ALIBABA CLOUD

Alibaba Cloud

Apsara Stack Enterprise

Tablestore

User Guide

Product Version: v3.16.2

Document Version: 20220905

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}`

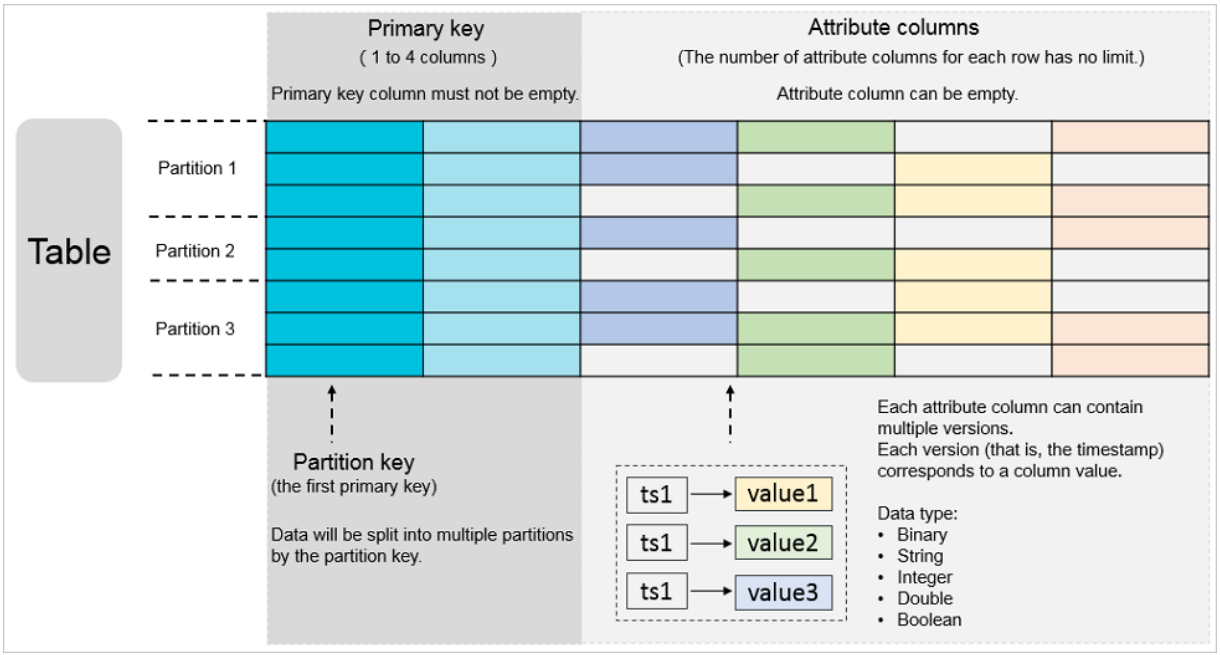
Table of Contents

1.Terms

This topic describes several basic terms used in Tablestore, including data model, max versions, time to live (TTL), max version offset, primary key and attribute, read/write throughput, region, instance, endpoint, and Serial ATA (SATA).

data model

A data model that consists of tables, rows, primary keys, and attribute columns in Tablestore. The following figure shows an example of a data model.



max versions

A data table attribute that indicates the maximum number of data versions that can be stored in each attribute column of a data table. If the number of versions in an attribute column exceeds the max versions value, the earliest version is asynchronously deleted.

TTL

A data table attribute that indicates the validity period of data in seconds. To save space and reduce costs for data storage, Tablestore deletes any data that exceeds its TTL.

max version offset

A data table attribute that describes the maximum allowable difference between the version to be written and the current time in seconds.

To prevent the writing of unexpected data, a server checks the versions of attribute columns when the server processes writing requests. If the specified version is earlier than the current writing time minus the max version offset value or later than or equal to the current writing time plus the max version offset value, data fails to be written to the row.

The valid version range of an attribute column: **[max{Data written time - Max version offset, Data written time - TTL value}, Data written time + Max version offset)**. Data written time is the number of seconds that have elapsed since 00:00:00, 1 January 1970. Versions of the attribute columns are written in milliseconds. A version of an attribute column must fall within the valid version range after the version number is converted to seconds (divide by 1,000).

primary key and attribute

A primary key is the unique identifier of each row in a table. A primary key consists of one to four primary key columns. When you create a table, you must define a primary key. You must specify the name, data type, and sequence of each primary key column. The data type of primary key columns can be only STRING, INTEGER, or BINARY. The size of a STRING or BINARY primary key column cannot exceed 1 KB.

An attribute is the attribute data stored in a row. You can create an unlimited number of attribute columns for each row.

read/write throughput

A Tablestore attribute that is measured by read/write capacity units (CUs).

region

An Apsara Stack physical data center. Tablestore is deployed across multiple Apsara Stack regions. Select a region that suits your business requirements.

instance

A logical entity that is used to manage tables in Tablestore. Instances correspond to databases in traditional relational databases. An instance is the basic unit of the Tablestore resource management system. Tablestore allows you to control access and meter resources by instance.

endpoint

The connection URL for each instance. You must specify an endpoint before you perform any operations on Tablestore tables and data.

SATA

A disk that is based on serial connections and provides stronger error-correcting capabilities. Serial ATA aims to improve the reliability of data during transmission.

2.Limits

This topic describes the usage limits of Tablestore.

The following table describes the limits on the usage of Tablestore. A part of limits indicate the maximum values that can be used rather than the suggested values. You can tailor table schemas and row sizes to improve performance.

