore CLI based on your operating system.
   - For Windows, double-click the ts.exe file.
   - For Linux or macOS, run the `./ts` command.

     > **Note** If you do not have execute permissions to run the command in Linux or macOS, run the `chmod 755 ts` command to obtain permissions and start the Tablestore CLI.

   The following code shows a sample Tablestore startup interface:

```
# Welcome to use Command Line Tool for Aliyun Tablestore. Current Version is '2021-11-11'
.
#    ________      _      _              _
#   |__    __|    | |    | |            | |
#      | |  __ _  | |__  | |  ___   ___ | |_   ___  _ __    ___
#      | | / _' || '_ \ | | / _ \ / __|| __| / _ \ | '__| / _ \
#      | || (_| || |_) || ||  __/ \__ \| |_ | (_) || |   |  __/
#      |_| \__,_||_.__/ |_| \___| |___/ \__| \___/ |_|    \___|
#
# Please visit our product website: https://www.aliyun.com/product/ots
# You can also join our DingTalk Chat Group (ID: 11789671 or 23307953) to discuss and ask
Tablestore related questions.
#
tablestore>
```

2. Configure access information.

   The following sample code shows how to configure access information by using an existing instance:

```
config --endpoint https://myinstance.cn-hangzhou.ots.aliyuncs.com --instance myinstance -
-id NTSVLeBHzgX2iZfcaXXPJ**** --key 7NR2DiotscDbauohSq9kSHX8BDp99bjs7eNpCR7o****
```

If no instance is created, you need only to specify the AccessKey ID and AccessKey secret. The following sample code shows how to specify the AccessKey ID and AccessKey secret:

```
config --id NTSVLeBHzgX2iZfcaXXPJ**** --key 7NR2DiotscDbauohSq9kSHX8BDp99bjs7eNpCR7o****
```

The following table describes the parameters that you can configure to specify access information.

Parameter	Required	Example	Description
--endpoint	No	https://myinstance.cn-hangzhou.ots.aliyuncs.com	The endpoint of the instance. For more information, see Endpoints. This parameter is required only if you use an existing instance to configure the access information.
--instance	No	myinstance	The name of the instance. This parameter is required only if you use an existing instance to configure the access information.
--id	Yes	NTSVLeBHzgX2iZfcaXXPJ****	The AccessKey ID and AccessKey secret of an Alibaba Cloud account or a RAM user.
--key	Yes	7NR2DiotscDbauohSq9kSHX8BDp99bjs7eNpCR7o****	

3.Instance operations

This topic describes how to use the Tablestore CLI to manage Tablestore instances.

Activate Tablestore

If Tablestore is activated, skip this operation. You must activate Tablestore only once. You are not charged when you activate Tablestore.

Run the following command to activate Tablestore:

```
enable_service
```

The following result is returned:

```
Your service is enabled.
```

Create an instance

Create an instance in a specified region.

> **Important** The instance name must be globally unique within the region. If an error indicating that the instance name that you specify is the same as the name of an existing instance occurs, specify a new instance name.

- Command syntax

  ```
  create_instance -d description -n instanceName -r regionId
  ```

  The following table describes the parameters that you can configure to create an instance in a specified region.

Parameter	Required	Example	Description
-n	Yes	myinstance	The name of the instance. For more information, see Instance.
-r	Yes	cn-hangzhou	The region ID of the instance. For more information, see Region.
-d	No	"First instance created by CLI."	The description of the instance.

- Example

  The following sample code shows how to create an instance named myinstance in the China (Hangzhou) region:

  ```
  create_instance -d "First instance created by CLI." -n myinstance -r cn-hangzhou
  ```

Query information about an instance

Query information about an instance, such as the instance name, creation time, and the ID of the account to which the instance belongs.

- Command syntax

```
describe_instance -r regionId -n instanceName
```

- Examples

  The following sample code shows how to query information about the instance named myinstance in the China (Hangzhou) region:

```
describe_instance -r cn-hangzhou -n myinstance
```

  The following result is returned:

```
{
    "Status": 1,
    "WriteCapacity": 5000,
    "ReadCapacity": 5000,
    "ClusterType": "SSD",
    "Timestamp": "",
    "UserId": "643941",
    "InstanceName": "myinstance",
    "CreateTime": "2021-10-31 14:19:43",
    "Network": "NORMAL",
    "Description": "First instance created by CLI.",
    "Quota": {
        "EntityQuota": 64
    },
    "TagInfos": {
        "TagInfo": []
    }
}
```

Query instances

Query all instances in a specified region.

- Command syntax

```
list_instance -r regionId
```

- Examples

  The following sample code shows how to query all instances in the China (Hangzhou) region:

```
list_instance -r cn-hangzhou
```

  The following table describes the parameter that you can configure to query all instances in a specified region.

Parameter	Required	Example	Description
-r	Yes	cn-hangzhou	The ID of the region.

  The following result is returned:

> **Note** If no instance is created in the specified region, the returned result is empty.

```
[
    "myinstance"
]
```

Configure an instance

Specify the endpoint of the instance based on the type of your network.

- Command syntax

```
config --endpoint endpoint --instance instanceName
```

The following table describes the parameters that you can configure for an instance.

Parameter	Required	Example	Description
--endpoint	Yes	http://myinstance.cn-hangzhou.ots.aliyuncs.com	The endpoint of the instance, which supports two network types: Internet and virtual private cloud (VPC). Specify the endpoint of the instance based on your business requirements. The format of the endpoint must comply with the following rules: ◦ Public: http(s)://<instance_name>.<region_id>.ots.aliyuncs.com ◦ VPC: http(s)://<instance_name>.<region_id>.vpc.tablestore.aliyuncs.com
--instance	Yes	myinstance	The name of the instance.

- Examples

The following sample code shows how to configure the public endpoint for the instance named myinstance in the China (Hangzhou) region:

```
config --endpoint http://myinstance.cn-hangzhou.ots.aliyuncs.com --instance myinstance
```

The following result is returned:

```
{
    "Endpoint": "http://myinstance.cn-hangzhou.ots.aliyuncs.com",
    "AccessKeyId": "NTSVLeBHzgX2iZfcaXXPJ****",
    "AccessKeySecret": "7NR2DiotscDbauohSq9kSHX8BDp99bjs7eNpCR7o****",
    "Instance": "myinstance"
}
```

The following table describes the response parameters.

Parameter	Required	Example	Description
Endpoint	No	https://myinstance.cn-hangzhou.ots.aliyuncs.com	The endpoint of the instance. For more information, see Endpoints.
Instance	No	myinstance	The name of the instance.
AccessKeyId	Yes	NTSVLeBHzgX2iZfcaXXPJ****	The AccessKey ID and AccessKey secret of an Alibaba Cloud account or a RAM user.
AccessKeySecret	Yes	7NR2DiotscDbauohSq9kSHX8BDp99bjs7eNpCR7o****	

4.Wide Column model

4.1. Operations on data tables

After you create a data table, you can use the data table, query information about the data table, list the names of data tables, update the data table, and delete the data table.

> **Note** For more information about the Wide Column model, see Wide Column model.

Create a table

Create a data table for which parameters such as primary key columns and time to live (TTL) are specified. You can also import a configuration file in the JSON format to create a data table.

Command syntax

```
create -t tableName --pk '[{"c":"<primaryKeyName>", "t":"<primaryKeyType>"}, {"c":"<primary
KeyName>", "t":"<primaryKeyType>", "opt":"<options>"}]'--ttl <timeToLive> --version <maxVer
sion>
```

The following table describes the parameters that you can configure to create a data table.

Parameter	Required	Example	Description
-m, --mode	No	widecolumn	The type of the table that you want to create. Default value: widecolumn. Valid values: • widecolumn: data table. • timeseries: time series table.
-t, --table	Yes	mytable	The name of the data table.

