Alibaba Cloud Table Store

Functions

Issue: 20190614

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 due to product version upgrades , adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall 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 the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults " and "as available" basis. 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 bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, 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 content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, 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 the Alibaba Cloud website, product programs, or content 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 contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information 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.
	This warning information 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 restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	 Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus , page names, and other UI elements.	Click OK.
`Courier font`	It is used for commands.	Run the `cd / d C :/ windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*

Style	Description	Example
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich {stand | slave}`

Contents

1 Table Store tables

When creating a Table Store table, you must specify a table name, a primary key, and reserved read/write throughput.

Naming conventions

Table Store table names:

- Can contain uppercase letters, lowercase letters, digits, and underscores.
- Must start with an uppercase letter, an lowercase letter, or an underscore.
- Are case sensitive.
- Must be 1 to 255 characters in length.
- Must be unique within the same instance (tables in different instances are allowed to use the same name).

Primary Key

When creating a Table Store table, you must specify the primary key of the table. A primary key contains at least one, and up to four primary key columns. Each primary key column has a name and type. Table Store has some restrictions on the names and types of the primary key columns. For more information, see *Primary key and attribute*.

Table Store indexes data based on the primary key. The primary key uniquely identifies each row in the table, so that no two rows have the same key. The rows are sorted in ascending order by their primary key.

Reserved read/write throughput

To guarantee the consistent and low-latency performance of Table Store, you can specify the reserved read/write throughput during table creation. If the value of the reserved read/write throughput is not 0, Table Store reserves the necessary capacity to meet the specified throughput requirements. At the same time, costs are determined based on the reserved read/write throughput. You can dynamically raise and lower the reserved read/write throughput based on business requirements. The reserved read/write throughput is set in quantities of read capacity units and write capacity units.

 **Note:**

Tables created in capacity instances do not support the reserved read/write throughput.

You can update the tables reserved read/write throughput through the UpdateTable operation. The rules for updating the reserved read/write throughput are as follows.

- A time interval of at least two minutes is required between two updates for the same table. For example, if you update the reserved read/write throughput of a table at 12:43:00, you must wait until after 12:45:00 to update the table for a second time. The required 2-minute time interval between updates is applied at the table level. Between 12:43:00 and 12:45:00, you can update the reserved read/write throughput for other tables.
- The frequency of adjusting the reserved read/write throughput in a calendar day (00:00:00 to 00:00:00 of the second day in UTC time) is unlimited. The adjustment interval must be more than two minutes. Adjusting the reserved read/write throughput of a table is defined as adjusting either the read capacity unit or write capacity unit setting. Such an operation is considered as updating the table.
- A reserved read/write throughput adjustment takes effect within one minute.

The consumed read/write throughput that exceeds the value of the reserved read /write throughput is classified as additional read/write throughput. Costs are calculated based on the unit price of the additional read/write throughput.

Initially, your applications may not have a high throughput. Depending on your business requirements, you can set a low reserved read/write throughput to minimize costs. As your business expands, you can increase the reserved read/write throughput of the table to reflect new business requirements. If you want to quickly import a large volume of data immediately after creating a table, you can set a high reserved write throughput to import the data quickly. After the large volume data import is completed, you can lower the reserved read/write throughput.

Data size restrictions of partition key

Table Store partitions the table data according to the partition key ranges. Rows with the same partition key are placed in the same partition. To prevent large indivisible partitions, we recommend that the total data size for all rows under a single partition key value must not exceed 10 GB.

Table Store load time

Table Store table is ready within one minute after it is created. You must wait for the table to finish loading before performing any data operations.

Best Practice

See Table operations

Table Store SDKs

Use Table Store Python SDK for table operations

2 Conditional update

A conditional update is an update of table data that executes only when specified conditions are met. A conditional update can be based on a combination of up to 10 conditions. Supported conditions include arithmetic operations (=, ! =, >, >=, <, and <=) and logical operations (NOT, AND, and OR). The conditional update is applicable to *PutRow*, *UpdateRow*, *DeleteRow*, and *BatchWriteRow*.

The column-based judgment conditions include the row existence condition and column-based condition.

- The *Row existence condition* is classified into `IGNORE`, `EXPECT_EXI ST`, and `EXPECT_NOT _EXIST`. When a table needs to be updated, the system first checks the row existence condition. If the row existence condition is not met, an error occurs during the update.
- The column-based condition supports `SingleColu mnValueCon dition` and `CompositeC olumnValue Condition`, which are used to perform the condition-based judgment based on the values of a column or certain columns, similar to the conditions used by the Table Store filters.

Conditional update also supports optimistic locking strategy. That is, when a row needs to be updated, the system first obtains the value of a column. For example, the value of Column A is 1, and its condition is set as `Column A = 1`. Set `Column A = 2`, then update the row. If a failure occurs during the update, it means that the row has been successfully updated by another client.

 **Note:**
In highly concurrent applications such as webpage view counting or gaming (where atomic counter updates are required), the probability of failed conditional updates is high. If this occurs, we recommend that you retry the update until successful.

Procedure

1. Construct SingleColumnValueCondition.

```
// set condition Col0 == 0 .
 SingleColu mnValueCon dition singleColu mnValueCon dition
= new SingleColu mnValueCon dition (" Col0 ",
 SingleColu mnValueCon dition . CompareOpe rator . EQUAL
, ColumnValu e . fromLong ( 0 ));
```

```
// If column Col0 does not exist , the condition
check fails .
 singleColu mnValueCon dition . setPassIfM issing ( false );
// Only check the latest version
 singleColu mnValueCon dition . setLatestV ersionsOnl y ( true
);
```

2. **Construct CompositeColumnValueCondition.**

```
// condition composite1 is ( Col0 == 0 ) AND ( Col1 >
100 )
 CompositeC olumnValue Condition composite1 = new
CompositeC olumnValue Condition ( CompositeC olumnValue
Condition . LogicOpera tor . AND );
 SingleColu mnValueCon dition single1 = new SingleColu
mnValueCon dition (" Col0 ",
         SingleColu mnValueCon dition . CompareOpe rator . EQUAL
, ColumnValu e . fromLong ( 0 ));
 SingleColu mnValueCon dition single2 = new SingleColu
mnValueCon dition (" Col1 ",
         SingleColu mnValueCon dition . CompareOpe rator .
GREATER_TH AN , ColumnValu e . fromLong ( 100 ));
 composite1 . addConditi on ( single1 );
 composite1 . addConditi on ( single2 );

// condition composite2 is ( ( Col0 == 0 ) AND ( Col1 >
100 ) ) OR ( Col2 <= 10 )
 CompositeC olumnValue Condition composite2 = new
CompositeC olumnValue Condition ( CompositeC olumnValue
Condition . LogicOpera tor . OR );
 SingleColu mnValueCon dition single3 = new SingleColu
mnValueCon dition (" Col2 ",
         SingleColu mnValueCon dition . CompareOpe rator .
LESS_EQUAL , ColumnValu e . fromLong ( 10 ));
 composite2 . addConditi on ( composite1 );
 composite2 . addConditi on ( single3 );
```

3. **Implement an increasing column by the optimistic locking strategy based on the conditional update.**

```
 private static void updateRowW ithConditi on ( SyncClient
  client , String pkValue ) {
    // construct the primary
     PrimaryKey Builder primaryKey Builder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
     primaryKey Builder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME , PrimaryKey Value . fromString ( pkValue ));
     PrimaryKey primaryKey = primaryKey Builder . build ();

    // read a row
     SingleRowQ ueryCriter ia criteria = new SingleRowQ
ueryCriter ia ( TABLE_NAME , primaryKey );
     criteria . setMaxVers ions ( 1 );
     GetRowResp onse getRowResp onse = client . getRow ( new
  GetRowRequ est ( criteria ));
     Row row = getRowResp onse . getRow ();
     long col0Value = row . getLatestC olumn (" Col0 ").
getValue (). asLong ();

    // Col0 = Col0 + 1 by conditiona l update
     RowUpdateC hange rowUpdateC hange = new RowUpdateC
hange ( TABLE_NAME , primaryKey );
```

```
     Condition   condition  =  new   Condition ( RowExisten
ceExpectat  ion . EXPECT_EXI  ST );
     ColumnCond  ition   columnCond  ition  =  new   SingleColu
mnValueCon  dition (" Col0 ",  SingleColu  mnValueCon  dition .
CompareOpe  rator . EQUAL ,  ColumnValu  e . fromLong ( col0Value
));
     condition . setColumnC  ondition ( columnCond  ition );
     rowUpdateC  hange . setConditi  on ( condition );
     rowUpdateC  hange . put ( new   Column (" Col0 ",  ColumnValu
e . fromLong ( col0Value  +  1 )));

     try  {
         client . updateRow ( new   UpdateRowR  equest (
rowUpdateC  hange ));
     }  catch  ( TableStore  Exception   ex ) {
         System . out . println ( ex . toString ());
     }
}
```

Example

The following operations are examples of updates that are executed for highly concurrent applications:

```
//  Get   the   old   value
    old_value  =  Read ();
//  compute   such   as   increment   1
    new_value  =  func ( old_value );
//  Update   by   the   new   value
    Update ( new_value );
```

The conditional update makes sure `Update ( new_value )` if value equals to `old_value` in a highly concurrent environment where `old_value` may be updated by another client.

Billing

Writing or updating data successfully does not affect the capacity unit (CU) calculation rules of the interfaces. However, if the conditional update fails, one unit of write CU and one unit of read CU are consumed, which are billable.

3 Auto-increment function of the primary key column

If you set a primary key column as an auto-increment column, you do not need to enter this column when writing data in a row. Instead, Table Store automatically generates the primary key value, which is unique in the partition key, and which increases progressively.

Features

Table Store, in conjunction with the auto-increment function of an primary key column, has the following features:

- The system architecture exclusive to Table Store and the implementation through an auto-increment primary key column make sure that the value generated for the auto-incrementing column is unique and strictly incrementing.
- The automatically generated auto-increment column value is a 64-bit signed long integer.
- The level of the partition key increases progressively.
- The auto-increment function is a table level. The tables with an auto-increment column and the tables without an auto-increment column can be created in the same instance.

If the auto-increment primary key column is set, the conditional update logic is not changed. See the following table for more information.

API	IGNORE	EXPECT_EXIST	EXPECT_NOT_EXIST
PutRow: The row exists.	Fail	Succeed	Fail
PutRow: The row does not exist.	Succeed	Fail	Fail
UpdateRow: The row exists.	Fail	Succeed	Fail
UpdateRow: The row does not exist.	Succeed	Fail	Fail

API	IGNORE	EXPECT_EXIST	EXPECT_NOT_EXIST
DeleteRow: The row exists.	Fail	Fail	Fail
DeleteRow: The row does not exist.	Succeed	Succeed	Fail

Limits

Table Store Auto-increment function of the primary key column mainly has the following restrictions:

- Table Store supports multiple primary keys. The first primary key is a partition key that cannot be set as an auto-increment column. However, one of other primary keys can be set as an auto-increment column.
- Only one primary key per table can be set as an auto-increment column.
- The attribute column cannot be set as an auto-increment column.
- The auto-increment column can only be set at the time the table is created. The existing table cannot set the auto-increment column.

Interface

- CreateTable
  - Set a column as an auto-incrementing column during table creation. For more information, see *Primary key column auto-increment*.
  - After table creation, you cannot configure the auto-incrementing feature of the table.
- UpdateTable

  You cannot change the auto-increment attribute of a table by using UpdateTable.
- PutRow/UpdateRow/BatchWriteRow
  - When writing the table, you do not need to set specific values for the column that you want to set as auto-incrementing. You only need to set a placeholder, for example, AUTO_INCREMENT. For more information, see *Primary key column auto-increment*.
  - You can set ReturnType in ReturnContent as RT_PK, that is, to return the complete primary key value, which can be used in the GetRow query.

- GetRow/BatchGetRow

  GetRow requires a complete primary key column, which can be obtained by setting ReturnType in PutRow, UpdateRow, or BatchWriteRow as RT_PK.

- Other interfaces

  Not changed

Usage

Java SDK: Auto-increment of the primary key column

Billing

The auto-increment function of primary key columns does not affect the existing billing logic. Returned data of the primary key column does not consume additional read CUs.

4 Description of the data consumption framework

Tunnel Service uses comprehensive operations of Table Store to consume full and incremental data. You can easily consume and process history data and incremental data in tables.

A Tunnel client is an automatic data consumption framework of Tunnel Service. The Tunnel client regularly checks heartbeats to detect active channels, update status of the Channel and ChannelConnect classes, initialize, run, and terminate data processing tasks.

The Tunnel client supports the following features for processing full and incrementa l data: load balancing, fault recovery, checkpoints, and partition information synchronization to ensure the sequence of consuming information. The Tunnel client allows you to focus on the processing logic of each record.

The following sections describe the features of the Tunnel client, including automatic data processing, load balancing, and fault tolerance. For more information, see *Github* to check source code of the Tunnel client.

Automatic data processing

The Tunnel client regularly checks for heartbeats to detect active channels, update status of the Channel and ChannelConnect classes, initialize, run, and terminate data processing tasks. This section describes the data processing logic. For more information, see source code.

1. Initialize resources of the Tunnel client.
   a. Change the status of the Tunnel client from Ready to Started.
   b. Set the HeartbeatTimeout and ClientTag parameters in TunnelWorkerConfig to run the ConnectTunnel task and connect Tunnel Service to obtain the ClientId of the current Tunnel client.
   c. Initialize the ChannelDialer class to create a ChannelConnect task. Each ChannelConnect class corresponds to a Channel class, and the ChannelConnect task records data consumption checkpoints.
   d. Set the Callback parameter for processing data and the CheckpointInterval parameter for specifying the interval of outputting checkpoints in Tunnel

Service. In this way, you can create a data processor that automatically outputs checkpoints.

e. Initialize the TunnelStateMachine class to automatically update the status of the Channel class.