Item	Limit	Description
The number of instances created in an Apsara Stack tenant account	1024	If you need to increase the maximum number of instances, contact an administrator.
The number of tables in an instance	1024	If you need to increase the maximum number of tables, contact an administrator.
The length of an instance name	3 to 16 bytes	The instance name can contain uppercase and lowercase letters, digits, and hyphens (-). It must start with a letter and cannot end with a hyphen (-).
The length of a table name	1 to 255 bytes	The table name can contain uppercase and lowercase letters, digits, and underscores (_). It must start with a letter or underscore (_).
The length of a column name	1 to 255 bytes	The column name can contain uppercase and lowercase letters, digits, and underscores (_). It must start with a letter or underscore (_).
The number of columns in a primary key	1 to 4	A primary key can contain one to four columns.
The size of the value in a STRING primary key column	1 KB	The size of the value in a STRING primary key column cannot exceed 1 KB.
The size of the value in a STRING attribute column	2 MB	The size of the value in a STRING attribute column cannot exceed 2 MB.
The size of the value in a BINARY primary key column	1 KB	The size of the value in a BINARY primary key column cannot exceed 1 KB.
The size of the value in a BINARY attribute column	2 MB	The size of the value in a BINARY attribute column cannot exceed 2 MB.
The number of attribute columns in a single row	Unlimited	A single row can contain an unlimited number of attribute columns.
The number of attribute columns written by one request	1,024	During a PutRow, UpdateRow, or BatchWriteRow operation, the number of attribute columns written to a single row cannot exceed 1,024.
The data size of a row	Unlimited	The total size of all column names and column values for a row is unlimited.

Item	Limit	Description
The number of columns that are specified by the columns_to_get parameter in a read request	0 to 128	The maximum number of columns obtained from a single row of data in a read request cannot exceed 128.
The number of UpdateTable operations for a table	Upper limit: unlimited Lower limit: unlimited	The frequency of UpdateTable operations for a table is limited.
The frequency of UpdateTable operations for a table	Once every two minutes	The reserved read/write throughput for a table can be adjusted once every two minutes at most.
The number of rows read by one BatchGetRow request	100	None.
The number of rows written by one BatchWriteRow request	200	None.
The size of data written by one BatchWriteRow request	4 MB	None.
Data returned by one GetRange request	5,000 rows or 4 MB	The amount of data returned by a request cannot exceed 5,000 rows or 4 MB. When either of the limits is exceeded, data that exceeds the limits is truncated at the row-level. The data primary key information in the next row is returned.
The data size of an HTTP request body	5 MB	None.

3.Quick start

3.1. Log on to the Tablestore console

This topic shows how to log on to the Tablestore console.

Prerequisites

- Before you log on to the Apsara Uni-manager Management Console, the endpoint of the console is obtained from the deployment staff.
- We recommend that you use Google Chrome.

Procedure

1. Enter the URL of the Apsara Uni-manager Management Console in the address bar and press the Enter key.
2. Enter your username and password.

   Obtain the username and password that you use to log on to the Apsara Uni-manager Management Console from the operations administrator.

   > **Note** The first time that you log on to the Apsara Uni-manager Management Console, you must change the password of your account. For security purposes, your password must meet the minimum complexity requirements. The password must be 10 to 32 characters in length and must contain at least two of the following character types:
   >
   > - Uppercase or lowercase letters
   > - Digits
   > - Special characters, including exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%)

3. Click **Log On**.
4. If your account has multi-factor authentication (MFA) enabled, perform corresponding operations in the following scenarios:
   - You log on to the Apsara Uni-manager Management Console for the first time after MFA is enabled by the administrator:
     a. On the Bind Virtual MFA Device page, bind an MFA device.
     b. Enter the username and password again as in Step 2 and click **Log On**.
     c. Enter a six-digit MFA verification code and click **Authenticate**.
   - You have enabled MFA and bound an MFA device:

     Enter a six-digit MFA verification code and click **Authenticate**.

     > **Note** For more information, see the *Bind a virtual MFA device to enable MFA* topic in *Apsara Uni-manager Management Console User Guide*.

5. In the navigation pane, choose **Products > Storage > Tablestore**.

3.2. Create instances

An instance is a logical entity in Tablestore and is used to manage tables. An instance is the basic unit of resource management in Tablestore. Tablestore controls access requests from applications and collects statistics about the resources that are used by applications at the instance level. This topic describes how to create an instance.

Procedure

1. Log on to the Tablestore console.
2. On the **Overview** tab, click **Create Instance**.

   > **Note** You can create different instances to manage the associated tables for different businesses, or create different instances for development, testing, and production environments of the same business. By default, Tablestore allows you to create up to 1,024 instances and up to 1,024 tables in each instance by using an Alibaba Cloud account.

3. On the **Create Instance** page, configure the parameters. The following table describes the parameters.

Parameter	Description
Organization	Select an organization.
Resource Set	Select a resource set.
Region	Select the region in which the instance resides.
Instance Name	Enter a name for the instance. Instance naming conventions: The name must be 3 to 16 characters in length and can contain only letters, digits, and hyphens (-). The name must start with a letter and cannot start with case-insensitive string `ali` or `ots`.
Description	Enter a description for the instance.
Instance Type	Select an instance type. Tablestore provides high-performance instances and capacity instances. The instance type varies based on the type of the cluster that you deploy.

4. Click **Submit**.
5. In the message that appears, click **Back to Console**.

   On the **Overview** tab, you can view the created instance.