-p, --pk	Yes	[{"c":"uid","t":"string"}, {"c":"pid","t":"integer"}]	The primary key column settings of the data table. The value of this parameter is a JSON array. The following fields are supported: • c: required. The name of the primary key column. • t: required. The type of the primary key column. Valid values: string, integer, and binary. • opt: optional. The optional configuration item. Valid values: none and auto. Default value: none. If you set opt to auto, the primary key column is an auto-increment primary key column. For more information about auto-increment primary key columns, see Auto-increment of primary key columns. **Important** You do not need to define attribute columns when you create a data table. The attribute columns differ with each row, and the name of an attribute column is specified when you write data. For more information about how to write data to a data table, see Operations on data. You can add one to four primary key columns. The first primary key column is the partition key. After you create a data table, you cannot modify the configurations and the order of primary key columns.
--ttl	No	864000	The duration during which the data in the data table can be retained. If the retention period exceeds the TTL value, Tablestore automatically deletes expired data. Unit: seconds. If you do not specify a value for this parameter, the default value -1 is used. The value of -1 specifies that data never expires. The value of this parameter must be -1 or a value that is equal to or greater than 86400 seconds (one day).
--version	No	1	The maximum number of versions that can be retained for data in attribute columns of the data table. If the number of versions of data in attribute columns exceeds the value of this parameter, the system deletes data of earlier versions. If you do not specify a value for this parameter, the default value 1 is used. The value of 1 specifies that only data of the latest version is retained. The value of this parameter for an attribute column is a positive integer.

--read_cu	No	0	The reserved read throughput and reserved write throughput of the data table. The default value 0 specifies that all throughput is billed based on the pay-as-you-go billing method. Unit: capacity unit (CU). You can set the reserved read throughput or reserved write throughput only to 0 for data tables in capacity instances. Reserved throughput does not apply to these instances. • If you set the reserved read throughput or reserved write throughput to a value that is greater than 0 for a data table, Tablestore reserves resources for the data table. After you create the data table, Tablestore charges you for the reserved throughput. Additional throughput is billed based on the pay-as-you-go billing method. • If you set the reserved read throughput or reserved write throughput to 0, Tablestore does not reserve resources for the data table.
--write_cu	No	0	
-i, --input	No	/tmp/create_table_meta.json	The path of the configuration file that is used to create a data table. The configuration file must be in JSON format.

You can also run the following commands to create a configuration file:

- Windows

```
create -i D:\\localpath\\filename.json
```

- Linux and macOS

```
create -i /localpath/filename.json
```

The following example shows the content of a configuration file:

```
{
    "Name": "mytable",
    "Meta": {
        "Pk": [
            {
                "C": "uid",
                "T": "string",
                "Opt": "none"
            },
            {
                "C": "pid",
                "T": "integer",
                "Opt": "none"
            }
        ]
    },
    "Option": {
        "TTL": 864000,
        "Version": 3
    },
    "CU": {
        "Read": 0,
        "Write": 0
    }
}
```

Examples

Create a data table named mytable. The data table contains the uid and pid primary key columns. The uid column is of the STRING type. The pid column is of the INTEGER type. Data in the data table never expires.

```
create -t mytable --pk '[{"c":"uid", "t":"string"}, {"c":"pid", "t":"integer"}]'
```

Create a data table named mytable. The data table contains the uid and pid primary key columns. The uid column is of the STRING type. The pid column is of the INTEGER type. The pid column is specified as an auto-increment primary key column. Data in the data table never expires.

```
create -t mytable --pk '[{"c":"uid", "t":"string"}, {"c":"pid", "t":"integer", "opt":"auto"
}]'
```

Create a data table named mytable. The data table contains the uid and pid primary key columns. The uid column is of the STRING type. The pid column is of the INTEGER type. The TTL is set to 864000 seconds (10 days). Only the data of the latest version is retained.

```
create -t mytable --pk '[{"C":"uid","t":"string"}, {"c":"pid","t":"integer"}]' --ttl 864000
--version 1
```

Use a data table

Select the data table on which you want to perform table operations or data operations.

Command syntax

```
use --wc -t <tableName>
```

The following table describes the parameters that you must configure to use a data table.

Parameter	Required	Example	Description
--wc	No	N/A	Specifies that the table on which you want to perform operations is a data table.
-t, --table	Yes	mytable	The name of the data table.

Example

The following sample code shows how to use the data table named mytable:

```
use -t mytable
```

List the names of tables

List the names of all tables, all data tables, or all time series tables in an instance.

- List the names of all tables of the same type as the current table

  ```
  list
  ```

- List the names of all tables

  ```
  list -a
  ```

- List the names of all data tables

  ```
  list -w
  ```

- List the names of all time series tables

  ```
  list -t
  ```

The following table describes the parameters that you can configure to list the names of tables.

Parameter	Required	Example	Description
-a, --all	No	N/A	Lists the names of all tables.
-d, --detail	No	N/A	Lists the details about the tables.
-w, --wc	No	N/A	Lists the names of all data tables.
-t, --ts	No	N/A	Lists the names of all time series tables.

Update a data table

Update information about a data table, such as TTL and max versions.

Command syntax

```
alter -t <tableName> --ttl <timeToLive> --version <maxVersion> --read_cu <readCU> --write_cu <writeCU>
```

Example

Change the TTL of the data table named mytable to 86400 seconds (one day). Set the max versions to 1, reserved read CU to 0, and reserved write CU to 0.

```
alter -t mytable --ttl 86400 --version 1 --read_cu 0 --write_cu 0
```

Query the information about a data series table

You can query information about a data table and then save the information to a JSON file on your local PC.

Command syntax

```
desc -t <tableName> -o /localpath/filename.json
```

The following table describes the parameters that you can configure to query the information about a data table.

Parameter	Required	Example	Description
-t, --table	No	mytable	The name of the data table.
-f, --print_format	No	json	The output format of the information about the data table. Default value: json. Valid values: json and table.
-o, --output	No	/tmp/describe_table_meta.json	The path of the local JSON file in which the information about the data table is stored.

Examples

The following sample code shows how to query information about the current table.

```
desc
```

The following sample code shows how to query information about the current data table and save the information to the local file named describe_table_meta.json:

```
desc -o  /tmp/describe_table_meta.json
```

Delete a data table

Delete a data table that you no longer need.

Command syntax

```
drop -t <tableName> -y
```

The following table describes the parameters that you must configure to delete a data table.

Parameter	Required	Example	Description
--t, --table	Yes	mytable	The name of the data table.
-y, --yes	Yes	N/A	Specifies that the confirmation information must appear. This parameter is required.

Example

The following sample code shows how to delete a data table named mytable.

```
drop -t mytable -y
```

4.2. Operations on data

You can use the Tablestore CLI to insert a row of data into a data table, update a row of data in a data table, read data and delete a row of data from a data table, scan data in a data table, import data to a data table, and export data from a data table.

Insert a row of data

You can run the following command to insert a row of data into a data table. Alternatively, you can import a JSON configuration file to insert a row of data into a data table.

- Command syntax

  ```
  put --pk '[primaryKeyValue, primaryKeyValue]' --attr '[{"c":"attributeColumnName", "v":"attributeColumnValue"}, {"c":"attributeColumnName", "v":"attributeColumnValue", "ts":timestamp}]' --condition condition
  ```

  The following table describes the parameters you can configure to insert a row of data into a data table.

Parameter	Required	Example	Description

-p, --pk	Yes	["86", 6771]	The values of the primary key columns of the data table. The values are included in an array. **Important** ◦ The number and types of the primary key columns that you specify must be the same as the actual number and types of primary key columns in the data table. ◦ If a primary key column is an auto-increment primary key column, you need only to set the value of the auto-increment primary key column to the placeholder null.