2. Regularly check heartbeat messages.

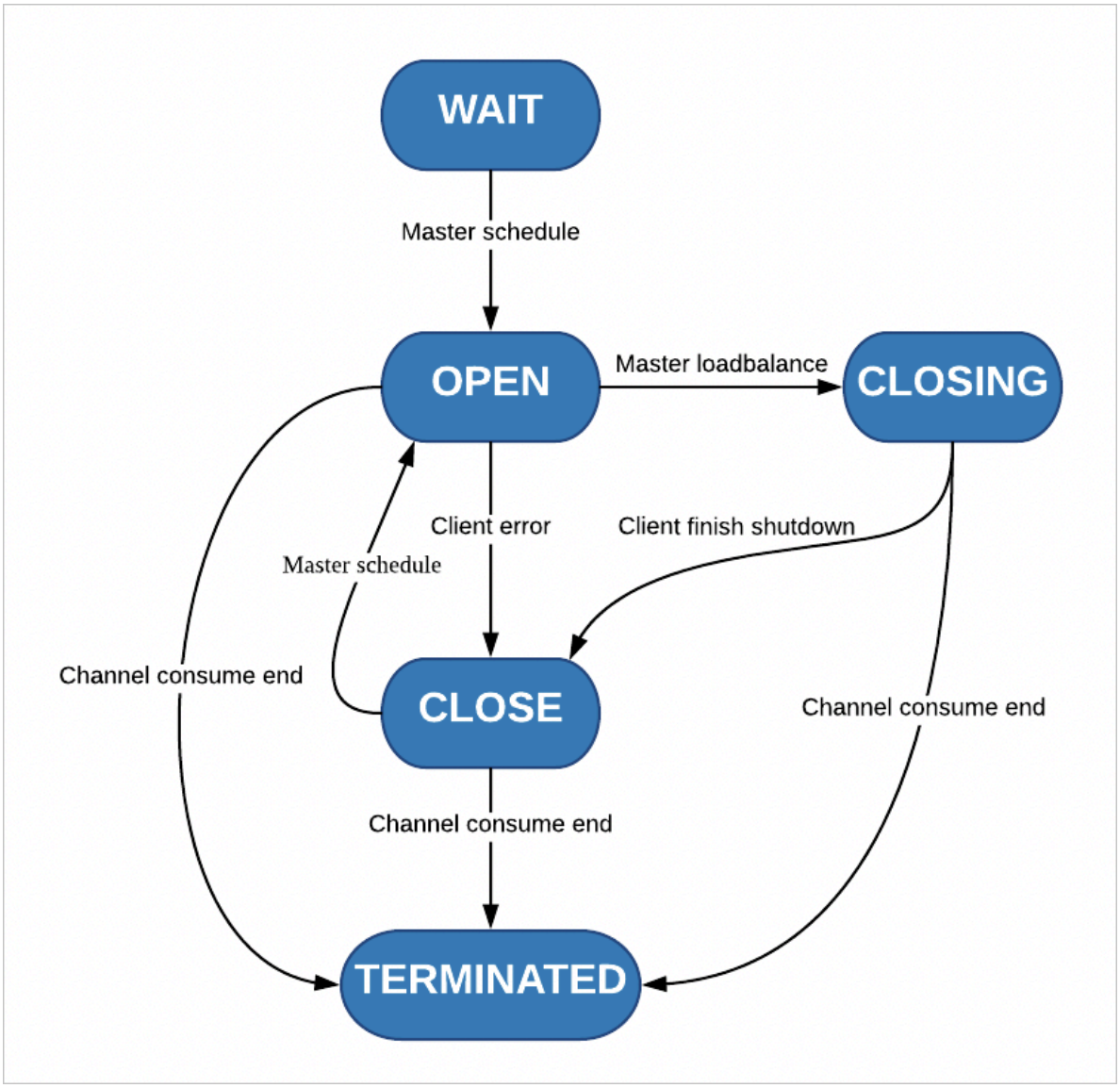
   You can set the heartbeatIntervalInSec parameter in TunnelWorkerConfig to set the interval for checking the heartbeat.

   a. Send a heartbeat request to obtain the list of latest available channels from Tunnel Service. The list includes the ChannelId, channel versions, and channel status.
   b. Merge the list of channels from Tunnel Service with the local list of channels, and create and update ChannelConnect tasks. Follow these rules:
      - Merge: overwrite the earlier version in the local list with the later version for the same ChannelId from Tunnel Service, and insert the new channels from Tunnel Service into the local list.
      - Create a ChannelConnect task: create a ChannelConnect task in WAIT status for a channel that has no ChannelConnect task. If the ChannelConnect task corresponds to a channel in OPEN status, run the ReadRecords&&ProcessRecords task that cyclically processes data for this ChannelConnect task. For more information, see the ProcessDataPipeline class in source code.
      - Update an existing ChannelConnect task: after you merge the lists of channels, if a channel corresponds to a ChannelConnect task, update the ChannelConnect status according to the status of channels with the same ChannelId. For example, if channels are in Close status, set their ChannelConnect tasks to the Closed status to terminate the corresponding pipeline tasks. For more information, see the ChannelConnect.notifyStatus method in source code.

3. Automatically process channel status.

   Based on the number of active Tunnel clients obtained in the heartbeat request, Tunnel Service allocates available partitions to different clients to balance the

loads. Tunnel Service automatically processes channel status as described in the following figure, and drives channel consumption and load balancing.



Tunnel Service and Tunnel clients change their status by using heartbeat requests and channel version updates.

a. Each channel is initially in WAIT status.
b. The channel for incremental data changes to the OPEN status only when the channel consumption on the parent partition is terminated.
c. Tunnel Service allocates the partition in OPEN status to each Tunnel client.
d. During load balancing, Tunnel Service and Tunnel clients use a scheduling protocol for changing a channel status from Open, Closing to Closed. After consuming a BaseData channel or a Stream channel, Tunnel clients report the channel as Terminated.

Automatic load balancing and excellent horizontal scaling

- Multiple Tunnel clients can consume data by using the same Tunnel or TunnelId . When the Tunnel clients run the heartbeat task, Tunnel Service automatically redistributes channels and tries to allocate active channels to each Tunnel client to achieve load balancing.
- You can easily add Tunnel clients to scale out. Tunnel clients can run on one or more instances.

Automatic resource clearing and fault tolerance

- Resource clearing: if Tunnel clients do not shut down normally, such as exceptiona l exit or manual termination, the system recycles resources automatically. For example, the system can release the thread pool, call the shutdown method that you have registered for the corresponding channel, and terminate the connection to Tunnel Service.
- Fault tolerance: when a Tunnel client has non-parametric errors such as heartbeat timeout, the system automatically renews connections to continue stable data consumption.

5 Stream

5.1 Overview

Table Store Stream is a data channel that retrieves incremental data from Table Store tables.

You can use the Table Store Stream API to obtain these changes. You can process incremental data streams in real time and replicate changes.

How Stream works

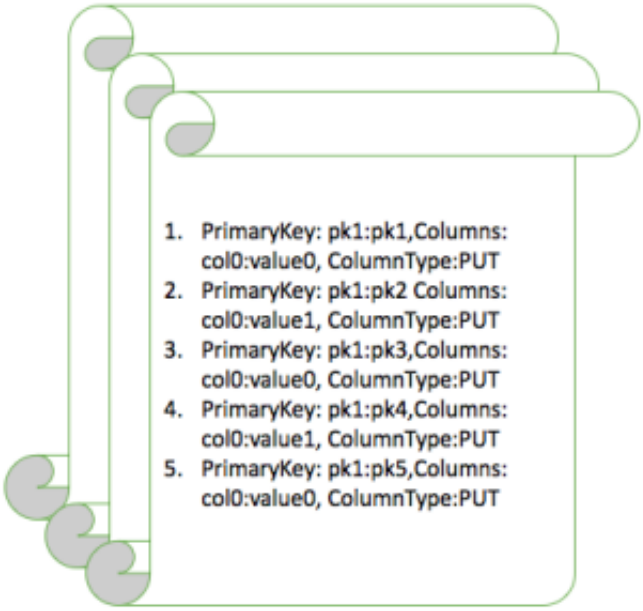
As a distributed NoSQL database, Table Store stores changes in the commit logs of Table Store when executing write operations (including put, delete, and update). Meanwhile, the database also performs regular checkpoints to flush earlier commit log entries.

When Stream is enabled, the log file is retained. The incremental data can be read through the channels provided by Stream during the retention period.

Table Store stores data in shards. Therefore, operations made to the same shard share one commit log. The incremental data is also retrieved at shard level.

When Stream is enabled, the system generates and maintains an offset value (an iterator) to indicate the current read position. You can obtain the iterator of the current shard using the `GetShardIt erator` operation. The iterator can be passed in later when you read incremental data stored in this shard. This makes sure that Stream knows which row of log records to read from and return the incremental data. When the incremental data is returned, Stream also returns a new offset for subsequent reads. The whole process can be compared to reading paged data where the iterator is equivalent to the offset of the page.

For example, your database generates some database log files in sequence, as shown in the following figure.



When you enable Stream on row 3 of file A, the iterator points to row 3 of file A. When reading data, you can pass in the iterator to read modifications that occurred after the third operation pk3 in this figure.

The Stream API also provides an operation to disable this data stream. When you enable it again, Stream generates a new iterator for the current shard, pointing to a new offset that marks the current time. You can use this iterator to read incremental data that occurs after the current time.

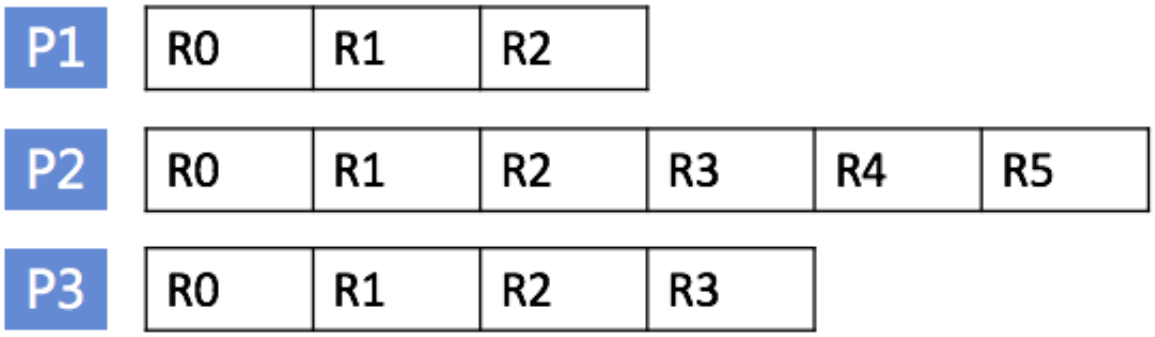
Write operations that occur on the same primary key must be read in sequence to guarantee consistency. However, before reading the incremental data, you do not know which primary keys have changes. Therefore, the operation for reading incremental data takes a shard ID and reads from a specific shard. To read the incremental data of the entire table, you can list all shards of the current table. Stream makes sure that write operations made to the same shard are returned in the sequence they were made. In this way, data changes made to a specific shard are read in the same sequence as they were written, and the data consistency for the same primary key can be guaranteed. If you continue to read the Stream data for all the shards, you can make sure that all incremental data in the table is read.

You can either enable Stream when creating a table or enable or disable Stream later using the `UpdateTabl e` operation. When a `put` , `update` , or `delete` operation occurs, a modification record is written to Stream. The record indicates the primary key values of the row that you modified and the actual modifications.

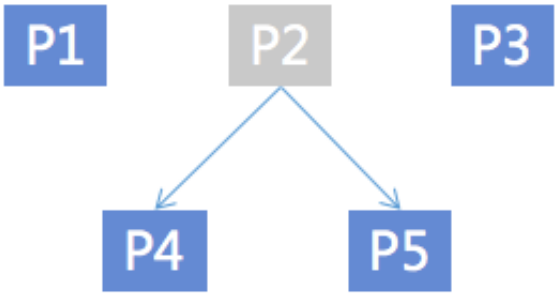
Note:

- Each modification record exists in Stream only once.
- For each shard, Stream processes modifications in the sequence they were made. However, modifications made to different shards are not sequenced.

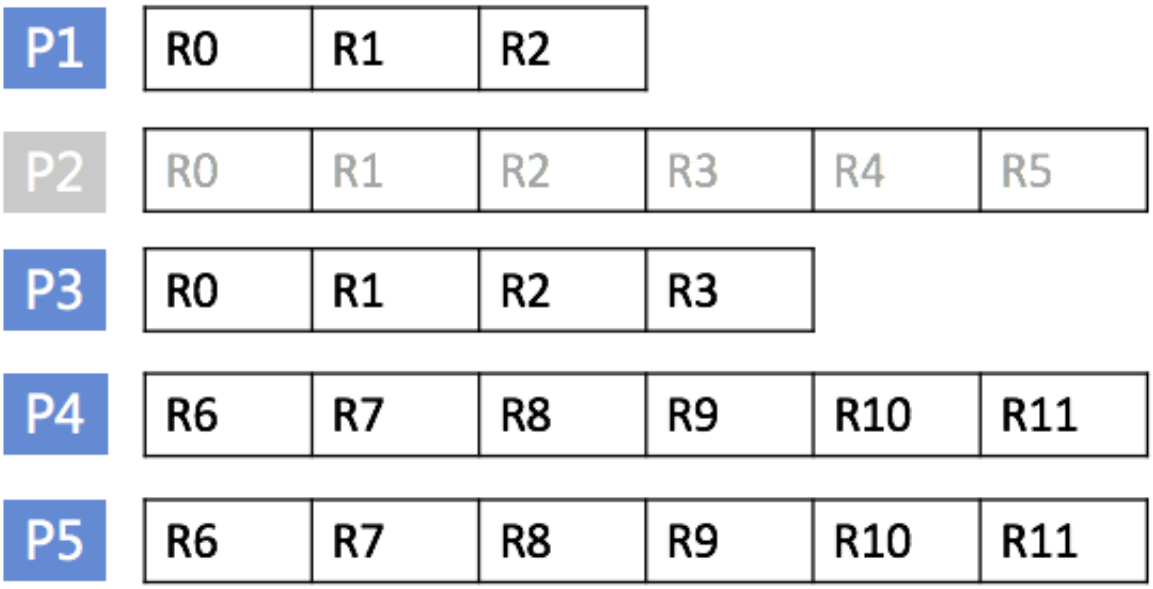
Example



In this figure, the current table has three shards. Each row in this figure represents a shard, and each column represents an update operation on a specific shard. Each shard maintains its own update log. You can use the `DescribeStream` operation to obtain information about the shard, and then read the changes in sequence for this shard. However, the system may split or merge shards in response to varying loads. New shards are created during merge or split operations, and earlier shards no longer generate new incremental data.