   You can perform the following operations on the instance:

   - Click the name of the instance or click **Manage Instance** in the Actions column of the instance. On the **Instance Management** page, you can click different tabs to perform different operations.
     - On the **Instance Details** tab, you can view access URLs of the instance and basic information about the instance, create data tables, and perform operations on and manage data tables.

- On the **Instance Monitoring** tab, you can view the monitoring metrics of different categories or operation types in the specified time range. Categories include service monitoring overview, queries per second (QPS), number of rows, and traffic statistics.
- On the **Network Management** tab, you can bind or unbind virtual private clouds (VPCs) and view the list of VPCs.

- Click **Release** in the **Actions** column to release the instance.

> **Notice**
>
> - Before you release an instance, make sure that all data tables in the instance are deleted, and VPCs are unbound from the instance.
> - To prevent conflicts, make sure that the name of the instance that you want to create is different from the name of the instance that is being released.

3.3. Create data tables

This topic describes how to create a data table in the Tablestore console.

Procedure

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the required instance or click **Manage Instance** in the Actions column corresponding to the instance.
3. In the **Tables** section of the **Instance Details** tab, click **Create Table**.

   > **Note** You can create up to 1,024 data tables in each instance.

4. In the **Create Table** dialog box, set **Table Name** and **Primary Key**.

   The following table describes the parameters that you can configure.

Parameter	Description
Table Name	The name of the data table. This name is used to identify a data table in an instance. The name must be 1 to 255 bytes in length and can contain letters, digits, and underscores (_). The name must start with a letter or an underscore (_).

Parameter	Description
Primary Key	One or more primary key columns in the data table that uniquely identify each row in the table. Enter a name for the primary key column and select a data type. Click **Add Primary Key Column** to add a primary key column. 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. **Note** ◦ In Tablestore, only a primary key column can be used as an auto-increment primary key column. Partition keys cannot be used as auto-increment primary key columns. ◦ After a primary key column is set to an auto-increment primary key column, Tablestore automatically generates a value for the auto-increment primary key column when you write a row of data. You do not need to specify a value for the auto-increment primary key column. The values of auto-increment primary key columns are incremental and unique within the rows that share the same partition key. ◦ Naming conventions for primary key columns: The name must be 1 to 255 bytes in length and can contain letters, digits, and underscores (_). The name must start with a letter or an underscore (_). ◦ The **STRING**, **INTEGER**, **BINARY**, and **Auto Increment** data types are supported by primary key columns. You can select **Auto Increment** as the data type only for a primary key column that is not a partition key.

5. (Optional) Configure advanced parameters.

   If you need to configure parameters such as Time to Live and Max Versions, perform the following operations:

   i. Turn on **Advanced Settings**.

ii. Configure advanced parameters.

The following table describes the advanced parameters that you can configure.

Parameter	Description
Time to Live	The period for which data in the table can be retained. When the retention period exceeds the Time to Live (TTL) value, the system deletes the expired data. The minimum TTL value is 86,400 seconds (one day). The value of -1 indicates that data never expires.
Max Versions	The maximum number of versions of data that can be retained for an attribute column. When the versions of data in an attribute column exceed the Max Versions value, the system deletes the earliest versions of data to keep the maximum number of versions equal to the Max Versions value. Valid values: 1 to 10.
Max Version Offset	The difference between the version number and the data written time must be within the value of Max Version Offset. Otherwise, an error occurs when the data is written. Unit: seconds. The valid version range of data in an attribute column is calculated based on the following formula: Valid version range = **[Data written time - Max version offset, Data written time + Max version offset)**.
Reserved Read Throughput	You can configure this parameter only for high-performance instances. The read or write throughput that is allocated and reserved for the data table. Valid values: integers from 0 to 5000. When Reserved Read Throughput or Reserved Write Throughput is set to **0**, Tablestore does not reserve related resources for the data table.
Reserved Write Throughput	

6. (Optional) Create a secondary index.

If you need to create a secondary index, perform the following operations:

i. Turn on **Global Secondary Index**.

ii. Click the **+ Add** button in the Pre-defined Column section. Enter the name of the pre-defined column and select a data type from the drop-down list.

- This operation is performed to create a predefined column for the data table. Tablestore uses a schema-free model. You can write any columns to a row and do not need to specify the schema. When you create a data table, you can also predefine columns and specify their data types.
- You can add up to 20 predefined columns. To delete a predefined column that you add, click the  icon on the left of the predefined column.
- The name of a predefined column must be 1 to 255 bytes in length and can contain letters, digits, and underscores (_). The name must start with a letter or an underscore (_).
- The data types of predefined columns include STRING, INTEGER, BINARY, FLOAT, and BOOLEAN.

iii. Click **Add Global Secondary Index**. Enter Index Name and specify Primary Key and Pre-defined Column for the index.

- The name of a secondary index must be 1 to 255 bytes in length and can contain letters, digits, and underscores (_). The name must start with a letter or an underscore (_).
- The primary key columns of the secondary index consist of all primary key columns and any number of predefined columns of the data table.
- Pre-defined Column is optional. You can set the predefined columns of the secondary index to only the predefined columns of the data table for which the secondary index is created.

7. Click **OK**.

After a data table is created, you can view the data table in the **Tables** section. If the data table that you created is not displayed in the list of data tables, click the  icon to refresh the list of data tables.