--attr	Yes	[{"c":"name", "v":"redchen"}, {"c":"country", "v":"china", "t":"string", "ts":1626860469000}]	The settings of the attribute column of the data table. The value of this parameter is a JSON array. Each attribute column is configured with the following fields: ◦ c: required. The name of the attribute column. ◦ v: required. The value of the attribute column. ◦ t: optional. The type of the attribute column. Valid values: integer, string, binary, boolean, and double. If you set this field to string, the value of the attribute column is a string encoded in UTF-8. If you set the type of the attribute column to binary, this field is required. ◦ ts: optional. The timestamp is the version number of the data. The timestamp can be automatically generated or customized. If you do not configure this field, Tablestore automatically generates a timestamp. For more information, see Data versions and TTL.
--condition	No	ignore	The row existence condition of conditional update that is used to determine whether to insert a row of data. Default value: ignore. Valid values: ◦ ignore: Data is inserted regardless of whether the row exists. If the row exists, existing data is overwritten by the inserted data. ◦ exist: Data is inserted only when the row exists. Existing data is overwritten by the inserted data. ◦ not_exist: Data is inserted only when the row does not exist. For more information about conditional update, see Conditional update.

-i, --input	No	/temp/inputdata.json	The path of the JSON configuration file that is used to insert data.

You can also use the configuration file to insert data. The command syntax varies based on the operating system.

- Windows

```
put -i D:\\localpath\\filename.json
```

- Linux and macOS

```
put -i /localpath/filename.json
```

The following example shows the content of a configuration file:

```
{
    "PK":{
        "Values":[
            "86",
            6771
        ]
    },
    "Attr":{
        "Values":[
            {
                "C":"age",
                "V":32,
                "TS":1626860801604,
                "IsInt":true
            }
        ]
    }
}
```

- Examples

  Insert a row of data into a data table. The value of the first primary key column in the row is "86". The value of the second primary key column in the row is 6771. The row contains the following two attribute columns: name and country. The name and country columns are of the string type.

```
put --pk '["86", 6771]' --attr '[{"c":"name", "v":"redchen"}, {"c":"country", "v":"china"
}]'
```

  Insert a row of data into a data table. The value of the first primary key column in the row is "86". The value of the second primary key column in the row is 6771. The row contains the following two attribute columns: name and country. The name and country columns are of the string type. Data is inserted regardless of whether the row exists. If the row exists, the inserted data overwrites the existing data.

```
put --pk '["86", 6771]' --attr '[{"c":"name", "v":"redchen"}, {"c":"country", "v":"china"
}]'  --condition ignore
```

Insert a row of data into a data table. The value of the first primary key column in the row is "86". The value of the second primary key column in the row is 6771. The row contains the following two attribute columns: name and country. The name and country columns are of the string type. The timestamp of data in the country column is 15327798534.

```
put --pk '["86", 6771]' --attr '[{"c":"name", "v":"redchen"}, {"c":"country", "v":"china"
, "t":"string", "ts":15327798534}]'
```

Insert a row of data into a data table where the second primary key column is an auto-increment primary key column. The value of the first primary key column in the row is "86". The value of the second primary key column in the row is null. The row contains the following two attribute columns: name and country. The name and country columns are of the string type.

```
put --pk '["86", null]' --attr '[{"c":"name", "v":"redchen"}, {"c":"country", "v":"china"
}]'
```

Read data

You can read data from a data table and export the data to a local JSON file.

> **Note** If the row that you want to read does not exist, an empty result is returned.

- Command syntax

```
get --pk '[primaryKeyValue,primaryKeyValue]'
```

The following table describes the parameters you can configure to read a row of data.

Parameter	Required	Example	Description
-p, --pk	Yes	["86",6771]	The values of the primary key columns of the data table. The values are included in an array. **Important** The number and types of the primary key columns that you specify must be the same as the actual number and types of primary key columns in the data table.
--columns	No	name,uid	The set of columns that you want to read. You can specify the name of a primary key column or an attribute column. If you do not specify a column name, all data in the row is returned.
--max_version	No	1	The maximum number of data versions that you want to read.

--time_range_start	No	1626860469000	The version range of data that you want to read. time_range_start specifies the start timestamp and time_range_end specifies the end timestamp. The range includes the start value and excludes the end value.
--time_range_end	No	1626865270000	
--time_range_specific	No	1626862870000	The specific version of data that you want to read.
-o, --output	No	/tmp/querydata.json	The local path of the JSON file to which the query results are exported.

- Example

  Read a row of data in which the value of the first primary key column is "86" and the value of the second primary key column is 6771.

```
get --pk '["86",6771]'
```

Update a row of data

You can run the following command to update a row of data in a data table. Alternatively, you can import a JSON configuration file to update a row of data in a data table.

- Command syntax

```
update --pk '[primaryKeyValue, primaryKeyValue]' --attr '[{"c":"attributeColumnName", "v":"attributeColumnValue"}, {"c":"attributeColumnName", "v":"attributeColumnValue", "ts":timestamp}]' --condition condition
```

The following table describes the parameters you can configure to update a row of data.

Parameter	Required	Example	Description
-p, --pk	Yes	["86", 6771]	The values of the primary key columns of the data table. The values are included in an array. **Important** The number and types of the primary key columns that you specify must be the same as the actual number and types of primary key columns in the data table.

--attr	Yes	[{"c":"name", "v":"redchen"}, {"c":"country", "v":"china", "ts":15327798534}]	The settings of the attribute column of the data table. The value of this parameter is a JSON array. Each attribute column is configured with the following fields: ◦ c: required. The name of the attribute column. ◦ v: required. The value of the attribute column. ◦ t: optional. The type of the attribute column. Valid values: integer, string, binary, boolean, and double. If you set this field to string, the value of the attribute column is a string encoded in UTF-8. If you set the type of the attribute column to binary, this field is required. ◦ ts: optional. The timestamp is the version number of the data. The timestamp can be automatically generated or customized. If you do not configure this field, Tablestore automatically generates a timestamp.
--condition	No	ignore	The row existence condition of conditional update that is used to determine whether to update a row of data. Default value: ignore. Valid values: ◦ ignore: Data is inserted regardless of whether the row exists. If the row exists, existing data is overwritten by the inserted data. ◦ exist: Data is inserted only when the row exists. Existing data is overwritten by the inserted data. ◦ not_exist: Data is inserted only when the row does not exist. For more information about conditional update, see Conditional update.

-i, --input	No	/tmp/inputdata.json	The path of the JSON configuration file that is used to update data.

You can also use the configuration file to update data. The command syntax varies based on the operating system.

- Windows

```
update -i D:\\localpath\\filename.json
```

- Linux and macOS

```
update -i /localpath/filename.json
```

The following example shows the content of a configuration file:

```
{
    "PK":{
        "Values":[
            "86",
            6771
        ]
    },
    "Attr":{
        "Values":[
            {
                "C":"age",
                "V":32,
                "TS":1626860801604,
                "IsInt":true
            }
        ]
    }
}
```

- Example

  Update a row of data in which the value of the first primary key column is "86" and the value of the second primary key column is 6771. Data is inserted regardless of whether the row exists. If the row exists, the inserted data overwrites the existing data.

```
update --pk '["86", 6771]' --attr '[{"c":"name", "v":"redchen"}, {"c":"country", "v":"chi
na"}]'  --condition ignore
```

Delete a row of data

You can delete a row of data with a specified primary key.

- Command syntax

```
delete --pk '[primaryKeyValue,primaryKeyValue]'
```

- Example

  Delete a row of data in which the value of the first primary key column is "86" and the value of the second primary key column is 6771.

```
delete --pk '["86", 6771]'
```

Scan data

You can scan a data table to obtain all data or a specified number of rows of data in the data table.

- Command syntax

```
scan --limit limit
```

The following table describes parameters that you can configure to scan data in a data table.