In this figure, shard P2 splits into shards P4 and P5. You can read data from shards P4 and P5 in parallel, without affecting one another. However, before you read from shards P4 and P5, make sure that you have read all incremental data on shard P2.



For example, in this figure, when you start reading the R6 log entry of shard P4, make sure that R5 of shard P2 has already been read. After R5 is read, shard P2 does not generate new data.

5.2 Stream API/SDK

API

- Enable and disable Stream

  You can specify whether Stream is enabled or disabled when creating a table. Also, you can use the `UpdateTable` operation to enable or disable Stream later. The `CreateTable` and `UpdateTable` operations now include a `StreamSpecification` parameter that allows you to set Stream parameters:

  - enable_stream: Whether to enable Stream.
  - expiration_time: Stream data expiration time. Expired modification log entries are deleted.

- Read modification logs

  To read Stream data, follow these steps:

  1. Call `ListStream s` to obtain the current table's Stream information, such as Stream ID. For more information, see *ListStream*.
  2. Call `DescribeSt ream` to obtain the current Stream's data shard information, such as the shard list. Each shard log contains shard information such as the parent shard and shardID. For more information, see *DescribeStream*.
  3. After obtaining StreamID and shardID, use `GetShardIt erator` to obtain the current shard's read iterator value. This value marks the starting position for reading the shard log. For more information, see *GetShardIterator*.
  4. Call `GetStreamR ecord` to read the specific modification logs. Each call returns a new iterator for the next read to use. For more information, see *GetStreamRecord*.

  > Notice:
  >
  > - Operations made to the same primary key have to be sequenced. Stream makes sure that operations made to the same shard are sequenced. However, shards may be split or merged, so before you read the data of a shard, make sure that data of the shard's parent shard and parent_sibling has been read.
  > - When an empty `NextShardI terator` is returned, it indicates that incremental data in the current shard has been fully read. This situation occurs typically when the shard is inactive after a split or merge operation. When a shard has been fully read, you can call `DescribeSt ream` again to retrieve information about the new shard.

SDK

Table Store Java SDK supports the Stream interface. For more information, see *Java SDK*.

5.3 Stream Client

You can use Table Store Stream APIs and Table Store SDKs to read Stream records. When you obtain incremental data in real time mode, note that information in shards is not static. Shards may be split or merged. When shards are changed, you must

process the dependencies between them to make sure that data in a single primary key is read in sequence. In addition, if your data is generated concurrently from multiple clients, multiple consumers must concurrently read the incremental records in each shard to improve the efficiency of exporting incremental data.

Stream Client is used to resolve common problems during Stream data processing , for example, load balancing, fault recovery, checkpoint, and shard information synchronization to guarantee the information consumption sequence. After using Stream Client, you only need to focus on the processing logic of each record.

This topic describes the principles of Stream Client, and how to use Stream Client to efficiently build a data tunnel that is applicable to your own services.

How Stream Client works

To easily implement job scheduling and record the read progress of each current shard, Stream Client uses a table of Table Store to record the information. You can customize the table name, but you must make sure that this table name is not used by other services.

Stream Client defines a lease for each shard, and the owner of each lease is called the worker. A lease is used to record the incremental data consumers (that is, workers) of the shard and read the progress. When a new consumer is started, the worker is initialized, checking the shard and lease information and creating a lease for a shard if the shard does not have one. When a new shard is generated from shard splitting or combination, Stream Client inserts a lease record into the table. The new record is grabbed and continuously processed by a worker of a Stream Client. If a new worker joins, load balancing is implemented to dispatch the record to the new worker.

The following table describes the schema of the lease record.

Parameter	Description
Primary key StreamId	ID of the currently processed Stream.
Primary key StatusType	Key of the current lease.
Primary StatusValue	ID of the shard corresponding to the current lease.
Attribute Checkpoint	Location where Stream data is consumed in the current shard (for user fault recovery).

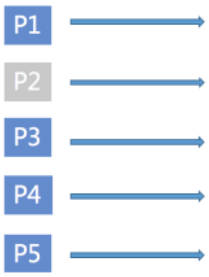
Parameter	Description
Attribute LeaseCounter	Optimistic lock. The owner of each lease continues to update the counter value. Lease renewal indicates that the current lease is continuously occupied.
Attribute LeaseOwner	Name of the worker that owns the current lease.
Attribute LeaseStealer	Worker to which the lease is to be moved during load balancing.
Attribute ParentShardIds	Parent shard of the current shard. When the worker is consuming the current shard, make sure that the Stream of the parent shard has been consumed.

Example

The following figure shows a typical distributed architecture of using Stream Client to consume incremental data.

In this figure, worker1 and worker2 are two consumers based on Stream Client, for example, programs started on the ECS. The data source constantly reads/writes a table in Table Store. In the initial stage, the table contains shards P1, P2, and P3. With the increase of the traffic and data volume, P2 is split into P4 and P5. In the initial stage, worker1 consumes data of P1, and worker2 consumes data of P2 and P3. After P2 is split, P4 will be allocated to worker1, and P5 will be allocated to worker2. However, Stream Client makes sure that data of P4 and P5 is consumed after consumption of record R5 of P2 is complete. If a new consumer worker3 is deployed at this time, a shard on worker2 may be dispatched to worker3, resulting in load balancing.

In the preceding scenario, Stream Client generates the following lease information in the table:



StreamId	StatusType	StatusValue	Checkpoint	Counter	owner	Stealer	ParentShardIds
Table_123456	LeaseKey	ShardId1	checkpoint1	101	worker1		
Table_123456	LeaseKey	ShardId2	checkpoint2	95	worker2		
Table_123456	LeaseKey	ShardId3	checkpoint3	102	worker2		
Table_123456	LeaseKey	ShardId4	checkpoint4	55	worker1		p2
Table_123456	LeaseKey	ShardId5	checkpoint5	55	worker2		p2

The worker in Stream Client is the carrier of the consumed Stream data. Each shard is allocated to a worker (lease owner). The owner constantly renews the current shard lease through heartbeats, that is, by updating LeaseCounter. Generally, each Steam consumer has a worker. After the worker is initialized, it obtains information about the shard to be processed. At the same time, the worker maintains its own thread pool , and concurrently and cyclically pulls incremental data of each shard it owns. The worker initialization process is as follows:

1. Reads the Table Store configuration and initializes the client that accesses Table Store through an intranet.
2. Obtains the Stream information of the corresponding table and initializes the lease management class. The lease management class synchronizes the lease informatio n and creates a new lease record for the new shard.
3. Initializes the shard synchronization class, which maintain the heartbeats of the current owned shards.
4. Cyclically obtains the incremental data of the shard currently owned by the worker .

Download Stream Client

- Download and install the *JAR package*
- Maven:

```
< dependency >
  < groupId > com . aliyun . openservic  es </ groupId >
  < artifactId > tablestore - streamclie  nt </ artifactId >
  < version > 1 . 0 . 0 </ version >
```

```
</ dependency >
```

Note:

The code of Stream Client is open-sourced. You can *download the source code* to learn about the principle. You are also welcomed to share good Stream-based sample code with us.

Use Stream Client APIs

Stream Client provides the `IRecordPro cessor` API, facilitating you to use Stream Client to consume the Stream data and hide the shard read logic and dispatch logic. The worker of Stream Client calls the `processRec ords` function after pulling the Steam data to trigger your data processing logic.

```
public interface IRecordPro cessor {

    void initialize ( Initializa tionInput initializa tionInput
);

    void processRec ords ( ProcessRec ordsInput processRec
ordsInput );

    void shutdown ( ShutdownIn put shutdownIn put );
}
```

The parameters are described as follows:

Parameter	Description
void initialize(InitializationInput initializationInput);	Used to initialize a read task. It indicates that Stream Client is about to read data of a shard.
void processRecords(ProcessRec ordsInput processRecordsInput);	Indicates how the user wants to process this batch of records after the data is read. The `getCheckpo inter` function in `ProcessRec ordsInput` can be used to obtain `IRecordPro cessorChec kpointer`. The framework provides this API to implement the checkpoint. You can determine how often the checkpoint is implemented.
void shutdown(ShutdownInput shutdownInput);	Used to end the read task of a shard.

 **Note:**

- The read tasks are implemented in different machines, the process may encounter various types of errors, for example, restart due to an environment factor. Therefore, you must periodically record the completed data (checkpoint). When a task is restarted, it is continued from the last checkpoint. In other words, Stream Client does not guarantee that a record is sent through `ProcessRec ordsInput` only once. It only guarantees that the record is sent at least once, and that the record sequence does not change. If some data is repeatedly sent, you must pay attention to the service processing logic.
- If you want to reduce the repeat data processing times in case of an error, you can increase the frequency of the checkpoint operation. However, too frequent checkpoints reduce the system throughput. Therefore, determine the checkpoint frequency based on your service features.
- If you find that the incremental data fails to be consumed in time, you can increase resources for the consumer, such as using more nodes to read the Stream record.

The following provides a simple example to describe how to use Stream Client to obtain the incremental data in real time and output the incremental data on the console.

```
public class StreamSamp le {
   class RecordProc essor implements IRecordPro cessor {

       private long creationTi me = System . currentTim
eMillis ();
       private String workerIden tifier ;

       public RecordProc essor ( String workerIden tifier ) {
           this . workerIden tifier = workerIden tifier ;
      }

       public void initialize ( Initializa tionInput
initializa tionInput ) {
          // Trace some info before start the query
like stream info etc .
      }

       public void processRec ords ( ProcessRec ordsInput
processRec ordsInput ) {
           List < StreamReco rd > records = processRec ordsInput
. getRecords ();

           if ( records . size () == 0 ) {
              // No more records we can wait for the
next query
               System . out . println (" no more records ");
```

```
          }
           for  ( int   i  =  0 ;  i  <  records . size ();  i ++) {
               System . out . println (" records :" +  records . get
( i ));
          }

          //  Since   we   don ' t   persist   the   stream   record
  we   can   skip   blow   step
           System . out . println ( processRec  ordsInput .
getCheckpo  inter (). getLargest  PermittedC  heckpointV  alue ());
           try  {
               processRec  ordsInput . getCheckpo  inter ().
checkpoint ();
          }  catch  ( ShutdownEx  ception   e ) {
               e . printStack  Trace ();
          }  catch  ( StreamClie  ntExceptio  n   e ) {
               e . printStack  Trace ();
          }  catch  ( Dependency  Exception   e ) {
               e . printStack  Trace ();
          }
      }

       public   void   shutdown ( ShutdownIn  put   shutdownIn  put )
{
          //  finish   the   query   task   and   trace   the
shutdown   reason
           System . out . println ( shutdownIn  put . getShutdow
nReason ());
      }
  }

   class   RecordProc  essorFacto  ry   implements   IRecordPro
cessorFact  ory  {

       private   final   String   workerIden  tifier ;

       public   RecordProc  essorFacto  ry ( String   workerIden
tifier ) {
           this . workerIden  tifier  =  workerIden  tifier ;
      }

       public   IRecordPro  cessor   createProc  essor () {
           return   new   StreamSamp  le . RecordProc  essor (
workerIden  tifier );
      }
  }

   public   Worker   getNewWork  er ( String   workerIden  tifier ) {
      //  Please   replace   with   your   table   info
       final   String   endPoint  = "";
       final   String   accessId  = "";
       final   String   accessKey  = "";
       final   String   instanceNa  me  = "";

       StreamConf  ig   streamConf  ig  =  new   StreamConf  ig ();
       streamConf  ig . setOTSClie  nt ( new   SyncClient ( endPoint
,  accessId ,  accessKey ,
               instanceNa  me ));
       streamConf  ig . setDataTab  leName (" teststream ");
       streamConf  ig . setStatusT  ableName (" statusTabl  e ");

       Worker   worker  =  new   Worker ( workerIden  tifier ,  new
ClientConf  ig (),  streamConf  ig ,
```

```
                new   StreamSamp  le . RecordProc  essorFacto  ry (
 workerIden  tifier ),  Executors . newCachedT  hreadPool (),  null );
        return   worker ;
   }

    public   static   void   main ( String []  args )  throws
 Interrupte  dException  {
        StreamSamp  le   test  =  new   StreamSamp  le ();
        Worker   worker1  =  test . getNewWork  er (" worker1 ");
        Thread   thread1  =  new   Thread ( worker1 );
        thread1 . start ();
   }
}
```

6 HBase

6.1 Table Store HBase Client

In addition to SDKs and RESTful APIs, Table Store HBase Client can be used to access Table Store through Java applications built on open source HBase APIs. Based on Java SDKs for Table Store version 4.2.x and later, Table Store HBase Client supports open source APIs for HBase version 1.x.x and later.

Table Store HBase Client can be obtained from any of the following three channels:

- *GitHub tablestore-hbase-client project*
- *Compressed package*
- Maven

```
< dependenci  es >
        < dependency >
            < groupId > com . aliyun . openservic  es </ groupId >
            < artifactId > tablestore - hbase - client </ artifactId
>
            < version > 1 . 2 . 0 </ version >
        </ dependency >
    </ dependenci  es >
```

Table Store is a fully managed NoSQL database service. When using TableStore HBase Client, you can simply ignore HBase Server. Instead, you only need to perform table or data operations using APIs exposed by Client.

Compared with self-built HBase services, Table Store has the following advantages:

Items	Table Store	Self-built HBase cluster
Cost	Billing is based on actual data volumes. By providing high performance and capacity instances, Table Store can be tailored to all scenarios.	Allocates resources based on traffic peaks. Resources remain idle during off-peak periods, resulting in high operation and maintenance costs.

Items	Table Store	Self-built HBase cluster
Security	Integrates Alibaba Cloud RAM and supports multiple authentication and authorization mechanisms, VPC, and primary/RAM user account management. Authorization granularity can be defined at both the table-level and API-level.	Requires extra security mechanisms.
Reliability	Supports automatic redundant data backup and failover. Data availability is 99.9% or greater, and data reliability is 99.99999999%.	Is dependent on cluster reliability.
Scalability	Server Load Balancer of Table Store supports PB-level data transfer from a single table. Manual resizing is not needed even if millions of bytes of data is concurrently stored.	Complex online/offline processes are required if a cluster reaches high usage capacity, which can severely impact online services.