After a data table is created, you can perform the following operations on the table:

- Click the name of the table or click an action in the Actions column of the table, click a tab on the **Manage Table** page that appears, and then perform specific operations.
  - On the **Basic Information** tab, you can view the basic information, advanced features, primary key columns, predefined columns, and Stream information about the data table.
  - On the **Query Data** tab, you can insert or update data, query data, view data details, and delete multiple rows of data at a time.
  - On the **Indexes** tab, you can create secondary indexes, view details of the indexes, query data, and delete indexes.
  - On the **Tunnels** tab, you can enable the Stream feature, create a tunnel, show the list of channels, and delete a tunnel.
  - On the **Monitoring Indicators** tab, you can view the monitoring metrics of different categories or operation types for a table or an index within a specified time range. Categories include service monitoring overview, average access latency, queries per second (QPS), number of rows, and traffic statistics.

- Click the icon in the Actions column of the table and choose **Delete** from the shortcut menu.

  Click OK in the Delete Table dialog box.

  > **Notice** If you delete a data table, the table and the data in the table are permanently deleted from Tablestore and cannot be restored. Exercise caution when you perform this operation.

3.4. Read and write data in the console

After a data table is created, you can use the Tablestore console to perform read and write operations on the data in the data table.

Write data

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the required instance or click **Manage Instance** in the Actions column corresponding to the instance.
3. In the **Tables** section of the **Instance Details** tab, click the name of the required data table, and click the **Query Data** tab. You can also click **Query** in the Actions column of the data table.
4. On the **Query Data** tab, click **Insert**.
5. In the **Insert** dialog box, set Primary Key Value. Click **Add Column**. Set **Name**, **Type**, **Value**, and **Version**.

   By default, **System Time** is selected, which indicates that the current system time is used as the version number of the data. You can also clear **System Time** and enter the version number of the data.
6. Click **OK**.

   Rows that are written to the table are displayed on the **Query Data** tab.

Update data

You can update data in the attribute columns of a row.

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the required instance or click **Manage Instance** in the Actions column corresponding to the instance.
3. In the **Tables** section of the **Instance Details** tab, click the name of the required data table, and click the **Query Data** tab. You can also click **Query** in the Actions column of the data table.
4. On the **Query Data** tab, select the row of data that you want to update. Click **Update**.
5. In the **Update** dialog box, modify the type and value of a primary key column, add or remove attribute columns, or update or delete data from attribute columns.

   - You can click **Add Column** to add an attribute column. You can also click the icon to delete an attribute column.

- In the first Actions column, if you select **Update**, you can modify data in attribute columns. If you select **Delete**, you can delete data of the selected version. If you select **Delete All**, you can delete all versions of the data.

6. Click **OK**.

Read data

In the Tablestore console, you can query data in a single row (GetRow) or data within a specified range (RangeQuery).

To query data in a single row, perform the following operations:

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the required instance or click **Manage Instance** in the Actions column corresponding to the instance.
3. In the **Tables** section of the **Instance Details** tab, click the name of the required data table, and click the **Query Data** tab. You can also click **Query** in the Actions column of the data table.
4. On the **Query Data** tab, click **Search**.
5. Specify query conditions.
   i. In the **Search** dialog box, set Modes to **GetRow**.
   ii. By default, the system returns all attribute columns. To return specified attribute columns, turn off **All Columns** and enter the names of attribute columns that you want to return. Separate multiple attribute columns with commas (,).
   iii. Configure the **Primary Key Value** parameter of the row that you want to query.

   The integrity and accuracy of the primary key value affect the query results.
   iv. Set **Max Versions** to specify the maximum number of versions to return.

   > **Note** You can specify a maximum of 20 versions in the console. This limit does not apply when you use Tablestore SDKs.

6. Click **OK**.
   Data that meets the query conditions is displayed on the **Query Data** tab.

To query data within a specified range, perform the following steps:

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the required instance or click **Manage Instance** in the Actions column corresponding to the instance.
3. In the **Tables** section of the **Instance Details** tab, click the name of the required data table, and click the **Query Data** tab. You can also click **Query** in the Actions column of the data table.
4. On the **Query Data** tab, click **Search**.
5. Specify query conditions.
   i. In the **Search** dialog box, set Modes to **Range Search**.
   ii. By default, the system returns all attribute columns. To return specified attribute columns, turn off **All Columns** and enter the names of attribute columns that you want to return. Separate multiple attribute columns with commas (,).

iii. Set Start Primary Key Column and End Primary Key Column.

You can set Start Primary Key Column to **Min Value** or **Custom** and End Primary Key Column to **Max Value** or **Custom**. If you select **Custom**, enter a custom value.

> **Note**
>
> - The value in the first primary key column takes priority when the range query mode is used. When the start primary key column value and the end primary key column value are the same in the first primary key column, the system uses the value in the second primary key column to perform queries. The query rules for subsequent primary keys are the same as those for the first two primary keys.
> - The Custom range is a left-open and right-closed interval.

iv. Set **Max Versions** to specify the maximum number of versions to return.

> **Note** You can specify a maximum of 20 versions in the console. This limit does not apply when you use Tablestore SDKs.

v. Set Sequence to **Forward Search** or **Backward Search**.

6. Click **OK**.
   Data that meets the query conditions is displayed on the **Query Data** tab.

Delete data

You can delete data that you no longer need.

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the required instance or click **Manage Instance** in the Actions column corresponding to the instance.
3. In the **Tables** section of the **Instance Details** tab, click the name of the required data table, and click the **Query Data** tab. You can also click **Query** in the Actions column of the data table.
4. On the **Query Data** tab, select the row of data that you want to delete. Click **Delete**.
5. In the **Delete** message, click **OK**.