Parameter	Required	Example	Description
--limit	No	10	Optional. The maximum number of rows that you want to return. If you do not configure this parameter, all data in the data table is scanned.

- Example

The following sample code shows how to scan data in a data table to return up to 10 rows of data:

```
scan --limit 10
```

Export data

You can export data from a data table to a local JSON file.

- Command syntax

```
scan -o /localpath/filename.json -c attributeColumnName,attributeColumnName,attributeColu
mnName
```

The following table describes the parameters you can configure to export data from a data table to a local file.

Parameter	Required	Example	Description
-c, --columns	Yes	uid,name	The set of columns that you want to export. You can specify the name of a primary key column or an attribute column. If you do not specify a column name, all data in the row is exported.
--max_version	No	1	The maximum number of data versions that can be exported.

--time_range_start	No	1626865596000	The version range of data that you want to export. time_range_start specifies the start timestamp and time_range_end specifies the end timestamp. The range includes the start value and excludes the end value.
--time_range_end	No	1626869196000	
--time_range_specific	No	1626867396000	The specific version of data that you want to export.
--backward	No	N/A	Specifies that the system sorts the exported data in descending order of primary keys.
-o, --output	Yes	/tmp/mydata.json	The local path of the JSON file to which the query results are exported.
-l, --limit	No	10	The maximum number of rows that you want to return in the query.
-b, --begin	No	1000	The value range of data that you want to export.
-e, --end	No	2000	

- Examples

  The following sample code shows how to export all data from the current table to the mydata.json local file:

  ```
  scan -o /tmp/mydata.json
  ```

  The following sample code shows how to export data from the uid and name columns of the current table to the mydata.json local file:

  ```
  scan -o /tmp/mydata.json -c uid,name
  ```

Import data

You can import data from a local JSON file to a data table.

- Command syntax

  ```
  import -i /localpath/filename.json --ignore_version
  ```

  The following table describes the parameters you can configure to import data from a local file to a data table.

Parameter	Required	Example	Description

-a, --action	No	put	The mode in which data is imported. Default value: put. Valid values: ◦ put: If the row exists, all versions of data in all columns of the existing row are deleted and a new row of data is written to the data table. ◦ update: If the row exists, attribute columns can be added to or deleted from the row, a specified version of data in an attribute column can be deleted, or the existing data in an attribute column can be updated. If the row does not exist, a new row of data is added.
-i, --input	Yes	/tmp/inputdata.json	The path of the local JSON file from which data is imported to the current data table.
--ignore_version	No	N/A	The Tablestore CLI ignores timestamp checks. The current time is used as the timestamp.

The following example shows the content of a local configuration file:

```
{"PK":{"Values":["redchen",0]},"Attr":{"Values":[{"C":"country","V":"china0"},{"C":"name"
,"V":"redchen0"}]}}
{"PK":{"Values":["redchen",1]},"Attr":{"Values":[{"C":"country","V":"china1"},{"C":"name"
,"V":"redchen1"}]}}
```

- Examples

  The following sample code shows how to import data from the mydata.json file to the current data table:

  ```
  import -i /tmp/mydata.json
  ```

  The following sample code shows how to import data from the mydata.json file to the current data table with the current time used as the timestamp:

  ```
  import -i /tmp/mydata.json --ignore_version
  ```

4.3. Secondary index

After you create a secondary index for a data table, you can use the index table, view information about the index table, query data by using the index table, and delete the index table.

Prerequisites

- A data table for which the max versions parameter is set to 1 is created.

- Predefined columns are specified for the data table.

Create a secondary index

> **Note** Secondary indexes include global secondary indexes and local secondary indexes. For more information, see Overview.

- Command syntax

```
create_index -t <tableName> -n <indexName> -i <indexType> --pk <primaryKeyName,primaryKeyName> --attr <definedColumn,definedColumn>
```

The following table describes the parameters that you can configure to create a secondary index.

Parameter	Required	Example	Description
-t,--table	No	mytable	The name of the data table.
-n,--name	Yes	index0	The name of the secondary index that you want to create.
-i,--index_type	No	global	The type of the secondary index. Valid values: ○ global: global secondary index. This is the default value. If you use a global secondary index, Tablestore automatically synchronizes the data from the indexed columns and primary key columns of a data table to the columns of an index table in asynchronous mode. The synchronization latency is within a few milliseconds. ○ local: local secondary index. If you use a local secondary index, Tablestore automatically synchronizes the data from the indexed columns and primary key columns of a data table to the columns of an index table in synchronous mode. After data is written to the data table, you can query the data from the index table.

-k,--pk	Yes	uid,pid	The index columns of the index table. The index columns are a combination of primary key columns and predefined columns of the data table. If you use a local secondary index, the first primary key column of the index table must be the same as the first primary key column of the data table for which the index table is created.
-a,--attr	Yes	col0,col1	The attribute columns of the index table. The attribute columns are a combination of predefined columns of the data table.
-b,--without_base_data	No	N/A	Specifies that the secondary index that you want to create does not include the existing data in the data table.

- Examples

  The following code provides an example on how to create a global secondary index named index0 for the data table named mytable. In this example, the secondary index includes existing data of the data table.

```
create_index -t mytable -n index0 -i global --pk pid,uid -a name,col0
```

  The following code provides an example on how to create a global secondary index named index1 for the data table named mytable. In this example, the secondary index does not include the existing data of the data table.

```
create_index -t mytable -n index1 -i global --pk pid,uid -a name,col0 -b
```

  The following code provides an example on how to create a local secondary index named index2 for the data table named mytable. In this example, the secondary index includes the existing data of the data table.

```
create_index -t mytable -n index2 -i local -k uid,name -a col0,col1
```

Use tables

Select the data table on which you want to perform table operations or data operations.

- Command syntax

```
use --wc -t <tableName>
```

  The following table describes the parameters that you must configure to use a data table.

Parameter	Required	Example	Description
--wc	No	N/A	Specifies that the table on which you want to perform operations is a data table or an index table.
-t,--table	Yes	index0	The name of the index table.

- Examples

The following code provides an example on how to use the index table named index0:

```
use -t index0
```

View information about secondary indexes

You can query information about a data table and then save the information to a JSON file on your local PC.

- Command syntax

```
desc -t <tableName> -o /localpath/filename.json
```

The following table describes the parameters that you can configure to query the information about a data table.

Parameter	Required	Example	Description
-t,--table	No	index0	The name of the data table or index table.
-f,--print_format	No	json	The output format of the information about the table. Default value: json. Valid values: json and table.
-o,--output	No	/tmp/describe_table_meta.json	The path of the local JSON file in which the information about the table is stored.

- Examples

The following sample code shows how to query information about the current table.

```
desc
```

The following sample code shows how to query information about the current data table and save the information to the local file named describe_table_meta.json:

```
desc -o  /tmp/describe_table_meta.json
```

Query data by using secondary indexes

Read a single row of data

You can read data from a data table and export the data to a local JSON file.

> **Note** If the row that you want to read does not exist, an empty result is returned.

- Command syntax

```
get --pk '[primaryKeyValue,primaryKeyValue]'
```

The following table describes the parameters you can configure to read a row of data.

Parameter	Required	Example	Description

-p, --pk	Yes	["86",6771]	The values of the primary key columns of the data table. The values are included in an array. **Important** The number and types of the primary key columns that you specify must be the same as the actual number and types of primary key columns in the data table.
--columns	No	name,uid	The set of columns that you want to read. You can specify the name of a primary key column or an attribute column. If you do not specify a column name, all data in the row is returned.
--max_version	No	1	The maximum number of data versions that you want to read.
--time_range_start	No	1626860469000	The version range of data that you want to read. time_range_start specifies the start timestamp and time_range_end specifies the end timestamp. The range includes the start value and excludes the end value.
--time_range_end	No	1626865270000	
--time_range_specific	No	1626862870000	The specific version of data that you want to read.
-o, --output	No	/tmp/querydata.json	The local path of the JSON file to which the query results are exported.