6.2 Table Store HBase Client supported functions

API support differences between Table Store and HBase

Table Store and HBase, while similar in terms of *Data model* functionality, have different APIs. The following sections detail differences between Table Store HBase Client APIs and HBase APIs.

Functions supported by Table Store HBase Client APIs:

- CreateTable

  Table Store does not support ColumnFamily as all data can be considered to be in the same ColumnFamily. This means that TTL and Max Versions of Table Store

are at the table-level. Therefore, Table Store has some support for the following functions:

Functions	Supported or Not
family max version	Table-level Max Versions supported. Default value: 1
family min version	Unsupported
family ttl	Table-level TTL supported
is/set ReadOnly	Supported through the sub-account of RAM
Pre-partitioning	Unsupported
blockcache	Unsupported
blocksize	Unsupported
BloomFilter	Unsupported
column max version	Unsupported
cell ttl	Unsupported
Control parameter	Unsupported

- Put

Functions	Supported or Not
Writes multiple columns of data at a time	Supported
Specifies a timestamp	Supported
Uses the system time by default if no timestamp is specified	Supported
Single-row ACL	Unsupported
ttl	Unsupported
Cell Visibility	Unsupported
tag	Unsupported

- Get

  Table Store guarantees high data consistency. If the HTTP 200 status code (OK) is returned after data is written to an API, the data is permanently written to all copies, and can be read immediately by Get.

Functions	Supported or Not
Reads a row of data	Supported
Reads all columns in a ColumnFamily	Supported
Reads data from a specified column	Supported
Reads data with a specified timestamp	Supported
Reads data of a specified number of versions	Supported
TimeRange	Supported
ColumnfamilyTimeRange	Unsupported
RowOffsetPerColumnFamily	Supported
MaxResultsPerColumnFamily	Unsupported
checkExistenceOnly	Unsupported
closestRowBefore	Supported
attribute	Unsupported
cacheblock:true	Supported
cacheblock:false	Unsupported
IsolationLevel:READ_COMMITTED	Supported
IsolationLevel:READ_UNCOMMITTED	Unsupported
IsolationLevel:STRONG	Supported
IsolationLevel:TIMELINE	Unsupported

- Scan

  Table Store guarantees high data consistency. If the HTTP 200 status code (OK) is returned after data is written to an API, the data is permanently written to all copies, which can be read immediately by Scan.

Functions	Supported or Not
Determines a scanning range based on the specified start and stop	Supported

Functions	Supported or Not
Globally scans data if no scanning range is specified	Supported
prefix filter	Supported
Reads data using the same logic as Get	Supported
Reads data in reverse order	Supported
caching	Supported
batch	Unsupported
maxResultSize, indicating the maximum size of the returned data volume	Unsupported
small	Unsupported
batch	Unsupported
cacheblock:true	Supported
cacheblock:false	Unsupported
IsolationLevel:READ_COMMITTED	Supported
IsolationLevel:READ_UNCOMMITTED	Unsupported
IsolationLevel:STRONG	Supported
IsolationLevel:TIMELINE	Unsupported
allowPartialResults	Unsupported

- Batch

Functions	Supported or Not
Get	Supported
Put	Supported
Delete	Supported
batchCallback	Unsupported

- Delete

Functions	Supported or Not
Deletes a row	Supported
Deletes all versions of the specified column	Supported

Functions	Supported or Not
Deletes the specified version of the specified column	Supported
Deletes the specified ColumnFamily	Unsupported
When a timestamp is specified, deleteColumn deletes the versions that are equal to the timestamp	Supported
When a timestamp is specified, deleteFamily and deleteColumn delete the versions that are earlier than or equal to the timestamp	Unsupported
When no timestamp is specified, deleteColumn deletes the latest version	Unsupported
When no timestamp is specified, deleteFamily and deleteColumn delete the version of the current system time	Unsupported
addDeleteMarker	Unsupported

- checkAndXXX

Functions	Supported or Not
CheckAndPut	Supported
checkAndMutate	Supported
CheckAndDelete	Supported
Checks whether the value of a column meets the conditions. If yes, checkAndXXX deletes the column.	Supported
Uses the default value if no value is specified	Supported
Checks row A and executes row B.	Unsupported

- Exist

Functions	Supported or Not
Checks whether one or more rows exist and does not return any content	Supported

- Filter

Functions	Supported or Not
ColumnPaginationFilter	columnOffset and count unsupported
SingleColumnValueFilter	Supported: LongComparator, BinaryComparator, and ByteArrayComparable Unsupported: RegexStringComparator, SubstringComparator, and BitComparator

Functions not supported by Table Store HBase Client APIs

- Namespaces

  Table Store uses instances to manage a data table. An instance is the minimum billing unit in Table Store. You can manage instances in the *Table Store console*. Therefore, the following features are not supported:

  - createNamespace(NamespaceDescriptor descriptor)
  - deleteNamespace(String name)
  - getNamespaceDescriptor(String name)
  - listNamespaceDescriptors()
  - listTableDescriptorsByNamespace(String name)
  - listTableNamesByNamespace(String name)
  - modifyNamespace(NamespaceDescriptor descriptor)
- Region management

  Data partition is the basic unit for data storage and management in Table Store. Table Store automatically splits or merges the data partitions based on their data volumes and access conditions. Therefore, Table Store does not support features related to Region management in HBase.
- Snapshots

  Table Store does not support Snapshots, or related featurs of Snapshots.

- Table management

  Table Store automatically splits, merges, and compacts data partitions in tables. Therefore, the following features are not supported:

  - getTableDescriptor(TableName tableName)
  - compact(TableName tableName)
  - compact(TableName tableName, byte[] columnFamily)
  - flush(TableName tableName)
  - getCompactionState(TableName tableName)
  - majorCompact(TableName tableName)
  - majorCompact(TableName tableName, byte[] columnFamily)
  - modifyTable(TableName tableName, HTableDescriptor htd)
  - split(TableName tableName)
  - split(TableName tableName, byte[] splitPoint)
- Coprocessors

  Table Store does not support the coprocessor. Therefore, the following features are not supported:

  - coprocessorService()
  - coprocessorService(ServerName serverName)
  - getMasterCoprocessors()
- Distributed procedures

  Table Store does not support Distributed procedures. Therefore, the following features are not supported:

  - execProcedure(String signature, String instance, Map props)
  - execProcedureWithRet(String signature, String instance, Map props)
  - isProcedureFinished(String signature, String instance, Map props)
- Increment and Append

  Table Store does not support atomic increase/decrease or atomic Append.

6.3 Differences between Table Store and HBase

This topic introduces features of Table Store HBase Client and explains restricted and supported functions when compared with HBase. Features are listed as follows.

Table

Table Store only supports single ColumnFamilies, that is, it does not support multi-ColumnFamilies.

Row and Cell

- Table Store does not support ACL settings.
- Table Store does not support Cell Visibility settings.
- Table Store does not support Tag settings.

GET

Table Store only supports single ColumnFamilies. Therefore, it does not support ColumnFamily related APIs, including:

- setColumnFamilyTimeRange(byte[] cf, long minStamp, long maxStamp)
- setMaxResultsPerColumnFamily(int limit)
- setRowOffsetPerColumnFamily(int offset)

SCAN

Similar to GET, Table Store does not support ColumnFamily related APIs and cannot be used to set partial optimization APIs, including:

- setBatch(int batch)
- setMaxResultSize(long maxResultSize)
- setAllowPartialResults(boolean allowPartialResults)
- setLoadColumnFamiliesOnDemand(boolean value)
- setSmall(boolean small)

Batch

Table Store does not support BatchCallback.

Mutations and Deletions

- Table Store does not support deletion of the specified ColumnFamily.
- Table Store does not support deletion of the versions with the latest timestamp.
- Table Store does not support deletion of all versions earlier than the specified timestamp.

Increment and Append

Table Store does not support Increment or Append features.

Filter

- Table Store supports ColumnPaginationFilter.
- Table Store supports FilterList.
- Table Store partially supports SingleColumnValueFilter, and supports only BinaryComparator.
- Table Store does not support other Filters.

Optimization

Some of the HBase APIs involve access and storage optimization. These APIs are not opened currently:

- blockcache: The default value is "true", which cannot be modified.
- blocksize: The default value is "64 KB", which cannot be modified.
- IsolationLevel: The default value is "READ_COMMITTED", which cannot be modified.
- Consistency: The default value is "STRONG", which cannot be modified.

Admin

The `org . apache . hadoop . hbase . client . Admin` APIs of HBase are used for management and control, most of which are not required in Table Store.

As Table Store is a cloud service, it automatically performs operations such as operation and maintenance, management, and control, which does not need to be concerned. Table Store currently does not support a few of APIs.

- CreateTable

  Table Store only supports single ColumnFamilies. Therefore, you can create only one ColumnFamily when creating a table. The ColumnFamily supports the MaxVersions and TimeToLive parameters.

- Maintenance task

  In Table Store, the following APIs related to task maintenance are automatically processed:

  - abort(String why, Throwable e)
  - balancer()
  - enableCatalogJanitor(boolean enable)
  - getMasterInfoPort()
  - isCatalogJanitorEnabled()
  - rollWALWriter(ServerName serverName) -runCatalogScan()
  - setBalancerRunning(boolean on, boolean synchronous)
  - updateConfiguration(ServerName serverName)
  - updateConfiguration()
  - stopMaster()
  - shutdown()
- Namespaces

  In Table Store, the instance name is similar to Namespaces in HBase. Therefore, it does not support Namespaces related APIs, including:

  - createNamespace(NamespaceDescriptor descriptor)
  - modifyNamespace(NamespaceDescriptor descriptor)
  - getNamespaceDescriptor(String name)
  - listNamespaceDescriptors()
  - listTableDescriptorsByNamespace(String name)
  - listTableNamesByNamespace(String name)
  - deleteNamespace(String name)

- Region

  Table Store automatically performs Region related operations. Therefore, it does not support the following APIs:

  - assign(byte[] regionName)
  - closeRegion(byte[] regionname, String serverName)
  - closeRegion(ServerName sn, HRegionInfo hri)
  - closeRegion(String regionname, String serverName)
  - closeRegionWithEncodedRegionName(String encodedRegionName, String serverName)
  - compactRegion(byte[] regionName)
  - compactRegion(byte[] regionName, byte[] columnFamily)
  - compactRegionServer(ServerName sn, boolean major)
  - flushRegion(byte[] regionName)
  - getAlterStatus(byte[] tableName)
  - getAlterStatus(TableName tableName)
  - getCompactionStateForRegion(byte[] regionName)
  - getOnlineRegions(ServerName sn)
  - majorCompactRegion(byte[] regionName)
  - majorCompactRegion(byte[] regionName, byte[] columnFamily)
  - mergeRegions(byte[] encodedNameOfRegionA, byte[] encodedNameOfRegionB, boolean forcible)
  - move(byte[] encodedRegionName, byte[] destServerName)
  - offline(byte[] regionName)
  - splitRegion(byte[] regionName)
  - splitRegion(byte[] regionName, byte[] splitPoint)
  - stopRegionServer(String hostnamePort)
  - unassign(byte[] regionName, boolean force)

Snapshots

Table Store does not support Snapshots related APIs.

Replication

Table Store does not support Replication related APIs.

Coprocessors

Table Store does not support Coprocessors related APIs.

Distributed procedures

Table Store does not support Distributed procedures related APIs.

Table Management

Table Store automatically performs Table related operations, which does not need to be concerned. Therefore, Table Store does not support the following APIs:

- compact(TableName tableName)
- compact(TableName tableName, byte[] columnFamily)
- flush(TableName tableName)
- getCompactionState(TableName tableName)
- majorCompact(TableName tableName)
- majorCompact(TableName tableName, byte[] columnFamily)
- modifyTable(TableName tableName, HTableDescriptor htd)
- split(TableName tableName)
- split(TableName tableName, byte[] splitPoint)

Restrictions

As Table Store is a cloud service, to guarantee the optimal overall performance, some parameters are restricted and cannot be reconfigured. For more information about the restrictions, see *Limits*.

6.4 Migrate from HBase to Table Store

The following information explains how to migrate HBase to Table Store.

Dependencies

Table Store HBase Client v1.2.0 depends on HBase Client v1.2.0 and Table Store Java SDK v4.2.1. The configuration of `pom . xml` is as follows.

```
< dependenci  es >
       < dependency >
           < groupId > com . aliyun . openservic  es </ groupId >
           < artifactId > tablestore - hbase - client </ artifactId >
           < version > 1 . 2 . 0 </ version >
       </ dependency >
```

```
    </ dependenci  es >
```

If you want to use another HBase Client or Table Store Java SDK version, you must use the exclusion tag. In the following example, HBase Client v1.2.1 and Table Store Java SDK v4.2.0 are used.

```
  < dependenci  es >
       < dependency >
           < groupId > com . aliyun . openservic  es </ groupId >
           < artifactId > tablestore - hbase - client </ artifactId >
           < version > 1 . 2 . 0 </ version >
           < exclusions >
               < exclusion >
                   < groupId > com . aliyun . openservic  es </
groupId >
                   < artifactId > tablestore </ artifactId >
               </ exclusion >
               < exclusion >
                   < groupId > org . apache . hbase </ groupId >
                   < artifactId > hbase - client </ artifactId >
               </ exclusion >
           </ exclusions >
       </ dependency >
       < dependency >
           < groupId > org . apache . hbase </ groupId >
           < artifactId > hbase - client </ artifactId >
           < version > 1 . 2 . 1 </ version >
       </ dependency >
       < dependency >
           < groupId > com . aliyun . openservic  es </ groupId >
           < artifactId > tablestore </ artifactId >
           < classifier > jar - with - dependenci  es </ classifier >
           < version > 4 . 2 . 0 </ version >
       </ dependency >
   </ dependenci  es >
```

Table Store HBase Client v1.2.x is only compatible with HBase Client v1.2.x, because API changes exist in HBase Client v1.2.x and earlier.

If you want to use HBase Client version v1.1.x, use Table Store HBase Client version v1.1.x.