3.5. Bind a VPC to a Tablestore instance

After you bind a VPC to a Tablestore instance, you can access the Tablestore instance from the ECS instances in the VPC in the same region.

Prerequisites

- A VPC that is within the same region as the Tablestore instance is created.
- After the VPC is created, create an ECS instance in the VPC.

Procedure

1. Log on to the Tablestore console.

2. On the **Overview** page, click the name of the required instance or click **Manage Instance** in the Actions column corresponding to the instance.
3. Click the **Network Management** tab.
4. On the **Network Management** tab, click **Bind VPC**.
5. In the **Bind VPC** dialog box, select a VPC and switch, enter **Instance VPC Name**.

   The name of a VPC can contain only letters and digits and must start with a letter. The name must be 3 to 16 bytes in length.
6. Click **OK**.

   After the VPC is bound to the instance, you can view the information of the VPC in the **VPC List** on the **Network Management** tab. You can use the VPC address to access the Tablestore instance from the ECS instances in the VPC.

   After you bind a VPC, you can perform the following operations:

   - Click **VPC Instance List** in the Actions column to view the VPC instances list, which contains the instance name, instance VPC name, and VPC domain name.
   - Click **Unbind** in the Actions column to unbind the VPC from the instance. After the VPC is unbound, the ECS instance in the VPC can no longer access the Tablestore instance by using the VPC address. To access the Tablestore instance from the ECS instance, you must bind the VPC to the Tablestore instance again.

3.6. Use Tunnel Service

After the Stream feature is enabled, you can create tunnels for the data table to consume historical and incremental data in the data table.

Background information

Tunnel Service is built on the Tablestore API to provide tunnels that are used to consume data in full, incremental, and differential modes. You can create full, incremental, and differential tunnels and consume data that is stored in distributed manner through these tunnels in real time.

Enable Stream

After the Stream feature is enabled, the system periodically deletes expired Stream operation logs that have been stored longer than the specified period.

1. Log on to the Table Store console.
2. On the **Overview** page, click the name of the required instance or click Manage Instance in the **Actions** column of the required instance.
3. In the **Tables** section of the **Instance Details** tab, click the name of a data table and then click the Tunnels tab. You can also click in the Actions column that corresponds to the data table and select **Tunnels**.
4. On the **Tunnels** tab, find the **Stream Information** section and click **Enabled**.
5. In the **Enable Stream** dialog box, set **Log Expiration Time**.

> Note
>
> - **Log Expiration Time** specifies the duration after which Stream operation logs expire.
> - The unit of **Log Expiration Time** is hour. The value must be a non-zero integer and cannot be changed after it is specified. The maximum value can be set to 168.

6. Click **Enabled**.

Create a tunnel

After you create a tunnel for a data table, you can use the tunnel to consume historical and incremental data in the data table.

1. Log on to the Table Store console.
2. On the **Overview** page, click the name of the required instance or click Manage Instance in the **Actions** column of the required instance.
3. In the **Tables** section of the **Instance Details** tab, click the name of a data table and then click the Tunnels tab. You can also click in the Actions column that corresponds to the data table and select **Tunnels**.
4. On the **Tunnels** tab, click **Create Tunnel**.
5. In the **Create Tunnel** dialog box that appears, set **Tunnel Name** and **Type**.

   Tunnel Service provides three types of tunnels to consume data that is stored in distributed manner in real time: **Incremental**, **Full**, and **Differential**.
6. Click **OK**.

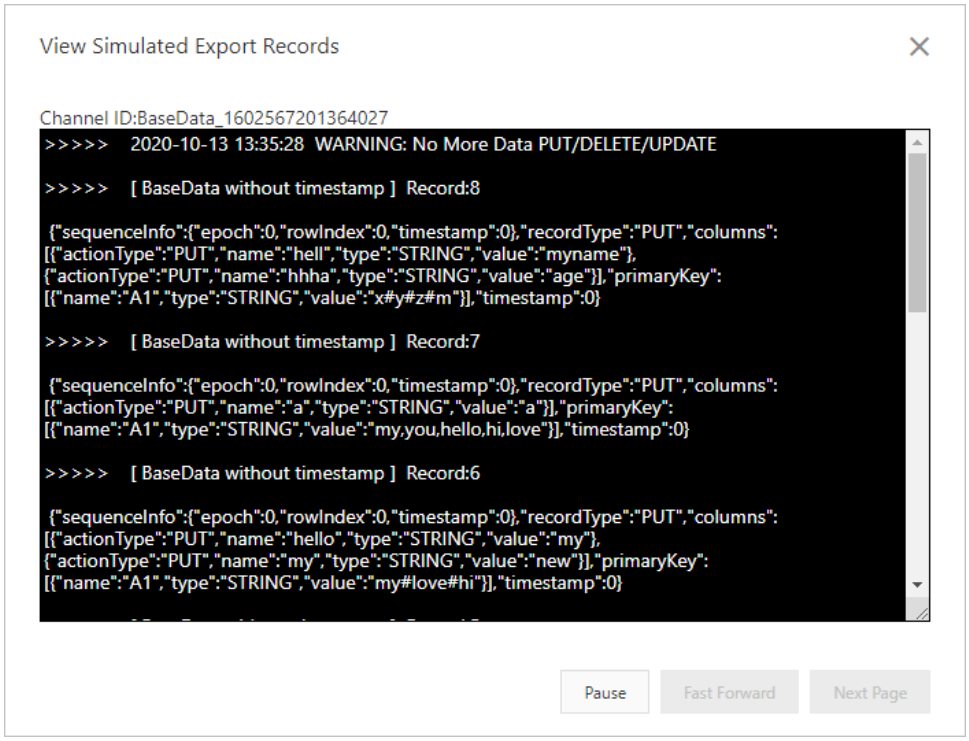
   After you create a tunnel, you can view information about the tunnel on the **Tunnels** tab.