- Example
  Read a row of data in which the value of the first primary key column is "86" and the value of the second primary key column is 6771.

```
get --pk '["86",6771]'
```

Scan data

You can scan a data table to obtain all data or a specified number of rows of data in the data table.

- Command syntax

```
scan --limit limit
```

The following table describes parameters that you can configure to scan data in a data table.

Parameter	Required	Example	Description

--limit	No	10	Optional. The maximum number of rows that you want to return. If you do not configure this parameter, all data in the data table is scanned.

- Example
  The following sample code shows how to scan data in a data table to return up to 10 rows of data:

```
scan --limit 10
```

Export data

You can export data from a data table to a local JSON file.

- Command syntax

```
scan -o /localpath/filename.json -c attributeColumnName,attributeColumnName,attributeColumnName
```

The following table describes the parameters you can configure to export data from a data table to a local file.

Parameter	Required	Example	Description
-c, --columns	Yes	uid,name	The set of columns that you want to export. You can specify the name of a primary key column or an attribute column. If you do not specify a column name, all data in the row is exported.
--max_version	No	1	The maximum number of data versions that can be exported.
--time_range_start	No	1626865596000	The version range of data that you want to export. time_range_start specifies the start timestamp and time_range_end specifies the end timestamp. The range includes the start value and excludes the end value.
--time_range_end	No	1626869196000	
--time_range_specific	No	1626867396000	The specific version of data that you want to export.
--backward	No	N/A	Specifies that the system sorts the exported data in descending order of primary keys.
-o, --output	Yes	/tmp/mydata.json	The local path of the JSON file to which the query results are exported.

-l, --limit	No	10	The maximum number of rows that you want to return in the query.
-b, --begin	No	1000	The value range of data that you want to export.
-e, --end	No	2000	

- Examples
  The following sample code shows how to export all data from the current table to the mydata.json local file:

```
scan -o /tmp/mydata.json
```

  The following sample code shows how to export data from the uid and name columns of the current table to the mydata.json local file:

```
scan -o /tmp/mydata.json -c uid,name
```

Delete secondary indexes

This section describes how to delete index tables that you no longer need.

- Command syntax

```
drop_index -t <tableName> -i <indexName> -y
```

  The following table describes the parameters that you must configure to use a data table.

Parameter	Required	Example	Description
-t,--table	No	mytable	The name of the data table.
-i,--index	Yes	index0	The name of the secondary index.
-y,--yes	Yes	N/A	Specifies that the confirmation information must appear. This parameter is required.

- Examples

  The following code provides an example on how to delete the index table named index0 for the current data table:

```
drop_index -i index0 -y
```

  The following code provides an example on how to delete the index table named index0 for the data table named mytable:

```
drop_index -t mytable -i index0 -y
```

4.4. Search index

After a search index is created for a data table, you can list the names of search indexes, query information about the search index, query data by using the search index, and delete the search index.

Create a search index

You can create a search index.

- Command syntax

```
create_search_index -n search_index_name
```

The following table describes the parameters you can configure to create a search index.

Parameter	Required	Example	Description
-n, --name	Yes	search_index	The name of the search index.
-t, --table	No	mytable	The name of the data table.

- Examples

The following sample code shows how to create a search index named search_index:

```
create_search_index -n search_index
```

The following sample code shows how to enter the index schema as prompted by the system:

```
{
   "IndexSetting": {
       "RoutingFields": null
   },
   "FieldSchemas": [
       {
           "FieldName": "gid",
           "FieldType": "LONG",
           "Index": true,
           "EnableSortAndAgg": true,
           "Store": true,
           "IsArray": false,
           "IsVirtualField": false
       },
       {
           "FieldName": "uid",
           "FieldType": "LONG",
           "Index": true,
           "EnableSortAndAgg": true,
           "Store": true,
           "IsArray": false,
           "IsVirtualField": false
       },
       {
           "FieldName": "col2",
           "FieldType": "LONG",
           "Index": true,
           "EnableSortAndAgg": true,
           "Store": true,
           "IsArray": false,
           "IsVirtualField": false
       },
```

```
        {
            "FieldName": "col3",
            "FieldType": "TEXT",
            "Index": true,
            "Analyzer": "single_word",
            "AnalyzerParameter": {
                "CaseSensitive": true,
                "DelimitWord": null
            },
            "EnableSortAndAgg": false,
            "Store": true,
            "IsArray": false,
            "IsVirtualField": false
        },
        {
            "FieldName": "col1",
            "FieldType": "KEYWORD",
            "Index": true,
            "EnableSortAndAgg": true,
            "Store": true,
            "IsArray": false,
            "IsVirtualField": false
        },
        {
            "FieldName": "col3V",
            "FieldType": "LONG",
            "Index": true,
            "EnableSortAndAgg": true,
            "Store": true,
            "IsArray": false,
            "IsVirtualField": true,
            "SourceFieldNames": [
                "col3"
            ]
        }
    ]
}
```

List search indexes

You can list search indexes that are created for the current data table.

- Command syntax

```
list_search_index
```

The following table describes the parameters that you can configure to list search indexes that are created for the current data table.

Parameter	Required	Example	Description
-a, --all	No	N/A	Specifies that the search indexes of all data tables are displayed.

-d, --detail	No	N/A	Specifies that the details about the search indexes are displayed.

- Examples

  The following sample code shows how to list the search indexes of the current data table:

```
list_search_index
```

  The following result is returned:

```
+-----------+--------------+
| TableName | IndexName    |
+-----------+--------------+
| mytable   | search_index |
+-----------+--------------+
```

Query information about a search index

You can query information about a search index.

- Command syntax

```
describe_search_index -n search_index_name
```

- Example

  The following sample code shows how to query information about the search index named search_index:

```
describe_search_index -n search_index
```

Query data by using a search index

You can query data that meets specified conditions by using a search index and then perform aggregation operations on the returned results.

- Command syntax

```
search -n search_index_name --return_all_indexed
```

- Examples

  The following sample code shows how to query data by using the search index named search_index and return all columns for which indexing is enabled:

```
search -n search_index --return_all_indexed
```

  The following sample code shows how to enter the query conditions as prompted by the system:

```
{
    "Offset": -1,
    "Limit": 10,
    "Collapse": null,
    "Sort": null,
    "GetTotalCount": true,
    "Token": null,
    "Query": {
        "Name": "TermQuery",
        "Query": {
            "FieldName": "uid",
            "Term": 10001
        }
    },
    "Aggregations": [{
        "Name": "avg",
        "Aggregation": {
            "AggName": "agg1",
            "Field": "pid"
        }
    }]
}
```

Delete a search index

You can delete a search index that you no longer need.

- Command syntax

```
drop_search_index -n search_index_name
```

- Example

  The following sample code shows how to delete the search index named search_index:

```
drop_search_index -n search_index
```

4.5. Tunnel Service

Tunnel Service is a centralized service that uses the Table Store API to allow you to consume full and incremental data. Tunnel Service provides tunnels that are used to export and consume data in the full, incremental, and differential modes. After you create a tunnel, you can use it to consume historical and incremental data that is exported from a specified table.

Create a tunnel

Create a tunnel for a data table.

- Command syntax

```
create_tunnel -n name
```

  The following table describes the parameters that you can configure to create a tunnel for a data table.

Parameter	Required	Example	Description
-t, --table	No	mytable	The name of the data table.
-n, --name	Yes	t1	The name of the tunnel.
-m, --mode	No	stream_data_only	The type of the tunnel. Default value: stream_data_only. Valid values: ◦ base_data_only: full tunnel. Only full data is consumed and processed. ◦ stream_data_only: incremental tunnel. Only incremental data is consumed and processed. ◦ base_and_stream: differential tunnel. After full data is consumed and processed, incremental data is consumed and processed.