If you want to use HBase Client version v0.x.x, see *Migrate HBase of an earlier version*.

Configure the file

To migrate data from HBase Client to Table Store HBase Client, modify the following two items in the configuration file.

- HBase Connection type

  Set Connection to TableStoreConnection.

```
    < property >
        < name > hbase . client . connection . impl </ name >
```

```
        < value > com . alicloud . tablestore . hbase . Tablestore
Connection </ value >
   </ property >
```

- Configuration items of Table Store

  Table Store is a cloud service and provides strict permission management. Table Store offers strict permission management. To access Table Store, you must configure access information such as the AccessKey.

  - You need to configure the following four items before accessing Table Store:

```
< property >
  < name > tablestore . client . endpoint </ name >
  < value ></ value >
 </ property >
 < property >
  < name > tablestore . client . instancena  me </ name >
  < value ></ value >
 </ property >
 < property >
  < name > tablestore . client . accesskeyi  d </ name >
  < value ></ value >
 </ property >
 < property >
  < name > tablestore . client . accesskeys  ecret </ name >
  < value ></ value >
 </ property >
```

  - Optional items you can configure are as follows.

```
 < property >
  < name > hbase . client . tablestore . family </ name >
  < value > f1 </ value >
 </ property >
 < property >
  < name > hbase . client . tablestore . family .$ tablename </
name >
  < value > f2 </ value >
 </ property >
 < property >
  < name > tablestore . client . max . connection  s </ name >
  < value > 300 </ value >
 </ property >
 < property >
  < name > tablestore . client . socket . timeout </ name >
  < value > 15000 </ value >
 </ property >
 < property >
  < name > tablestore . client . connection . timeout </ name >
  < value > 15000 </ value >
 </ property >
 < property >
  < name > tablestore . client . operation . timeout </ name >
  < value > 2147483647 </ value >
 </ property >
 < property >
  < name > tablestore . client . retries </ name >
  < value > 3 </ value >
```

```
  </ property >
```

- hbase.client.tablestore.family and hbase.client.tablestore.family.$tablename
  - Table Store only supports single ColumnFamilies. When you use HBase APIs, you must enter the content of the family.

    `hbase . client . tablestore . family` indicates global configuration, while `hbase . client . tablestore . family .$ tablename` indicates configuration of a single table.
  - Rule: For tables whose names are T, search for `hbase . client . tablestore . family . T` first. If the family does not exist, search for `hbase . client . tablestore . family` . If the family does not exist, use the default value f.
- tablestore.client.max.connections

  Maximum connections. The default value is 300.
- tablestore.client.socket.timeout

  Socket time-out time. The default value is 15 seconds.
- tablestore.client.connection.timeout

  Connection time-out time. The default value is 15 seconds.
- tablestore.client.operation.timeout

  API time-out time. The default value is Integer.MAX_VALUE, indicating that the API never times out.
- tablestore.client.retries

  Number of retries when a request fails. The default value is 3.

6.5 Migrate HBase of an earlier version

Table Store HBase Client supports APIs of HBase Client 1.0.0 and later versions.

Compared with earlier versions, HBase Client 1.0.0 has big changes which are incompatible with HBase Client of earlier versions.

If you use an HBase Client from version 0.x.x (that is, an earlier version than 1.0.0), this topic explains how to integrate your HBase Client version with Table Store.

Connection APIs

HBase 1.0.0 and later versions cancel the HConnection APIs, and instead use the `org . apache . hadoop . hbase . client . Connection Factory` series to provide the Connection APIs and replace ConnectionManager and HConnectionManager with ConnectionFactory.

Creating a Connection API has relatively high cost, however, Connection APIs guarantee thread safety. When using a Connection API, you can generate only one Connection object in the program. Multiple threads can then share this object.

You also need to manage the Connection lifecycle, and close it after use.

The latest code is as follows:

```
 Connection   connection  =  Connection  Factory . createConn  ection
 ( config );
// ...
 connection . close ();
```

TableName series

In HBase version 1.0.0 and earlier, you can use a String-type name when creating a table. For later HBase versions, you can use the `org . apache . hadoop . hbase . TableName` .

The latest code is as follows:

```
 String   tableName  = " MyTable ";
//  or   byte []  tableName  =  Bytes . toBytes (" MyTable ");
 TableName   tableNameO  bj  =  TableName . valueOf ( tableName );
```

Table, BufferedMutator, and RegionLocator APIs

From HBase Client v1.0.0, the HTable APIs are replaced with the Table, BufferedMu tator, and RegionLocator APIs.

- `org . apache . hadoop . hbase . client . Table` : Used to operate reading, writing, and other requests of a single table.
- `org . apache . hadoop . hbase . client . BufferedMu tator` : Used for asynchronous batch writing. This API corresponds to `setAutoFlu sh ( boolean )` of the HTableInterface API of the earlier versions.
- `org . apache . hadoop . hbase . client . RegionLoca tor` : Indicates the table partition information.

The Table, BufferedMutator, and RegionLocator APIs do not guarantee thread safety. However, they are lightweight and can be used to create an object for each thread.

Admin APIs

From HBase Client v1.0.0, HBaseAdmin APIs are replaced by `org . apache . hadoop . hbase . client . Admin`. As Table Store is a cloud service, and most operation and maintenance APIs are automatically processed, most Admin APIs are not supported. For more information, see *Differences between Table Store and HBase*.

Use the Connection instance to create an Admin instance:

```
Admin admin = connection . getAdmin ();
```

6.6 Hello World

This topic describes how to use Table Store HBase Client to implement a simple Hello World program, and includes the following operations:

- Configure project dependencies.
- Connect Table Store
- Create a table
- Write Data
- Read Data
- Scan data
- Delete a table

Code position

This sample program uses HBase APIs to access Table Store. The complete sample program is located in the *Github aliyun-tablestore-hbase-client* project. The directory is src/test/java/samples/HelloWorld.java.

Use HBase APIs

- Configure project dependencies

  Configure Maven dependencies as follows.

```
< dependenci es >
    < dependency >
        < groupId > com . aliyun . openservic es </ groupId >
        < artifactId > tablestore - hbase - client </ artifactId
>
        < version > 1 . 2 . 0 </ version >
```

```
        </ dependency >
    </ dependenci  es >
```

For more information about advanced configurations, see *Migrate from HBase to Table Store*.

- Configure the file

  Add the following configuration items to hbase-site.xml.

```
< configurat  ion >
    < property >
        < name > hbase . client . connection . impl </ name >
        < value > com . alicloud . tablestore . hbase . Tablestore
 Connection </ value >
    </ property >
    < property >
        < name > tablestore . client . endpoint </ name >
        < value > endpoint </ value >
    </ property >
    < property >
        < name > tablestore . client . instancena  me </ name >
        < value > instance_n  ame </ value >
    </ property >
    < property >
        < name > tablestore . client . accesskeyi  d </ name >
        < value > access_key  _id </ value >
    </ property >
    < property >
        < name > tablestore . client . accesskeys  ecret </ name >
        < value > access_key  _secret </ value >
    </ property >
    < property >
        < name > hbase . client . tablestore . family </ name >
        < value > f1 </ value >
    </ property >
    < property >
        < name > hbase . client . tablestore . table </ name >
        < value > ots_adapto  r </ value >
    </ property >
</ configurat  ion >
```

For more information about advanced configurations, see *Migrate from HBase to Table Store*.

- Connect Table Store

  Create a TableStoreConnection object to connect Table Store.

```
   Configurat  ion   config  =  HBaseConfi  guration . create ();

  //  Create   a   Tablestore   Connection
   Connection   connection  =  Connection  Factory . createConn
ection ( config );

  //  Admin   is   used   for   creation ,  management ,  and
deletion
```

```
        Admin admin = connection . getAdmin ();
```

- Create a table

  Create a table using the specified table name. Use the default table name for MaxVersions and TimeToLive.

```
 // Create an HTableDesc riptor , which contains only
one ColumnFami ly
  HTableDesc riptor descriptor = new HTableDesc riptor (
TableName . valueOf ( TABLE_NAME ));

 // Create a ColumnFami ly . Use the default
ColumnFami ly name for Max Versions and TimeToLive .
The default ColumnFami ly name for Max Versions is
1 and for TimeToLive is Integer . INF_MAX
  descriptor . addFamily ( new HColumnDes criptor ( COLUMN_FAM
ILY_NAME ));

 // Use the createTabl e API of the Admin to
create a table
  System . out . println (" Create table " + descriptor .
getNameAsS tring ());
  admin . createTabl e ( descriptor );
```

- Write Data

  Write a row of data to Table Store.

```
 // Create a Tablestore Table for reading , writing ,
updating , deletion , and other operations on a single
  table
  Table table = connection . getTable ( TableName . valueOf (
TABLE_NAME ));

 // Create a Put object with the primary key
row_1
  System . out . println (" Write one row to the table
");
        Put put = new Put ( ROW_KEY );

 // Add a column . Table Store supports only single
ColumnFami lies . The ColumnFami ly name is configured
in hbase - site . xml . If the ColumnFami ly name is
not configured , the default name is " f ". In this
case , the value of COLUMN_FAM ILY_NAME may be null
when data is written .
  put . addColumn ( COLUMN_FAM ILY_NAME , COLUMN_NAM E ,
COLUMN_VAL UE );

 // Run put for Table , and use HBase APIs to
write the row of data to Table Store
        table . put ( put );
```

- Read Data

  Read data of the specified row.

```
 // Create a Get object to read the row whose
primary key is ROW_KEY .
```

```
    Result   getResult  =  table . get ( new   Get ( ROW_KEY ));
    Result   result  =  table . get ( get );

   //  Print   the   results
    String   value  =  Bytes . toString ( getResult . getValue (
COLUMN_FAM  ILY_NAME ,  COLUMN_NAM  E ));
    System . out . println (" Get   one   row   by   row   key ");
    System . out . printf ("\ t % s  = % s \ n ",  Bytes . toString (
ROW_KEY ),  value );
```

- **Scan data**

  Read data in the specified range.

```
   Scan   data   of   all   rows   in   the   table
   System . out . println (" Scan   for   all   rows :");
   Scan   scan  =  new   Scan ();

   ResultScan  ner   scanner  =  table . getScanner ( scan );

  //  Print   the   results   cyclically
   for  ( Result   row  :  scanner ) {
    byte []  valueBytes  =  row . getValue ( COLUMN_FAM  ILY_NAME ,
COLUMN_NAM  E );
    System . out . println ('\ t ' +  Bytes . toString ( valueBytes
));
  }
```

- **Delete a table**

  Use Admin APIs to delete a table.

```
    print (" Delete   the   table ");
    admin . disableTab  le ( table . getName ());
    admin . deleteTabl  e ( table . getName ());
```

Complete code

```
package   samples ;

import   org . apache . hadoop . conf . Configurat  ion ;
import   org . apache . hadoop . hbase . HBaseConfi  guration ;
import   org . apache . hadoop . hbase . HColumnDes  criptor ;
import   org . apache . hadoop . hbase . HTableDesc  riptor ;
import   org . apache . hadoop . hbase . TableName ;
import   org . apache . hadoop . hbase . client .*;
import   org . apache . hadoop . hbase . util . Bytes ;

import   java . io . IOExceptio  n ;

public   class   HelloWorld  {

    private   static   final   byte []  TABLE_NAME  =  Bytes .
toBytes (" HelloTable  store ");
    private   static   final   byte []  ROW_KEY  =  Bytes . toBytes
(" row_1 ");
    private   static   final   byte []  COLUMN_FAM  ILY_NAME  =
Bytes . toBytes (" f ");
    private   static   final   byte []  COLUMN_NAM  E  =  Bytes .
toBytes (" col_1 ");
```

```
    private   static   final   byte []  COLUMN_VAL  UE  =  Bytes .
toBytes (" col_value ");

    public   static   void   main ( String []  args ) {
        helloWorld ();
   }

    private   static   void   helloWorld () {

        try   {
            Configurat  ion   config  =  HBaseConfi  guration .
create ();
            Connection   connection  =  Connection  Factory .
createConn  ection ( config );
            Admin   admin  =  connection . getAdmin ();

            HTableDesc  riptor   descriptor  =  new   HTableDesc
riptor ( TableName . valueOf ( TABLE_NAME ));
            descriptor . addFamily ( new   HColumnDes  criptor (
COLUMN_FAM  ILY_NAME ));

            System . out . println (" Create   table  " +  descriptor
. getNameAsS  tring ());
            admin . createTabl  e ( descriptor );

            Table   table  =  connection . getTable ( TableName .
valueOf ( TABLE_NAME ));

            System . out . println (" Write   one   row   to   the
table ");
            Put   put  =  new   Put ( ROW_KEY );
            put . addColumn ( COLUMN_FAM  ILY_NAME ,  COLUMN_NAM  E
,  COLUMN_VAL  UE );
            table . put ( put );

            Result   getResult  =  table . get ( new   Get ( ROW_KEY
));
            String   value  =  Bytes . toString ( getResult .
getValue ( COLUMN_FAM  ILY_NAME ,  COLUMN_NAM  E ));
            System . out . println (" Get   a   one   row   by   row
  key ");
            System . out . printf ("\ t % s  = % s \ n ",  Bytes .
toString ( ROW_KEY ),  value );

            Scan   scan  =  new   Scan ();

            System . out . println (" Scan   for   all   rows :");
            ResultScan  ner   scanner  =  table . getScanner ( scan
);
            for  ( Result   row  :  scanner ) {
                byte []  valueBytes  =  row . getValue ( COLUMN_FAM
ILY_NAME ,  COLUMN_NAM  E );
                System . out . println ('\ t ' +  Bytes . toString (
valueBytes ));
            }

            System . out . println (" Delete   the   table ");
            admin . disableTab  le ( table . getName ());
            admin . deleteTabl  e ( table . getName ());

            table . close ();
            admin . close ();
            connection . close ();
       }  catch  ( IOExceptio  n   e ) {
```

```
            System . err . println (" Exception   while   running
 HelloTable  store : " +  e . toString ());
            System . exit ( 1 );
        }
    }
}
```

7 SearchIndex

7.1 Features

Core features

Query based on non-primary key columns

The original table only supports the query based on complete primary key columns or prefixes of primary key columns. The query based on non-primary key columns is not available in some scenarios. However, Search Index supports the query by non-primary key columns. You can create a Search Index structure for the required column and search data by using the value of the column.