Preview the data format of a tunnel

You can simulate data consumption by writing or deleting data and preview the data format of a tunnel.

1. Write data to or delete data from a data table. For more information about how to write or delete data in the console, see Read and write data in the console.
2. Preview the data format of a tunnel.
   i. Log on to the Table Store console.
   ii. On the **Overview** page, click the name of the required instance or click Manage Instance in the **Actions** column of the required instance.
   iii. In the **Tables** section of the **Instance Details** tab, click the name of the data table and then click the Tunnels tab. You can also click in the Actions column that corresponds to the data table and select **Tunnels**.
   iv. On the **Tunnels** tab, click **Show Channels** in the Actions column that corresponds to a tunnel.

v. In the channel list, click **View Simulated Export Records** in the Actions column of the required channel. In the dialog box that appears, click **Start**.



Enable data consumption for a tunnel

You can obtain the ID of a created tunnel and use the Tablestore SDK in any programming language for Tunnel Service to enable data consumption for the tunnel.

1. Obtain the ID of a created tunnel.
   i. Log on to the Table Store console.
   ii. On the **Overview** page, click the name of the required instance or click Manage Instance in the **Actions** column of the required instance.
   iii. In the **Tables** section of the **Instance Details** tab, click the name of a data table and then click the Tunnels tab. You can also click ⋮ in the Actions column that corresponds to the data table and select **Tunnels**.
   iv. On the **Tunnels** tab, copy the ID of a created tunnel.
2. Use the Tablestore SDK in any programming language for Tunnel Service to enable data consumption for the tunnel.

```
// Create a custom data consumption callback to implement the IChannelProcessor operati
on. Specify the process and shutdown methods.
private static class SimpleProcessor implements IChannelProcessor {
    @Override
    public void process(ProcessRecordsInput input) {
        System.out.println("Default record processor, would print records count");
        System.out.println(
            String.format("Process %d records, NextToken: %s", input.getRecords().size(
), input.getNextToken()));
        try {
            // Mock Record Process.
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void shutdown() {
        System.out.println("Mock shutdown");
    }
}
// Configure advanced parameters in TunnelWorkerConfig.
TunnelWorkerConfig config = new TunnelWorkerConfig(new SimpleProcessor());
// Configure TunnelWorker to start automatic data processing.
TunnelWorker worker = new TunnelWorker($tunnelId, tunnelClient, config);
try {
    worker.connectAndWorking();
} catch (Exception e) {
    e.printStackTrace();
    worker.shutdown();
    tunnelClient.shutdown();
}
```

4.Global secondary index

4.1. Usage notes

This topic describes the terms, limits, and precautions for global secondary indexes.

Terms

Term	Description
index table	The table created based on indexing of columns from the base table. The data in the index table is read-only.
predefined column	The column you predefine when you create a table. Tablestore uses a schema-free model. You can also specify the data type of the column. You can write an unlimited number of columns to a row. You do not need to specify a fixed number of predefined columns in a schema.
single-column index	The index that is created for a single column.
compound index	The index that is created for multiple columns in a table. A compound index can have indexed columns 1 and 2.
indexed attribute column	The predefined column in a base table that is mapped to non-primary key columns in an index table.
autocomplete	Tablestore automatically adds all primary key columns of the base table to the index table.

Limits

- The index table names must be unique in an instance.
- You can create a maximum of five index tables for a base table. If the limit is reached, the index table fails to be created.
- You can create a maximum of 20 predefined columns for a base table. If the limit is reached, the base table fails to be created.
- An index table can contain a maximum of four indexed columns, which are random combinations of the primary keys and predefined columns of the base table. If the limit is reached, the index table fails to be created.
- An index table can contain a maximum of eight attribute columns. If the limit is reached, the index table fails to be created.
- You can set the data type of an indexed column to STRING, INTEGER, or BINARY. The limits on index columns are the same as those on primary key columns of the base table.
- If an index table contains multiple columns, the size limit on the columns is the same as that on primary key columns of the base table.
- If you specify a column of the STRING or BINARY type as an attribute column of an index table, the limits on attribute columns are the same as those on attribute columns of the base table.
- You cannot create an index table on a table that has the time to live (TTL) parameter configured. If you want to create index tables on a table that has the TTL parameter configured, use DingTalk to

contact technical support.

- You cannot create an index table from a base table that has the max versions parameter configured. If a base table has the max versions parameter configured, index tables fail to be created from the base table. You cannot configure the max versions parameter for a base table that is associated with an index table.
- You cannot customize versions when you write data to a base table that is associated with an index table. Otherwise, the data fails to be written to the base table.
- You cannot use the Stream feature in an index table.
- An indexed base table cannot contain repeated rows that have the same primary key during the same batch write operation. Otherwise, the data fails to be written to the base table.

Usage notes

- Tablestore automatically adds all primary key columns of the base table to the index table. When you scan an index table, you must specify the range of primary key columns. The range can be anywhere from negative infinity to positive infinity. For example, a base table contains the primary key columns PK0 and PK1 and a predefined column Defined0.