- Examples

  The following sample code shows how to create a tunnel named t1 for a data table:

```
create_tunnel -n t1
```

  The following result is returned:

```
New tunnel created, its id is '9933470d-8a5e-4972-a5b0-b7ae6f836460'.
```

Query information about a tunnel

Query tunnel information and channel information about a tunnel.

- Command syntax

```
describe_tunnel -n name
```

  The following table describes the parameters that you can configure to query information about a tunnel.

Parameter	Required	Example	Description
-t, --table	No	mytable	The name of the data table.
-n, --name	Yes	t1	The name of the tunnel.
-o, --output	No	D:\\otstest\\mytunnel.txt	The local file in which you want to store the returned result.

- Examples

  The following sample code shows how to query information about the tunnel named t1:

```
describe_tunnel -n t1
```

The following result is returned:

```
Tunnel Info:
+--------------------------------------+------------+------------+---------------+-------
--+
| TunnelId                             | TunnelName | TunnelType | Stage         | Expire
d |
+--------------------------------------+------------+------------+---------------+-------
--+
| 9933470d-8a5e-4972-a5b0-b7ae6f836460 | t1         | Stream     | ProcessStream | false
|
+--------------------------------------+------------+------------+---------------+-------
--+
Channel Info:
+-----------------------------------------------------+-------------+---------------+--
--------+--------------------------------+
| ChannelId                                           | ChannelType | ChannelStatus | C
lientId | ChannelRPO                     |
+-----------------------------------------------------+-------------+---------------+--
--------+--------------------------------+
| cfd2c05b-54b6-48ec-aa6f-feb427f0ca57_1635771329155688 | Stream      | OPEN          |
| 1970-01-01 08:00:00 +0800 CST |
+-----------------------------------------------------+-------------+---------------+--
--------+--------------------------------+
```

Simulate the data consumption of a tunnel

After you create a tunnel, you can simulate the data consumption of the tunnel to preview the data format in the tunnel.

- Command syntax

```
consume_tunnel -n name -m mock_consume
```

The following table describes the parameters that you can configure to simulate the data consumption of a tunnel.

Parameter	Required	Example	Description
-c, --channel	No	cfd2c05b-54b6-48ec-aa6f-feb427f0ca57_1635771329155688	The ID of the channel. If you do not configure this parameter, the data in all channels of the tunnel is consumed.
-t, --table	No	mytable	The name of the data table.
-n, --name	Yes	t1	The name of the tunnel.

-m, --mode	Yes	mock_consume	The consumption mode. Default value: shadow_copy. Valid values: ○ shadow_copy: copies the consumption traffic of an online tunnel. ○ mock_consume: simulates data consumption but does not update checkpoint information. ○ real_consume: consumes data and updates checkpoint information. We recommend that you do not use this mode.

- Examples

  The following sample code shows how to simulate the data consumption of the tunnel named t1:

```
consume_tunnel -n t1 -m mock_consume
```

  After you run the command to simulate data consumption, data consumption records are displayed when data is written to the data table. The following sample code shows an example of the returned result:

```
Starting consume tunnel 't1' of table 'mytable', it may take a few seconds to start, plea
se wait...
null
```

Delete a tunnel

You can delete a tunnel that you no longer need.

- Command syntax

```
drop_tunnel -n name -y
```

  The following table describes the parameters that you can configure to delete a tunnel.

Parameter	Required	Example	Description
-t, --table	No	mytable	The name of the data table.
-n, --name	Yes	t1	The name of the tunnel.
-y, --yes	Yes	N/A	Specifies that confirmation information appears. This configuration item must be included in the command.

- Example

  The following sample code shows how to delete the tunnel named t1:

```
drop_tunnel -n t1 -y
```

4.6. SQL query

This topic describes how to create a mapping table for a data table, query all mapping tables of a data table in an instance, query information about a mapping table, and query data in a table in the SQL mode.

Enter the SQL mode

Run the `sql` command to enter the SQL mode.

Create a mapping table

To query data in a data table, you must create a mapping table for the data table.

The following sample code shows how to create a mapping table for the data table named mytable:

```
CREATE TABLE mytable(
     `uid` VARCHAR(1024),
     `pid` BIGINT(20),
     `b` DOUBLE,
     `c` BOOL,
     `d` MEDIUMTEXT,
     PRIMARY KEY(`uid`,`pid`)
);
```

Query mapping tables

Query the mapping tables of a table in an instance.

Run the `SHOW TABLES;` command to query the mapping tables of a table in an instance.

The following result is returned:

```
+----------------------+
| Tables_in_myinstance |
+----------------------+
| mytable              |
+----------------------+
| mytstable            |
+----------------------+
| mytstable::meta      |
+----------------------+
```

In the preceding command output, mytable is the data table for which mapping tables are created, mytstable is a time series data table, and mytstable::meta is a time series metadata table.

Query information about a mapping table

Query information about a mapping table.

- Command syntax

  ```
  DESCRIBE table_name;
  ```

- Examples

The following sample code shows how to query the information about the mapping table named mytable:

```
DESCRIBE mytable;
```

The following result is returned:

```
+-------+---------------+------+-----+-------+
| Field | Type          | Null | Key | Extra |
+-------+---------------+------+-----+-------+
| uid   | varchar(1024) | NO   | PRI |       |
+-------+---------------+------+-----+-------+
| pid   | bigint(20)    | NO   | PRI |       |
+-------+---------------+------+-----+-------+
| b     | double        | YES  |     |       |
+-------+---------------+------+-----+-------+
| c     | tinyint(1)    | YES  |     |       |
+-------+---------------+------+-----+-------+
| d     | mediumtext    | YES  |     |       |
+-------+---------------+------+-----+-------+
```

Delete a mapping table

If changes are made to the attribute columns of a data table, you can delete the mapping table of the data table and create a new mapping table.

- Command syntax

  ```
  DROP MAPPING TABLE table_name;
  ```

- Example

  The following sample code shows how to delete the mapping table named mytable:

  ```
  DROP MAPPING TABLE mytable;
  ```

Query data in a table

Query data in a table by executing the SELECT statement.

The following sample code shows how to query all data in the table named mytable:

```
SELECT * FROM mytable;
```

Exit the SQL mode

Run the `exit;` command to exit the SQL mode.

5.TimeSeries model

5.1. Operations on a time series table

After you create a time series table, you can use the table, query the information about the table, update the table, and delete the table. You can also list the names of all tables in an instance.

> **Note** For more information about the TimeSeries model, see Overview.

Create a time series table

Create a time series table for which the time to live (TTL) value is specified.

- Command syntax

```
create -m mode -t tableName --ttl timeToLive
```

The following table describes the parameters that you can configure to create a time series table.

Parameter	Required	Example	Description
-m, --model	Yes	timeseries	The type of the table that you want to create. Default value: widecolumn. Valid values: ○ widecolumn: data table. ○ timeseries: time series table.
-t, --table	Yes	mytable	The name of the time series table.
--ttl	No	864000	The TTL of the data in the time series table. Unit: seconds. The default value is -1, which indicates that the data never expires. If the system detects that the difference between the current time and the time column that is passed to the table exceeds the specified TTL value, the system automatically deletes the expired data. **Important** In a time series table, the system determines the time when the data is generated based on the time column that is passed to the table, not the time when the data is written to the table. Minimum value: 86400 seconds, which is one day. A value of -1 specifies that the data never expires.

- Example

  Create a time series table named mytable in which the data never expires.

```
create -m timeseries -t mytable --ttl -1
```

Use a time series table

Select the data table on which you want to perform table operations or data operations.

- Command syntax

```
use --ts -t tableName
```

The following table describes the parameters that you must configure to use a time series table.