Bool query

The bool query of Search Index is applicable to order scenarios. In these scenarios, a table may contain dozens of fields. You cannot determine the combination of fields that you want to query when you create a table. Even if you determine the combination of required fields, hundreds of combinations may be available. If you use a relational database, you need to create hundreds of indexes. Also, if you miss a combination method, you cannot query the corresponding data.

However, by using Table Store, you only need to create a Search Index structure that includes the possibly required field names. Therefore, you can freely combine these fields in a query. Search Index also supports multiple logical operators, such as AND, OR, and NOT.

Geographic location-based query

With the popularization of mobile devices, geographic location data is more and more important. The data is increasingly used in most applications, such as WeChat Moments, Weibo, food delivery, sports, and Internet of Vehicles (IoV). These applications provide geographic location data, so they must support query features.

Search Index supports the query based on geographic location data as follows:

- **Near: queries points within a specified radius based on an origin, such as the People Nearby feature in Discover of WeChat.**
- **Within: specifies a rectangle or polygon area to query points within this area.**

Based on these query features, you can use Table Store to easily query geographic location data, and do not need other databases or search systems.

Full-text search

Search Index can tokenize data and support full-text search. The system tokenizes data in two ways: single word and max word.

 Note:
Table Store does not support relevance searches.

- Single-word tokenization: the system tokenizes Chinese characters by words, and English words by spaces. English words are case-insensitive. Letters are not segmented from numbers if the letters are spliced with the numbers. For example , the system tokenizes "Mercedes-Benz E300L" into three words: mercedes, benz, and e300l.
- Max-word tokenization: the system tokenizes sentences into as many semantic words or meaningful words as possible.
  - For example, the system tokenizes "Mercedes-Benz E300L" into these words: mercedes-benz, mercedes, benz, e300l, e, 300, and l.

The tokenization involves two factors: data expansion rate and recall rate. A finer data granularity results in a higher recall rate and a higher data expansion rate. As a result, the corresponding query occupies more space.

Fuzzy query

Search Index provides the query based on wildcards. This feature is similar to the LIKE operator in relational databases. You can specify characters and wildcards such as question marks (`?`) or asterisks (`*`) to query data in the same way as the LIKE operator.

Prefix query

Search Index provides the prefix query. This feature is applicable to Chinese, English , and other languages. For example, in the query based on the prefix "apple", Table Store may return words such as "apple6s" and "applexr".

Nested query

In addition to a flat structure, online data such as pictures and labels have some complex multi-layered structures. For example, a database stores a large number

of pictures, and each picture has multiple elements, such as houses, cars, or people . Each element in a picture has a unique score. The score is evaluated according to the size and position of an element in a picture. Therefore, each picture has multiple labels. Each label has a name and a weighted score. You can use the nested query based on the conditions or field names of the labels.

The following example shows the JSON data format in a query:

```
{
 " tags ": [
   {
      " name ": " car ",
      " score ":  0 . 78
   },
   {
      " name ": " tree ",
      " score ":  0 . 24
   }
 ]
}
```

By using the nested query, you can effectively store and query data of multi-layered logical relationships. This query facilitates the modeling of complex data.

Cardinality

Search Index supports the cardinality feature for query results. The cardinality allows you to specify the highest frequency of occurrence of an attribute value to achieve high cardinality. For example, when you search for a laptop on an e-commerce platform, the first page may display the computers of a certain brand. This is not a user-friendly search. However, the cardinality feature of Table Store can avoid this issue.

Sorting

A table alphabetically sorts data based on primary keys. To sort data by other fields , you need to use the sorting feature of Search Index. Table Store supports multiple types of sorting, such as forward sorting, reverse sorting, single-field sorting, and multi-field sorting, so you can easily sort global data. By default, the system returns results in the order of primary keys in the table. You can use this method to sort global data.

Total number of rows

You can specify the number of rows that the system returns for the current request when you use Search Index. If you do not specify any query condition for Search

Index, the system returns the total number of rows where you have created indexes . When you stop writing new data to a table and create indexes on all attributes, the system returns the total number of rows in the table. This feature applies to data verification and data management.

SQL

Table Store does not support SQL statements and operators. But most of these SQL features can match similar features of Search Index, as shown in the following table.

SQL	Search Index	Supported
SHOW	API: DescribeSearchIndex	Yes
SELECT	Parameter: ColumnsToGet	Yes
FROM	Parameter: index name	Supported in a single index and not supported in multiple indexes
WHERE	Query: a variety of queries such as TermQuery	Yes
ORDER BY	Parameter: sort	Yes
LIMIT	Parameter: limit	Yes
DELETE	API: Query followed by DeleteRow	Yes
LIKE	Query: wildcard query	Yes
AND	Parameter: operator = and	Yes
OR	Parameter: operator = or	Yes
NOT	Query: bool query	Yes
BETWEEN	Query: range query	Yes
NULL	ExistQuery	Yes

7.2 API operations

7.2.1 Overview

SDKs

You can use the following SDKs to enable the Search Index feature.

- *Java SDK*

- *Python SDK*
- *Go SDK*
- *Node.js SDK*
- *.NET SDK*

API operations

Name	API operation	Description
Create	*CreateSearchIndex*	Create a Search Index structure.
Describe	*DescribeSearchIndex*	Obtain the details of a Search Index structure.
List	*ListSearchIndex*	Retrieve a list of Search Index structures.
Delete	*DeleteSearchIndex*	Delete a specified Search Index structure.
Search	*Search*	Search for required data.

Fields

The value of a Search Index field in Table Store comes from the value of the field of the same name in the corresponding table. The types of these fields must match each other as described in the following table.

Field type in the Search Index structure	Field type in the table	Description
Long	Integer	64-bit long integer.
Double	Double	64-bit long floating-point number.
Boolean	Boolean	Boolean value.
Keyword	String	Character string that can be tokenized.
Text	String	Character string or text that can be tokenized.

Field type in the Search Index structure	Field type in the table	Description
Geopoint	String	Coordinates of a geographic point in the format of "`latitude , longitude`". In this field, the former value indicates the latitude and the latter value indicates the longitude, such as "35.8,-45.91".
Nested	String	Nested type, such as "{["a": 1], ["a": 3]}".

Notice:

The types in this table must correspond to each other. Otherwise, Table Store discards the data as dirty data, especially when GeoPoint and Nested types describe data in special formats. If the formats do not match, Table Store also discards the data as dirty data. As a result, the data may be available in the table, but be unavailable in the Search Index structure.

In addition to the type of a Search Index field, you can also specify the attribute of the field:

Attribute	Type	Name	Description
Index	Boolean	Specifies whether Table Store creates an index for a column.	A value of true specifies that Table Store creates an inverted index or spatial index for the column. A value of false specifies that Table Store does not create any index for the column. If no index exists, you cannot search data by the column.

Attribute	Type	Name	Description
EnableSortAndAgg	Boolean	Specifies whether Table Store enables sorting and aggregation.	A value of true specifies that you can sort data by the column. A value of false specifies that you cannot sort data by the column.
Store	Boolean	Specifies whether Table Store stores original values in the index.	A value of true specifies that Table Store stores original values in the column to the index. Therefore, Table Store reads values of the column directly from the index, rather than from the primary table. This optimizes search performance.
IsArray	Boolean	Specifies whether the column is an array.	A value of true specifies that the column is an array. Data written to the column must be a JSON array, such as ["a","b","c"]. A Nested column is an array, and does not require the Array attribute.

 Note:

For more information about the differences between the Array type and the Nested type, see *Comparison between the Nested type and the Array type*.

For more information about the attributes that each field type support, see the following table.

Type	Index	EnableSort AndAgg	Store	Array
Long	Yes	Yes	Yes	Yes
Double	Yes	Yes	Yes	Yes
Boolean	Yes	Yes	Yes	Yes
Keyword	Yes	Yes	Yes	Yes
Text	Yes	No	Yes	Yes
Geopoint	Yes	Yes	Yes	Yes
Nested	Required for child fields.	Required for child fields.	Required for child fields.	Nested fields are arrays.

Query parameters and types

You must specify SearchRequest in a query. SearchRequest includes parameters as described in the following table.

Parameter	Type	Description
offset	Integer	The position that the current query starts from.
limit	Integer	The maximum number of items that the current query returns.
getTotalCount	Boolean	Specifies whether to return the total number of matched rows. This parameter is set to false by default. A value of true may affect the query performance.
Sort	Sort	Specifies the field and method for sorting.
collapse	Collapse	Specifies the name of the field that you want to collapse in the query result .
query	Query	The type of the current query. The following table lists the query types.

Query	Name	Description
MatchAllQuery	Table Store matches all rows in a query.	You can use MatchAllQuery to check the total number of rows.
MatchQuery	Table Store matches required data in a query.	Table Store tokenizes the query data, and queries the tokenized data. Logical operator OR applies to the tokens.
MatchPhraseQuery	Table Store matches required phrases in a query.	This query is similar to MatchQuery. But the matched target tokens must be adjacent to each other in the query data.
TermQuery	Table Store exactly matches a required term in a query.	You can use TermQuery to exactly match strings. Table Store uses exact matches to query data in a table, and does not tokenize the query data.
TermsQuery	Table Store matches multiple terms in a query.	This query is similar to TermQuery, but supports multiple terms. This query is also similar to the SQL `IN` operator.
PrefixQuery	Table Store matches a prefix in a query.	You can use PrefixQuery to query data in a table by matching a required prefix.
RangeQuery	Table Store matches targets within a range in a query.	You can use RangeQuery to query data within a specified range in a table.
WildcardQuery	Table Store matches targets that contain one or more wildcard characters.	You can use WildcardQuery to query data based on a string that contains one or more wildcard characters. This query is similar to the SQL LIKE operator.

Query	Name	Description
BoolQuery	Table Store combines multiple query conditions in a query.	You can use BoolQuery to combine multiple query conditions by using Logical operators AND, OR, and NOT.
GeoBoundingBoxQuery	Table Store matches targets within a geographic rectangular area in a query.	You can use GeoBoundingBoxQuery to specify a geographic rectangular area as a condition for a query. Table Store returns the rows where the value of a field falls within the geographic rectangular area.
GeoDistanceQuery	Table Store matches targets within an area by using a central point and a radius in a query.	You can use GeoDistanceQuery to specify a geographic circular area as a condition for a query. This circle is set of all points at a specified distance, radius, from the central point. Table Store returns the rows where the value of a field falls within the geographic circular area.
GeoPolygonQuery	Table Store matches targets within a geographic polygon area in a query.	You can use GeoPolygonQuery to specify a geographic polygon area as a condition for a query. Table Store returns the rows where the value of a field falls within the geographic polygon area.

Billing

For more information, see *Billing items and pricing*.

7.2.2 CreateSearchIndex

You can call this operation to create a Search Index structure. To use the Search Index feature for a table, you must create a Search Index structure in the table. One table can contain multiple Search Index structures.

Description

Parameters:

- TableName: the name of the target table where you want to create a Search Index structure.
- IndexName: the name of the target Search Index structure.
- IndexSchema: defines the schema of the target Search Index structure.
  - IndexSetting
    - RoutingFields: specifies the routing fields. You can specify some primary key columns as routing fields. Table Store distributes data that is written to a Search Index structure to different partitions based on the specified routing fields.
  - FieldSchemas
    - FieldName: the required name of the field of String type. This field name must be a column name in the table.
    - FieldType: the required type of the field.
    - Index: specifies whether to create an index for the field. This is an optional parameter of Boolean type. Default value: true.
    - IndexOptions: specifies whether to store terms such as position and offset to an inverted chain. You can use the default value of the optional parameter.
    - Analyzer: the type of the tokenizer. This is an optional parameter. Valid values: single_word and max_word.
    - EnableSortAndAgg: specifies whether to enable sorting and aggregation. This is an optional parameter of Boolean type. Default value: true.
    - Store: specifies whether Table Store stores original values in the index to accelerate queries. This is an optional parameter of Boolean type. Default value: true.

Example

```
/**
```

```
 * Create   a   Search   Index   structure   that   contains   the
 Col_Keywor  d   and   Col_Long   columns .  Set   the   type   of
 data   in   Col_Keywor  d   to   KEYWORD .  Set   the   type   of
 data   in   Col_Long   to   LONG .
 */
 private   static   void   createSear  chIndex ( SyncClient   client )
 {
     CreateSear  chIndexReq  uest   request  =  new   CreateSear
 chIndexReq  uest ();
     request . setTableNa  me ( TABLE_NAME ); //  Set   the   name
 of   the   table .
     request . setIndexNa  me ( INDEX_NAME ); //  Set   the   name
 of   the   index .
     IndexSchem  a   indexSchem  a  =  new   IndexSchem  a ();
     indexSchem  a . setFieldSc  hemas ( Arrays . asList (
             new   FieldSchem  a (" Col_Keywor  d ",  FieldType .
 KEYWORD ) //  Set   the   field   name   and   field   type .
                 . setIndex ( true ) //  Set   the   parameter   to
   true   to   enable   indexing .
                 . setEnableS  ortAndAgg ( true ), //  Set   the
 parameter   to   true   to   enable   sorting   and   aggregatio  n .
             new   FieldSchem  a (" Col_Long ",  FieldType . LONG )
                 . setIndex ( true )
                 . setEnableS  ortAndAgg ( true )));
     request . setIndexSc  hema ( indexSchem  a );
     client . createSear  chIndex ( request ); //  Use   client   to
 create   a   Search   Index   structure .
}
```

7.2.3 DescribeSearchIndex

You can call this operation to query the details of a Search Index structure. To use the Search Index feature for a table, you must create a Search Index structure in the table. One table can contain multiple Search Index structures.