  When you create an index for the Defined0 column, Tablestore generates an index table that has the primary key columns Defined0, PK0, and PK1. When you create an index for the Defined0 and PK1 columns, Tablestore generates an index table that has the primary key columns Defined0, PK1, and PK0. When you create an index for the PK1 column, Tablestore generates an index table that has the primary key columns PK1 and PK0. When you create an index table, you need only to specify the column that you want to index. Tablestore adds the other primary key columns of the central table to the index table. For example, a base table contains the primary key columns PK0 and PK1 and a predefined column Defined0.

  - When you create an index for the Defined0 column, Tablestore generates the index table that has the primary key columns Defined0, PK0, and PK1.
  - When you create an index for the PK1 column, Tablestore generates the index table that has the primary key columns PK1 and PK0.

- You can specify predefined columns as attribute columns in the base table. When you specify a predefined column of the base table as an attribute column of the index table, you can search this index table instead of the base table for the column value. However, this increases storage costs. Otherwise, you must query the base table based on the index table. You can choose between these methods.
- We recommend that you do not specify a column whose values are date or time as the first primary key column of an index table because it may slow down index table updates. We recommend that you hash columns related to the time or date and create indexes for the hashed columns. If you have similar requirements, use DingTalk to contact technical support.
- We recommend that you do not define a column of low cardinality or a column that contains enumerated values as the first primary key column of an index table. For example, the gender column restricts the horizontal scalability of the index table and leads to poor write performance.

4.2. Scenarios

Global secondary index allows you to create an index table based on a specified column. Data in the generated index is sorted by the specified index column. All data written to the base table is synchronized to the index asynchronously. If you only write data to a base table and query index tables created on the table, the query performance can be improved in most scenarios. This topic describes how to use a global secondary index to query phone records.

For example, the following table contains a number of phone records.

CellNumber	StartTime (Unix timestamps)	CalledNumber	Duration	BaseStationNumber
123456	1532574644	654321	60	1
234567	1532574714	765432	10	1
234567	1532574734	123456	20	3
345678	1532574795	123456	5	2
345678	1532574861	123456	100	2
456789	1532584054	345678	200	3

- The `CellNumber` and `StartTime` columns act as the primary key. CellNumber represents the `caller`. StartTime represents the `call start time`.
- The `CalledNumber`, `Duration`, and `BaseStationNumber` columns are predefined columns. CalledNumber represents the `call recipient`. Duration represents the `call duration`. BaseStationNumber represents the `base station number`.

When you end a phone call, information about the call is written to this table. You can create global secondary indexes for different query scenarios. For example, you can create global secondary indexes whose primary key is `CalledNumber` or `BaseStationNumber`.

Assume that you have the following query requirements:

- You want to query the rows where the value of CellNumber is `234567`.

  Tablestore uses a global ordering model, which sorts all rows by primary key and provides the `getRange` operation to perform sequential scans. When you use `getRange` to scan the base table for this example, you need only to set the minimum and maximum values of PK0 to `234567`, and set the minimum value of PK1 (call start time) to `0` and the maximum value of PK1 to `INT_MAX`.

```
private static void getRangeFromMainTable(SyncClient client, long cellNumber)
{
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(TABLE_NAME);
    // Specify the primary key to start from.
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.fromLo
ng(cellNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(0));
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    // Specify the primary key to end with.
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.fromLong
(cellNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strNum = String.format("%d", cellNumber);
    System.out.println("The cell number" + strNum + "makes the following calls:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        // If the nextStartPrimaryKey value is not null, continue the read operation.
        if (getRangeResponse.getNextStartPrimaryKey() ! = null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

- You want to query the rows where the value of CalledNumber is `123456`.

  Tablestore sorts all rows based on primary keys. Queries that involve this column are slow and inefficient because CalledNumber is a predefined column. Therefore, you create an index table based on `CalledNumber` to improve query speed and efficiency.

  `IndexOnBeCalledNumber`:

PK0	PK1	PK2
CalledNumber	CellNumber	StartTime
123456	234567	1532574734
123456	345678	1532574795
123456	345678	1532574861

PK0	PK1	PK2
654321	123456	1532574644
765432	234567	1532574714
345678	456789	1532584054

> **Note** Tablestore automatically adds all primary key columns of the central table to the index table. The primary key of the global secondary index consists of the index column and the primary key columns of the base table. Therefore, the global secondary index contains three primary key columns.

CalledNumber is a primary key column of `IndexOnBeCalledNumber`. You can perform a query on this index table to query the rows where the value of CalledNumber is 123456.

```
private static void getRangeFromIndexTable(SyncClient client, long cellNumber) {
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX0_NAME);
    // Specify the primary key to start from.
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_1, PrimaryKeyValue.fromLo
ng(cellNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    // Specify the primary key to end with.
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_1, PrimaryKeyValue.fromLong
(cellNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strNum = String.format("%d", cellNumber);
    System.out.println("The cell number" + strNum + "was called by the following numbers:
");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        // If the nextStartPrimaryKey value is not null, continue the read operation.
        if (getRangeResponse.getNextStartPrimaryKey() ! = null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

- You want to query the rows where the value of BaseStationNumber is `002` and the value of StartTime is `1532574740`.

  This query specifies `BaseStationNumber` and `StartTime` as conditions. Therefore, you can create a compound index based on the `BaseStationNumber` and `StartTime` columns.