Parameter	Required	Example	Description
--ts	Yes	N/A	Specifies that the table you want to use is a time series table.
-t, --table	Yes	mytable	The name of the time series table.

- Example

Use a time series table named mytable.

```
use --ts -t mytable
```

List the names of tables

List the names of all tables, all data tables, or all time series tables in an instance.

- List the names of all tables of the same type as the current table

```
list
```

- List the names of all tables

```
list -a
```

- List the names of all data tables

```
list -w
```

- List the names of all time series tables

```
list -t
```

The following table describes the parameters that you can configure to list the names of tables.

Parameter	Required	Example	Description
-a, --all	No	N/A	Lists the names of all tables.
-d, --detail	No	N/A	Lists the details about the tables.
-w, --wc	No	N/A	Lists the names of all data tables.
-t, --ts	No	N/A	Lists the names of all time series tables.

Update a time series table

Modify the TTL configuration of a time series table.

- Command syntax

```
alter --ttl timeToLive --ts
```

The following table describes the parameters that you must configure to modify the TTL configuration of a time series table.

Parameter	Required	Example	Description
--ts	Yes	N/A	Specifies that the table whose TTL configuration you want to modify is a time series table.
--ttl	Yes	864000	The TTL of the data in the time series table. Unit: seconds. The default value is -1, which indicates that the data never expires. If the system detects that the difference between the current time and the time column that is passed to the table exceeds the specified TTL value, the system automatically deletes the expired data. **Important** In a time series table, the system determines the time when the data is generated based on the time column that is passed to the table, not the time when the data is written to the table. Minimum value: 86400 seconds, which is one day. A value of -1 specifies that the data never expires.

- Example

  Set the TTL value of the current table to 86400 seconds.

```
alter --ttl 86400 --ts
```

Query the information about a time series table

Query the information about a time series table.

- Command syntax

```
desc --ts -t tableName
```

The following table describes the parameters that you must configure to query the information about a time series table.

Parameter	Required	Example	Description
--ts	No	N/A	Specifies that the table whose information you want to query is a time series table.

-t, --table	No	mytable	The name of the time series table. This parameter is optional.
-f, --print_format	No	json	The output format of the information about the data table. Default value: json. Valid values: json and table.
-o, --output	No	/tmp/describe_table_meta.json	The path of the local JSON file in which the information about the time series table is stored.

- Example

  Query the information about the current table.

  ```
  desc
  ```

  Query the information about a time series table named mytable.

  ```
  desc --ts -t mytable
  ```

Delete a time series table

Delete a time series table that you no longer need.

- Command syntax

  ```
  drop -t tableName --ts -y
  ```

  The following table describes the parameters that you must configure to delete a time series table.

Parameter	Required	Example	Description
-t, --table	Yes	mytable	The name of the time series table.
-y, --yes	Yes	N/A	Specifies that the confirmation information must appear. This parameter is required.
--ts	Yes	N/A	Specifies that the table you want to delete is a time series table.

- Example

  Delete a time series table named mytable.

  ```
  drop -t mytable --ts -y
  ```

5.2. Operations on time series data

The Tablestore CLI allows you to write and import time series data to time series tables and query time series data in time series tables. You can also use the Tablestore CLI to retrieve, scan, and update the time series in time series tables.

Write time series data

Write time series data to a time series table.

- Command syntax

```
putts --k '["measurement_name","data_source",["tagKey1=tagValue1","tagKey2=tagValue2"]]'
--field '[{"c":"fieldname","v":"fieldvalue"},{"c":"bool_field","v":true},{"c":"double_fie
ld","v":1.1},{"c":"int_value","v":10,"isint":true}]' --time 1635162859000000
```

The following table describes the parameters that you can configure to write time series data to a time series table.

Parameter	Required	Example	Description
--k	Yes	'["cpu","localhost", ["region=hangzhou","os=ubuntu"]]'	The time series identifier, which is an array that contains the following content: ◦ Measurement name: the metric name. In this example, the metric name is cpu. ◦ Data source: the data source from which data is generated. In this example, the data source is localhost. ◦ Tags: the tags. The value is an array in which each tag consists of tagKey and tagValue in the format tagKey=tagValue.

--field	Yes	'[{"c":"fieldname","v":"field value"}, {"c":"bool_field","v":true}, {"c":"double_field","v":1.1}, {"c":"int_value","v":10,"isint":true}]'	The data column of time series data, which is in the JSON format and contains the following content: ○ c: This parameter is required and specifies the name of the data column. ○ v: This parameter is required and specifies the value of the data column. The value of this parameter can be of the string, numeric, or Bool type. ○ isint: This parameter is optional and specifies whether the value of the numeric data column is converted to a value of the integer type. The value of this parameter is of the Bool type. Valid values: true and false. The default value is false, which specifies that the value of the numeric data column is converted to a value of the floating-point type. If you set this parameter to true, the value of the numeric data column is converted to a value of the integer type.
--time	No	1635162859000000	The time when the row of time series data is generated. The value of this parameter is a timestamp. Unit: microseconds (μs).

- Example

  Write a row of time series data.

```
putts --k '["cpu","localhost",["region=hangzhou","os=ubuntu"]]' --field '[{"c":"fieldname","v":"fieldvalue"},{"c":"bool_field","v":true},{"c":"double_field","v":1.1},{"c":"int_value","v":10,"isint":true}]' --time 1635162859000000
```

Import time series data

Import time series data from a local file to a time series table.

- Command syntax
  - Windows

```
import_timeseries --input D:\\localpath\\filename.txt
```

  - Linux and macOS

```
import_timeseries --input /localpath/filename.txt
```

The following table describes the parameters that you must configure to import time series data from a local file to a time series table.

Parameter	Required	Example	Description
-i, --input	Yes	/temp/import_timeseries.txt	The configuration file from which you want to import time series data.

The following example shows the content of a configuration file:

```
cpu,hostname=host_0,region=cn-hangzhou usage_user=58i,usage_system=2i,usage_idle=24i 1609459200000000000
cpu,hostname=host_1,region=cn-hangzhou usage_user=58i,usage_system=2i,usage_idle=24i 1609459200000000000
```

Each row in the example is a row of time series data in the format `measurement_name,tags fields timestamp`. The following table describes the parameters.

Parameter	Required	Example	Description
measurement	Yes	cpu	The metric type.
tags	Yes	hostname=host_0,region=cn-hangzhou	The tags. Each tag consists of tagKey and tagValue in the format tagKey=tagValue. The tagValue of the first tag specifies the data source from which the data is generated. In this example, host_0 specifies the data source of the time series data in the row.
fields	Yes	usage_user=58i,usage_system=2i,usage_idle=24i	The data column of the time series data. Each data column consists of columnKey and columnValue in the format columnKey=columnValue. The value of columnValue can be of the integer or floating-point type. If you add i to the end of the value, the data type of the value is integer. Otherwise, the data type of the value is floating-point.

timestamp	Yes	1609459200000000000	The time when the row of time series data is generated. The value of this parameter is a timestamp. Unit: nanoseconds (ns). **Important** After data is imported to the time series table, the timestamp in nanoseconds in the configuration file is automatically converted into a timestamp in microseconds.

- Example

  Import the time series data from the import_timeseries.txt file to a time series table.

```
import_timeseries --input /temp/import_timeseries.txt
```

Query time series data

Query the time series data in a specific time range.

- Command syntax

```
getts --k '["measurement_name","data_source",["tagKey1=tagValue1","tagKey2=tagValue2"]]'
--time_start 0 --time_end 1635162900000000 --limit 100
```

The following table describes the parameters that you can configure to query time series data.