Description

Name: DescribeSearchIndex

Parameters:

- **TableName: the name of the target table where you request the details of the Search Index structure.**
- **IndexName: the name of the target index.**

Example

```
 private   static   DescribeSe  archIndexR  esponse   describeSe
 archIndex ( SyncClient   client ) {
     DescribeSe  archIndexR  equest   request  =  new   DescribeSe
 archIndexR  equest ();
     request . setTableNa  me ( TABLE_NAME ); //  Set   the   name
 of   the   table .
     request . setIndexNa  me ( INDEX_NAME ); //  Set   the   name
 of   the   index .
     DescribeSe  archIndexR  esponse   response  =  client .
 describeSe  archIndex ( request );
```

```
    System . out . println ( response . jsonize ()); //  Display
 the   details   of   the   response .
    return   response ;
}
```

7.2.4 ListSearchIndex

You can call this operation to retrieve the list of all Search Index structures associated with an instance or a table.

Description

Name: ListSearchIndex

Parameter:

- TableName: the name of the target table. If you do not specify this optional parameter, Table Store returns the list of all indexes on the instance. If you specify a table, Table Store returns the list of all Search Index structures associated with the table.

Example

```
 private   static   List < SearchInde  xInfo >  listSearch  Index (
 SyncClient   client ) {
    ListSearch  IndexReque  st   request  =  new   ListSearch
 IndexReque  st ();
    request . setTableNa  me ( TABLE_NAME ); //  Set   the   name
 of   the   table .
    return   client . listSearch  Index ( request ). getIndexIn
 fos (); //  Return   all   Search   Index   structures   of   the
 specified   table .
}
```

7.2.5 DeleteSearchIndex

You can call this operation to delete a Search Index structure.

Description

Name: DeleteSearchIndex

Parameters:

- TableName: the name of the target table where you delete the Search Index structure.
- IndexName: the name of the target index that you want to delete.

Example

```
 private   static   void   deleteSear  chIndex ( SyncClient   client )
 {
```

```
    DeleteSear  chIndexReq  uest   request  =  new   DeleteSear
 chIndexReq  uest ();
    request . setTableNa  me ( TABLE_NAME ); //  Set   the   name
 of   the   table .
    request . setIndexNa  me ( INDEX_NAME ); //  Set   the   name
 of   the   index .
    client . deleteSear  chIndex ( request ); //  Use   client   to
 delete   the   target   Search   Index   structure .
}
```

7.2.6 Nested type

A Nested column contains nested documents. One document or one row can contain multiple child documents, and these child documents are saved to the same Nested column. You need to specify the structure of child documents in the Nested column. The structure includes the fields of the child documents and the element of each field. The following example defines the format of a Nested column in Java.

```
//  Specify   the   FieldSchem  a   class   for   the   child
 documents .
 List < FieldSchem  a >  subFieldSc  hemas  =  new   ArrayList <
 FieldSchem  a >();
 subFieldSc  hemas . add ( new   FieldSchem  a (" tagName ",
 FieldType . KEYWORD )
    . setIndex ( true ). setEnableS  ortAndAgg ( true ));
 subFieldSc  hemas . add ( new   FieldSchem  a (" score ",  FieldType
 . DOUBLE )
    . setIndex ( true ). setEnableS  ortAndAgg ( true ));

//  Set   FieldSchem  a   of   the   child   documents   as
 subfieldSc  hemas   of   the   NESTED   column .
 FieldSchem  a   nestedFiel  dSchema  =  new   FieldSchem  a (" tags
 ",  FieldType . NESTED )
    . setSubFiel  dSchemas ( subFieldSc  hemas );
```

This example defines the format of a Nested column named tags. The child documents include two fields: one is a KEYWORD field named tagName and the other is a DOUBLE field named score.

Table Store writes Nested columns as strings or JSON arrays to the primary table. The following example shows the data format of a Nested column.

```
[{" tagName ":" tag1 ", " score ": 0 . 8 }, {" tagName ":" tag2 ", "
 score ": 0 . 2 }]
```

This column contains two child documents. Even if a column contains only one child document, you must provide the strings as JSON arrays.

The Nested type has the following restrictions:

- Nested indexes do not support the IndexSort feature. But IndexSort can improve query performance in many scenarios.

- The Nested query provides lower performance than other types of queries.

7.2.7 Sort

You can use Sort to specify the method of sorting the result when you call the Search operation to search indexes.

The Search Index feature supports multiple sorting methods.

If you have not specified the sorting method for the search, the system applies the IndexSort parameter for the required indexes. By default, Table Store returns the query result in the order of primary key columns.

Table Store supports the following sorting methods:

- ScoreSort

  Sort the result by relevance score. ScoreSort is applicable to relevance scenarios such as full-text indexing.
- PrimaryKeySort

  Sort the result by the value of a primary key.
- FieldSort

  Sort the result by the value of a specified field.
- GeoDistanceSort

  Sort the result by the distance, radius, from a central point.

7.2.8 MatchAllQuery

You can use MatchAllQuery to query the total number of rows or any number of rows in a table.

Example

```
/**
 *  Use   MatchAllQu  ery   to   query   the   total   number   of
 rows   in   a   table .
 * @ param   client
 */
 private   static   void   matchAllQu  ery ( SyncClient   client ) {
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();

    /**
    *  Set   the   query   type   to   MatchAllQu  ery .
    */
     searchQuer  y . setQuery ( new   MatchAllQu  ery ());

    /**
```

```
     * In the MatchAllQu ery - based query result , the
 value of TotalCount is the total number of rows in
   a table . This value is an approximat e value when
   you query a table that contains a large number of
   rows .
     * To return only the total number of rows without
   any specific data , you can set Limit to 0 . Then
 , Table Store returns no data in the rows .
     */
     searchQuer y . setLimit ( 0 );
     SearchRequ est searchRequ est = new SearchRequ est (
 TABLE_NAME , INDEX_NAME , searchQuer y );

    /**
    * Set the total number of matched rows .
    */
     searchRequ est . setGetTota lCount ( true );
     SearchResp onse resp = client . search ( searchRequ est );
    /**
     * Check whether Table Store returns matched data
   from all partitions . When the value of isAllSucce
 ss is false , Table Store may fail to query some
 partitions and return a part of data .
     */
     if (! resp . isAllSucce ss ()) {
         System . out . println (" NotAllSucc ess !") ;
    }
     System . out . println (" IsAllSucce ss : " + resp .
 isAllSucce ss ());
     System . out . println (" TotalCount : " + resp . getTotalCo
 unt ()); // The total number of rows .
     System . out . println ( resp . getRequest Id ());
}
```

7.2.9 MatchQuery

You can use MatchQuery to query data in the fields of Text type in full-text search scenarios. Table Store tokenizes the value of Text type in the index and the target value that you specify for the MatchQuery type based on your configuration. Therefore, Table Store can match tokenized terms in a query.

For example, the title field value in a row is "Hangzhou West Lake Scenic Area". Table Store tokenizes the value into "Hangzhou", "West", "Lake", "Scenic", and "Area". If you specify the target term as "Lake Scenic" in MatchQuery, Table Store returns this row in the query result.

Parameters

- fieldName: the name of the target field.
- text: the target term. Table Store tokenizes this term into multiple terms.

- minimumShouldMatch: the minimum number of terms that the value of the fieldName field in a row contains when Table Store returns this row in the query result.
- operator: the operator used in a logical operation. The default operator OR specifies that Table Store returns the row when some of the tokens of the field value in the row match the target term. The operator AND specifies that Table Store returns the row only when all tokens of the field value in the row match the target term.

Example

```
/**
 *  Search   the   table   for   rows   where   the   value   of
 Col_Keywor  d   matches  " hangzhou ".  Table   Store   returns
 matched   rows   and   the   total   number   of   matched   rows .
 * @ param   client
 */
 private   static   void   matchQuery ( SyncClient   client ) {
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();
     MatchQuery   matchQuery  =  new   MatchQuery (); //  Set   the
 query   type   to   MatchQuery .
     matchQuery . setFieldNa  me (" Col_Keywor  d "); //  Set   the
 name   of   the   field   that   you   want   to   match .
     matchQuery . setText (" hangzhou "); //  Set   the   value   that
   you   want   to   match .
     searchQuer  y . setQuery ( matchQuery );
     searchQuer  y . setOffset ( 0 ); //  Set   Offset   to   0 .
     searchQuer  y . setLimit ( 20 ); //  Set   Limit   to   20   to
 return   20   rows   or   fewer .
     SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );
     SearchResp  onse   resp  =  client . search ( searchRequ  est );
     System . out . println (" TotalCount : " +  resp . getTotalCo
 unt ());
     System . out . println (" Row : " +  resp . getRows ()); //  If
   you   do   not   set   columnsToG  et ,  Table   Store   only
 returns   primary   keys   by   default .

     SearchRequ  est . ColumnsToG  et   columnsToG  et  =  new
 SearchRequ  est . ColumnsToG  et ();
     columnsToG  et . setReturnA  ll ( true ); //  Set   columnsToG
 et   to   true   to   return   all   columns .
     searchRequ  est . setColumns  ToGet ( columnsToG  et );

     resp  =  client . search ( searchRequ  est );
     System . out . println (" TotalCount : " +  resp . getTotalCo
 unt ()); //  The   total   number   of   matched   rows   instead
 of   the   number   of   returned   rows .
     System . out . println (" Row : " +  resp . getRows ());
```

```
}
```

7.2.10 MatchPhraseQuery

This query is similar to MatchQuery, but evaluates the positional relationship between multiple tokens. Table Store exactly matches the order and position of these tokens in the target row.

For example, the field value is "Hangzhou West Lake Scenic Area". If you specify the target term as "Hangzhou Scenic Area" in Query, Table Store returns the row that contains this target term when you use MatchQuery. However, when you use MatchPhraseQuery, Table Store does not return the row that contains this target term . The distance between "Hangzhou" and "Scenic Area" in Query is 0. But the distance in the field is 2, because the two words "West" and "Lake" exist between "Hangzhou" and "Scenic Area".

Parameters

- fieldName: the name of the target field.
- text: the target term. Table Store tokenizes this term into multiple terms before the query.

Example

```
/**
 *  Search   the   table   for   rows   where   the   value   of
 Col_Text   matches   " hangzhou   shanghai ."  Table   Store   returns
   the   total   number   of   rows   that   match   the   phrase   as
   a   whole   and   matched   rows   in   this   query .
 * @ param   client
 */
 private   static   void   matchPhras  eQuery ( SyncClient   client )
 {
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();
     MatchPhras  eQuery   matchPhras  eQuery  =  new   MatchPhras
 eQuery (); //  Set   the   query   type   to   MatchPhras  eQuery .
     matchPhras  eQuery . setFieldNa  me (" Col_Text "); //  Set   the
   field   that   you   want   to   match .
     matchPhras  eQuery . setText (" hangzhou   shanghai "); //  Set
 the   value   that   you   want   to   match .
     searchQuer  y . setQuery ( matchPhras  eQuery );
     searchQuer  y . setOffset ( 0 ); //  Set   Offset   to   0 .
     searchQuer  y . setLimit ( 20 ); //  Set   Limit   to   20   to
 return   20   rows   or   fewer .
     SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );
     SearchResp  onse   resp  =  client . search ( searchRequ  est );
     System . out . println (" TotalCount : " +  resp . getTotalCo
 unt ());
     System . out . println (" Row : " +  resp . getRows ()); //
 Return   primary   keys   only   by   default .
```

```
    SearchRequ est . ColumnsToG et columnsToG et = new
SearchRequ est . ColumnsToG et ();
    columnsToG et . setReturnA ll ( true ); // Set columnsToG
et to true to return all columns .
    searchRequ est . setColumns ToGet ( columnsToG et );

    resp = client . search ( searchRequ est );
    System . out . println (" TotalCount : " + resp . getTotalCo
unt ()); // The total number of matched rows instead
of the number of returned rows .
    System . out . println (" Row : " + resp . getRows ());
}
```

7.2.11 TermQuery

You can use TermQuery to query data that exactly matches the specified value of a field. When a table contains a Text string, Table Store tokenizes the string and exactly matches any of the tokens. For example, Table Store tokenizes Text string "tablestore is cool" into "tablestore," "is," and "cool". When you specify any of these tokens as a query string, you can retrieve the query result that contains the token.

Parameters

- **fieldName: the name of the target field.**
- **term: the target term. Table Store does not tokenize this term, but exactly matches the whole term.**

Example

```
/**
 * Search the table for rows where the value of
Col_Keywor d exactly matches " hangzhou ".
 * @ param client
 */
private static void termQuery ( SyncClient client ) {
    SearchQuer y searchQuer y = new SearchQuer y ();
    TermQuery termQuery = new TermQuery (); // Set the
query type to TermQuery .
    termQuery . setFieldNa me (" Col_Keywor d "); // Set the
name of the field that you want to match .
    termQuery . setTerm ( ColumnValu e . fromString (" hangzhou
")); // Set the value that you want to match .
    searchQuer y . setQuery ( termQuery );
    SearchRequ est searchRequ est = new SearchRequ est (
TABLE_NAME , INDEX_NAME , searchQuer y );

    SearchRequ est . ColumnsToG et columnsToG et = new
SearchRequ est . ColumnsToG et ();
    columnsToG et . setReturnA ll ( true ); // Set columnsToG
et to true to return all columns .
    searchRequ est . setColumns ToGet ( columnsToG et );

    SearchResp onse resp = client . search ( searchRequ est );
    System . out . println (" TotalCount : " + resp . getTotalCo
unt ()); // The total number of matched rows instead
of the number of returned rows .
```

```
    System . out . println (" Row : " +  resp . getRows ());
}
```

7.2.12 TermsQuery

This query is similar to TermQuery, but supports multiple terms. This query is also similar to the SQL IN operator.

Parameters

fieldName: the name of the target field.

terms: the target terms. Table Store returns the data in a row when the system matches one term in the row.