  `IndexOnBaseStation1`:

PK0	PK1	PK2
BaseStationNumber	StartTime	CellNumber
001	1532574644	123456

PK0	PK1	PK2
001	1532574714	234567
002	1532574795	345678
002	1532574861	345678
003	1532574734	234567
003	1532584054	456789

The following code provides an example on how to query the `IndexOnBaseStation1` index table:

```
private static void getRangeFromIndexTable(SyncClient client,
                                           long baseStationNumber,
                                           long startTime) {
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX1_NAME);
    // Specify the primary key to start from.
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLo
ng(baseStationNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(startTime));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    // Specify the primary key to end with.
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLong
(baseStationNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MAX)
;
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strBaseStationNum = String.format("%d", baseStationNumber);
    String strStartTime = String.format("%d", startTime);
    System.out.println("All called numbers forwarded by the base station" + strBaseStatio
nNum + "that start from" + strStartTime + "are listed:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        // If the nextStartPrimaryKey value is not null, continue the read operation.
        if (getRangeResponse.getNextStartPrimaryKey() ! = null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

- You want to query the rows where the value of BaseStationNumber is `003` and the value of StartTime ranges from `1532574861` to `1532584054` and return only the Duration column.

  In this query, you specify both `BaseStationNumber` and `StartTime` as conditions, but only the `Duration` column is returned. You can initiate a query on the previous index table, and then query Duration by querying the base table.

```
private static void getRowFromIndexAndMainTable(SyncClient client,
                                                long baseStationNumber,
                                                long startTime,
```

```
                                                long endTime,
                                                String colName) {
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX1_NAME);
    // Specify the primary key to start from.
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLo
ng(baseStationNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(startTime));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    // Specify the primary key to end with.
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLong
(baseStationNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLong
(endTime));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strBaseStationNum = String.format("%d", baseStationNumber);
    String strStartTime = String.format("%d", startTime);
    String strEndTime = String.format("%d", endTime);
    System.out.println("The duration of calls forwarded by the base station" + strBaseSta
tionNum + "from" + strStartTime + "to" + strEndTime + "is listed:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            PrimaryKey curIndexPrimaryKey = row.getPrimaryKey();
            PrimaryKeyColumn mainCalledNumber = curIndexPrimaryKey.getPrimaryKeyColumn(PR
IMARY_KEY_NAME_1);
            PrimaryKeyColumn callStartTime = curIndexPrimaryKey.getPrimaryKeyColumn(PRIMA
RY_KEY_NAME_2);
            PrimaryKeyBuilder mainTablePKBuilder = PrimaryKeyBuilder.createPrimaryKeyBuil
der();
            mainTablePKBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, mainCalledNumber.g
etValue());
            mainTablePKBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, callStartTime.getV
alue());
            PrimaryKey mainTablePK = mainTablePKBuilder.build(); // Specify primary keys
for the base table.
            // Query the base table.
            SingleRowQueryCriteria criteria = new SingleRowQueryCriteria(TABLE_NAME, main
TablePK);
            criteria.addColumnsToGet(colName); // Read the Duration column value of the b
ase table.
            // Set the latest version to read.
            criteria.setMaxVersions(1);
            GetRowResponse getRowResponse = client.getRow(new GetRowRequest(criteria));
            Row mainTableRow = getRowResponse.getRow();
```

```
            System.out.println(mainTableRow);
        }
        // If the nextStartPrimaryKey value is not null, continue the read operation.
        if (getRangeResponse.getNextStartPrimaryKey() ! = null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

To improve query performance, you can create a compound index based on `BaseStationNumber` and `StartTime` and specify `Duration` as an attribute column of the index table.

The following index table is created.

`IndexOnBaseStation2`:

PK0	PK1	PK2	Defined0
BaseStationNumber	StartTime	CellNumber	Duration
001	1532574644	123456	60
001	1532574714	234567	10
002	1532574795	345678	5
002	1532574861	345678	100
003	1532574734	234567	20
003	1532584054	456789	200

The following code provides an example on how to query the `IndexOnBaseStation2` index table:

```
private static void getRangeFromIndexTable(SyncClient client,
                                           long baseStationNumber,
                                           long startTime,
                                           long endTime,
                                           String colName) {
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX2_NAME);
    // Specify the primary key to start from.
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLo
ng(baseStationNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(startTime));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    // Specify the primary key to end with.
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLong
(baseStationNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLong
(endTime));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    // Specify the name of the column to read.
    rangeRowQueryCriteria.addColumnsToGet(colName);
    rangeRowQueryCriteria.setMaxVersions(1);
    String strBaseStationNum = String.format("%d", baseStationNumber);
    String strStartTime = String.format("%d", startTime);
    String strEndTime = String.format("%d", endTime);
    System.out.println("The duration of calls forwarded by the base station" + strBaseSta
tionNum + "from" + strStartTime + "to" + strEndTime + "is listed:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        // If the nextStartPrimaryKey value is not null, continue the read operation.
        if (getRangeResponse.getNextStartPrimaryKey() ! = null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

If you do not specify `Duration` as an attribute column for an index table, you must retrieve Duration by querying the base table. However, when you specify `Duration` as an attribute column for an index table, this column is stored in both the base table and the index table. The configuration improves query performance at the cost of storage space consumption.

- You want to query the total call duration, average call duration, maximum call duration, and minimum call duration of all calls forwarded by the base station `003` and whose call start time range from `1532574861` to `1532584054`.

  In this query, you want to query the statistics for the duration of all phone calls instead of the duration of each call that is queried in the previous scenario. You can obtain results by using the same method as in the previous query. Then, you can perform calculations on the Duration column to obtain the required result. You can also use SQL-on-OTS to directly return the final statistical results without the need for client computing. You can use most MySQL syntax in SQL-on-OTS. Additionally, SQL-on-OTS enable you to process complicated calculations that are applicable to your business.