Parameter	Required	Example	Description
--k	Yes	'["cpu","localhost",["region=hangzhou","os=ubuntu"]]'	The time series identifier, which is an array that contains the following content: ○ Measurement name: the metric name. In this example, the metric name is cpu. ○ Data source: the data source from which data is generated. In this example, the data source is localhost. ○ Tags: the tags. The value is an array in which each tag consists of tagKey and tagValue in the format tagKey=tagValue.
--time_start	Yes	0	The beginning of the time range to query.

--time_end	Yes	1667638230000000	The end of the time range to query.
--limit	No	100	The maximum number of entries to return. Valid values: 1 to 5000. The actual number of returned entries is determined by the server.

- Example

  Query all time series data that is generated before 1667638230000000 in the time series whose metric name is cpu, data source is localhost, and tags are "region=hangzhou" and "os=ubuntu".

```
getts --k '["cpu","localhost",["region=hangzhou","os=ubuntu"]]' --time_start 0 --time_end
1667638230000000 --limit 100
```

Retrieve time series

Retrieve the time series that meet the specified conditions.

- Command syntax

  Note You can abbreviate query_ts_meta in the command as qtm.

```
query_ts_meta --measurement measurement_name --datasource data_source --limit 10
```

The following table describes the parameters that you can configure to retrieve time series.

Parameter	Required	Example	Description
--measurement	No	cpu	The metric name.
--datasource	No	localhost	The data source.
--limit	No	100	The maximum number of entries to return. Valid values: 1 to 1000. The actual number of returned entries is determined by the server.
-e, --edit	No	{"measurement":"cpu","data_source":"localhost"}	The query condition in the JSON format.

You can also run the `qtm -e` command and specify the query conditions to perform a complex query. The following example shows the query conditions:

```
{
    "Type":"COMPOSITE",
    "QueryCondition":{
        "Operator":"AND",
        "SubConditions":[
            {
                "Type":"MEASUREMENT",
                "QueryCondition":{
                    "Operator":"EQUAL",
                    "Value":"CPU"
                }
            },
            {
                "Type":"SOURCE",
                "QueryCondition":{
                    "Operator":"EQUAL",
                    "Value":"127.0.0.1"
                }
            },
            {
                "Type":"TAG",
                "QueryCondition":{
                    "Operator":"GREATER_EQUAL",
                    "TagName":"Region",
                    "Value":"Jiangning"
                }
            }
        ]
    }
}
```

- Example

  Retrieve the time series whose metric name is cpu and data source is localhost.

```
query_ts_meta --measurement cpu --datasource localhost --limit 10
```

Scan time series

Obtain all time series or a specified number of time series in a time series table.

- Command syntax

> **Note** You can abbreviate query_ts_meta in the command as qtm.

```
query_ts_meta --limit limit
```

The following table describes the parameter that you can configure to scan time series.

Parameter	Required	Example	Description

--limit	No	10	The maximum number of rows to return for this scan. If you do not configure this parameter, all data in the table is scanned.

- Example

```
query_ts_meta --limit 10
```

Update a time series

Modify the properties of a time series.

- Command syntax

> **Note** You can abbreviate update_ts_meta in the command as utm.

```
update_ts_meta --k '["measurement_name","data_source",["tag1=value1","tag2=value2"]]' --a
ttrs '["key1=value1","key2=value2"]'
```

The following table describes the parameters that you can configure to modify the properties of a time series.

Parameter	Required	Example	Description
--k	Yes	'["cpu","localhost", ["region=hangzhou","os=ubuntu"]]'	The time series identifier, which is an array that contains the following content: ◦ Measurement name: the metric name. In this example, the metric name is cpu. ◦ Data source: the data source from which data is generated. In this example, the data source is localhost. ◦ Tags: the tags. The value is an array in which each tag consists of tagKey and tagValue in the format tagKey=tagValue.
--attrs	No	'["city=nanjing","region=jiangning"]'	The properties of the time series. The value is an array in which each property consists of a key and a value in the format key=value.

- Example

  Modify the properties of the specified time series to "city=nanjing" and "region=jiangning".

```
update_ts_meta --k '["cpu","localhost",["city=hangzhou","region=xihu"]]' --attrs '["city=
nanjing","region=jiangning"]'
```

5.3. SQL query

This topic describes how to query all mapping tables, query the information about a mapping table, and query the data in a table by executing SQL statements in SQL mode.

Enter the SQL mode

Run the `sql` command to enter the SQL mode.

Query mapping tables

Query the mapping tables of a table in an instance.

Run the `SHOW TABLES;` command to query the mapping tables of a table in an instance.

The following result is returned:

```
+----------------------+
| Tables_in_myinstance |
+----------------------+
| mytable              |
+----------------------+
| mytstable            |
+----------------------+
| mytstable::meta      |
+----------------------+
```

In the preceding command output, mytable is the data table for which mapping tables are created, mytstable is a time series data table, and mytstable::meta is a time series metadata table.

Query information about a mapping table

Query information about a mapping table.

- Command syntax

  ```
  DESCRIBE table_name;
  ```

- Examples

  The following sample code shows how to query the information about the mapping table named mytable:

  ```
  DESCRIBE mytable;
  ```

  The following result is returned:

```
+-------+---------------+------+-----+-------+
| Field | Type          | Null | Key | Extra |
+-------+---------------+------+-----+-------+
| uid   | varchar(1024) | NO   | PRI |       |
+-------+---------------+------+-----+-------+
| pid   | bigint(20)    | NO   | PRI |       |
+-------+---------------+------+-----+-------+
| b     | double        | YES  |     |       |
+-------+---------------+------+-----+-------+
| c     | tinyint(1)    | YES  |     |       |
+-------+---------------+------+-----+-------+
| d     | mediumtext    | YES  |     |       |
+-------+---------------+------+-----+-------+
```

Query the data in a table

Query the data in a table by executing the SELECT statement.

Run the following command to query all time series data in a time series table named mytstable:

```
SELECT * FROM mytstable limit 10;
```

Exit the SQL mode

Run the `exit;` command to exit the SQL mode.

6.View options

This topic describes how to use the help option to view all supported options.

- Command syntax

  ```
  help
  ```

  You can also view all options of a single command by using `command keyword help`. For example, you can run the `alter help` command to view all options of the alter command.

- Returned result

```
Commands:
  alter                     Alter table
  clear                     Clear the screen
  config                    Setup the configuration to access Tablestore
  consume_tunnel            Consume tunnel and print stream data
  create                    Create a new table
  create_index              Create a new secondary index
  create_instance           Create a new instance
  create_search_index       Create a new search index
  create_tunnel             Create tunnel.
  del                       Delete the specific row of table
  desc                      Describe table
  describe_instance         Describe the instance, to get the detail info
  describe_search_index     Describe the search index to get detail info
  describe_tunnel           Get the tunnel's detail info
  drop                      Drop the table
  drop_index                Drop the secondary index
  drop_search_index         Drop search index
  drop_tunnel               Drop the tunnel
  enable_service            Enable ots service before you start to use Tablestore, it is
totally free
  exit                      Exit the program
  export                    Export the whole data of table to file
  get                       Get row by primary key from table
  get_splits                Logically divide the data of the full table into several spl
its close to the specified size
  getts                     Get data from timeseries table
  help                      Display help
  import                    Load the data into table, only WideColumn table is supported

  import_timeseries         Load the data into time series table, only TimeSeries table
is supported
  list                  List all tables
  list_instance             List all the instances
  list_search_index         List all the search indexes of the table
  list_tunnel               List all the tunnels of the table
  press_check               Check data for press
  press_input               Input data for press
  put                       Insert a row into table
  putts                     put data to a timeseries table
  query_ts_meta             Query timeseries meta information
  quit                      Quit the program
  scan                      Scan table
  search                    Execute search query on search index
  sql                       Run in SQL mode, then you can use SQL to select data
  update                    Update the row in table, it will insert a new row if it is n
ot exist
  update_ts_meta            Update attributes for timeseries
  use                       Choose table to use
```