Example

```
/**
 *  Search   the   table   for   rows   where   the   value   of
 Col_Keywor  d   exactly   matches  " hangzhou "  or  " xi ' an ".
 * @ param   client
 */
 private   static   void   termQuery ( SyncClient   client ) {
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();
     TermsQuery   termsQuery  =  new   TermsQuery (); //  Set   the
 query   type   to   TermsQuery .
     termsQuery . setFieldNa  me (" Col_Keywor  d "); //  Set   the
 name   of   the   field   that   you   want   to   match .
     termsQuery . addTerm ( ColumnValu  e . fromString (" hangzhou
 ")); //  Set   the   value   that   you   want   to   match .
     termsQuery . addTerm ( ColumnValu  e . fromString (" xi ' an
 ")); //  Set   the   value   that   you   want   to   match .
     searchQuer  y . setQuery ( termsQuery );
     SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );

     SearchRequ  est . ColumnsToG  et   columnsToG  et  =  new
 SearchRequ  est . ColumnsToG  et ();
     columnsToG  et . setReturnA  ll ( true ); //  Set   columnsToG
 et   to   true   to   return   all   columns .
     searchRequ  est . setColumns  ToGet ( columnsToG  et );

     SearchResp  onse   resp  =  client . search ( searchRequ  est );
     System . out . println (" TotalCount : " +  resp . getTotalCo
 unt ()); //  The   total   number   of   matched   rows   instead
 of   the   number   of   returned   rows .
     System . out . println (" Row : " +  resp . getRows ());
```

```
}
```

7.2.13 PrefixQuery

You can use PrefixQuery to query data that matches a specified prefix. When a table contains a TEXT string, Table Store tokenizes the string and matches any of the tokens with the specified prefix.

Parameters

- fieldName: the name of the target field.
- prefix: the value of the specified prefix.

Example

```
/**
 *  Search   the   table   for   rows   where   the   value   of
 Col_Keywor  d   contains   the   prefix   that   exactly   matches  "
 hangzhou ".
 * @ param   client
 */
 private   static   void   prefixQuer  y ( SyncClient   client ) {
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();
     PrefixQuer  y   prefixQuer  y  =  new   PrefixQuer  y (); //  Set
   the   query   type   to   PrefixQuer  y .
     prefixQuer  y . setFieldNa  me (" Col_Keywor  d ");
     prefixQuer  y . setPrefix (" hangzhou ");
     searchQuer  y . setQuery ( prefixQuer  y );
     SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );

     SearchRequ  est . ColumnsToG  et   columnsToG  et  =  new
 SearchRequ  est . ColumnsToG  et ();
     columnsToG  et . setReturnA  ll ( true ); //  Set   columnsToG
 et   to   true   to   return   all   columns .
     searchRequ  est . setColumns  ToGet ( columnsToG  et );

     SearchResp  onse   resp  =  client . search ( searchRequ  est );
     System . out . println (" TotalCount : " +  resp . getTotalCo
 unt ()); //  The   total   number   of   matched   rows   instead
 of   the   number   of   returned   rows .
     System . out . println (" Row : " +  resp . getRows ());
```

7.2.14 RangeQuery

You can use RangeQuery to query data that falls within a specified range. When a table contains a TEXT string, Table Store tokenizes the string and matches any of the tokens that falls within the specified range.

Parameters

- fieldName: the name of the target field.
- from: the value of the start position.

- to: the value of the end position.
- includeLow: specifies whether the query result includes the value of the from parameter. This is a parameter of Boolean type.
- includeUpper: specifies whether the query result includes the value of the to parameter. This is a parameter of Boolean type.

Example

```
/**
 * Search the table for rows where the value of
Col_Long is greater than 3 . Table Store sorts these
rows by Col_Long in descending order .
* @ param client
*/
private static void rangeQuery ( SyncClient client ) {
    SearchQuer y searchQuer y = new SearchQuer y ();
    RangeQuery rangeQuery = new RangeQuery (); // Set the
query type to RangeQuery .
    rangeQuery . setFieldNa me (" Col_Long "); // Set the name
  of the target field .
    rangeQuery . greaterTha n ( ColumnValu e . fromLong ( 3 ));
 // Specify the range of the value of the field .
The required value is larger than 3 .
    searchQuer y . setQuery ( rangeQuery );

   // Sort the result by Col_Long in descending order
.
    FieldSort fieldSort = new FieldSort (" Col_Long ");
    fieldSort . setOrder ( SortOrder . DESC );
    searchQuer y . setSort ( new Sort ( Arrays . asList (( Sort .
Sorter ) fieldSort )));

    SearchRequ est searchRequ est = new SearchRequ est (
TABLE_NAME , INDEX_NAME , searchQuer y );

    SearchResp onse resp = client . search ( searchRequ est );
    System . out . println (" TotalCount : " + resp . getTotalCo
unt ()); // The total number of matched rows instead
of the number of returned rows .
    System . out . println (" Row : " + resp . getRows ());

   /**
    * You can specify a value for SearchAfte r to
  start a new query . For example , you can set
SearchAfte r to 5 and sort the result by Col_Long
in descending order . Then , you retrieve the rows that
  follow the row whose Col_Long is equal to 5 . This
  is similar to the method where you specify that
the value of Col_Long is smaller than 5 .
    */
    searchQuer y . setSearchA fter ( new SearchAfte r ( Arrays .
asList ( ColumnValu e . fromLong ( 5 ))));
    searchRequ est = new SearchRequ est ( TABLE_NAME ,
INDEX_NAME , searchQuer y );
    resp = client . search ( searchRequ est );
    System . out . println (" TotalCount : " + resp . getTotalCo
unt ()); // The total number of matched rows instead
of the number of returned rows .
    System . out . println (" Row : " + resp . getRows ());
```

```
}
```

7.2.15 WildcardQuery

You can use WildcardQuery to query data that matches wildcard characters. You can specify a value you want to match as a string that consists of one or more wildcard characters. An asterisk (*) is interpreted as a number of characters or an empty string. A question mark (?) is interpreted as any single character. For example, when you search the string "table*e", you can retrieve query results such as "tablestore".

Parameters

- fieldName: the name of the target field.
- value: the value that contains one or more wildcard characters. Table Store supports two types of wildcard characters: asterisk (*) and question mark (?). The value cannot start with an asterisk (*) and the length of the value can be 10 bytes or less.

Example

```
/**
 *  Search   the   table   for   rows   where   the   value   of
 Col_Keywor  d   matches  " hang * u ".
 * @ param   client
 */
 private   static   void   wildcardQu  ery ( SyncClient   client ) {
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();
     WildcardQu  ery   wildcardQu  ery  =  new   WildcardQu  ery
 (); //  Set   the   query   type   to   WildcardQu  ery .
     wildcardQu  ery . setFieldNa  me (" Col_Keywor  d ");
     wildcardQu  ery . setValue (" hang * u "); // Specify   a
 string   that   contains   one   or   more   wildcard   characters
 in   wildcardQu  ery .
     searchQuer  y . setQuery ( wildcardQu  ery );
     SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );

     SearchRequ  est . ColumnsToG  et   columnsToG  et  =  new
 SearchRequ  est . ColumnsToG  et ();
     columnsToG  et . setReturnA  ll ( true ); //  Set   columnsToG
 et   to   true   to   return   all   columns .
     searchRequ  est . setColumns  ToGet ( columnsToG  et );

     SearchResp  onse   resp  =  client . search ( searchRequ  est );

     System . out . println (" TotalCount : " +  resp . getTotalCo
 unt ()); //  The   total   number   of   matched   rows   instead
 of   the   number   of   returned   rows .
     System . out . println (" Row : " +  resp . getRows ());
```

```
}
```

7.2.16 BoolQuery

You can use BoolQuery to query data based on a combination of filtering conditions. This query contains one or more subqueries as filtering conditions. Table Store returns the rows that match the subqueries.

You can combine these subqueries in different ways. If you specify these subqueries as mustQueries, Table Store returns the result that matches all these subqueries. If you specify these subqueries as mustNotQueries, Table Store returns the result that matches none of these subqueries.

Parameters

- mustQueries: specifies the subqueries that the query result must match. This is a list of multiple fields of Query type.
- mustNotQueries: specifies the subqueries that the query result must not match. This is a list of multiple fields of Query type.
- shouldQueries: specifies the subqueries that the query result can or cannot match. If the query result match the subqueries, the overall relevance score is higher.
- minimumShouldMatch: specifies the minimum number of shouldQueries subqueries that the query result must match.

Example

```
/**
 * Use BoolQuery to query data that matches a
 combinatio n of conditions .
 * @ param client
 */
 public static void boolQuery ( SyncClient client ) {
    /**
     * Condition 1 : use RangeQuery to query data where
   the value of Col_Long is greater than 3 .
     */
     RangeQuery rangeQuery = new RangeQuery ();
     rangeQuery . setFieldNa me (" Col_Long ");
     rangeQuery . greaterTha n ( ColumnValu e . fromLong ( 3 ));

    /**
     * Condition 2 : use MatchQuery to query data where
   the value of Col_Keywor d matches " hangzhou ".
     */
     MatchQuery matchQuery = new MatchQuery (); // Set the
 query type to MatchQuery .
     matchQuery . setFieldNa me (" Col_Keywor d "); // Set the
 name of the field that you want to match .
     matchQuery . setText (" hangzhou "); // Set the value that
   you want to match .
```

```
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();
    {
        /**
         *  Create   a   query   of   BoolQuery   type   where   the
 result   meets   Conditions   1   and   2   at   the   same   time .
         */
         BoolQuery   boolQuery  =  new   BoolQuery ();
         boolQuery . setMustQue  ries ( Arrays . asList ( rangeQuery
 ,  matchQuery ));
         searchQuer  y . setQuery ( boolQuery );
         SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );
         SearchResp  onse   resp  =  client . search ( searchRequ  est
 );
         System . out . println (" TotalCount : " +  resp .
 getTotalCo  unt ()); //  The   total   number   of   matched   rows
 instead   of   the   number   of   returned   rows .
         System . out . println (" Row : " +  resp . getRows ());
    }

    {
        /**
         *  Create   a   query   of   BoolQuery   type   where   the
 result   meets   at   least   one   of   Conditions   1   and   2 .
         */
         BoolQuery   boolQuery  =  new   BoolQuery ();
         boolQuery . setShouldQ  ueries ( Arrays . asList ( rangeQuery
 ,  matchQuery ));
         boolQuery . setMinimum  ShouldMatc  h ( 1 ); //  Specify
   that   the   result   meets   at   least   one   of   the
 conditions .
         searchQuer  y . setQuery ( boolQuery );
         SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );
         SearchResp  onse   resp  =  client . search ( searchRequ  est
 );
         System . out . println (" TotalCount : " +  resp .
 getTotalCo  unt ()); //  The   total   number   of   matched   rows
 instead   of   the   number   of   returned   rows .
         System . out . println (" Row : " +  resp . getRows ());
    }
}
```

7.2.17 GeoDistanceQuery

You can use GeoDistanceQuery to query data that falls within a distance from a central point. You can specify the central point and the distance from this central point in the query. Table Store returns the rows where the value of a field falls within the distance from the central point.

Parameters

- **fieldName: the name of the target field.**
- **centerPoint: the central coordinate point that consists of latitude and longitude values.**

- distanceInMeter: the distance from the central point. This is a value of Double type . Unit: meters.

Example

```
/**
 *  Search   the   table   for   rows   where   the   value   of
 Col_GeoPoi  nt   falls   within   a   specified   distance   from   a
   specified   central   point .
 * @ param   client
 */
 public   static   void   geoDistanc  eQuery ( SyncClient   client ) {
     SearchQuer  y   searchQuer  y  =  new   SearchQuer  y ();
     GeoDistanc  eQuery   geoDistanc  eQuery  =  new   GeoDistanc
 eQuery ();  //  Set   the   query   type   to   GeoDistanc  eQuery .
     geoDistanc  eQuery . setFieldNa  me (" Col_GeoPoi  nt ");
     geoDistanc  eQuery . setCenterP  oint (" 5 , 5 "); //  Specify
 coordinate  s   for   a   central   point .
     geoDistanc  eQuery . setDistanc  eInMeter ( 10000 ); //  You
 can   specify   10 , 000   meters   or   less   as   the   distance
 from   the   central   point .
     searchQuer  y . setQuery ( geoDistanc  eQuery );

     SearchRequ  est   searchRequ  est  =  new   SearchRequ  est (
 TABLE_NAME ,  INDEX_NAME ,  searchQuer  y );

     SearchRequ  est . ColumnsToG  et   columnsToG  et  =  new
 SearchRequ  est . ColumnsToG  et ();
     columnsToG  et . setColumns ( Arrays . asList (" Col_GeoPoi  nt
 "));  // Specify   Col_GeoPoi  nt   as   the   column   that   you
 want   to   return .
     searchRequ  est . setColumns  ToGet ( columnsToG  et );

     SearchResp  onse   resp  =  client . search ( searchRequ  est );
     System . out . println (" TotalCount : " +  resp . getTotalCo
 unt ()); //  The   total   number   of   matched   rows   instead
 of   the   number   of   returned   rows .
     System . out . println (" Row : " +  resp . getRows ());
}
```

7.2.18 GeoBoundingBoxQuery

You can use GeoBoundingBoxQuery to query data that falls within a geographic rectangular area. You can specify the geographic rectangular area as a filtering condition in the query. Table Store returns the rows where the value of a field falls within the geographic rectangular area.

Parameters

- fieldName: the name of the target field.
- topLeft: coordinates of the upper-left corner of the geographic rectangular area.
- bottomRight: coordinates in the lower-right corner of the geographic rectangula r area. You can use the upper-left corner and lower-right corner to determine a unique geographic rectangular area.

Example

```
/**
 * The data type of Col_GeoPoi nt is Geopoint . You
 can obtain the rows where the value of Col_GeoPoi nt
   falls within a geographic rectangula r area . For the
   geographic rectangula r area , the upper - left vertex
 is " 10 , 0 " and the lower - right vertex is " 0 , 10 ".
 * @ param client
 */
 public static void geoBoundin gBoxQuery ( SyncClient client
 ) {
     SearchQuer y searchQuer y = new SearchQuer y ();
     GeoBoundin gBoxQuery geoBoundin gBoxQuery = new
 GeoBoundin gBoxQuery (); // Set the query type to
 GeoBoundin gBoxQuery .
     geoBoundin gBoxQuery . setFieldNa me (" Col_GeoPoi nt "); //
 Set the name of the field that you want to match
 .
     geoBoundin gBoxQuery . setTopLeft (" 10 , 0 "); // Specify
   coordinate s for the upper - left vertex of the
 geographic rectangula r area .
     geoBoundin gBoxQuery . setBottomR ight (" 0 , 10 "); //
 Specify coordinate s for the lower - right vertex of
 the geographic rectangula r area .
     searchQuer y . setQuery ( geoBoundin gBoxQuery );

     SearchRequ est searchRequ est = new SearchRequ est (
 TABLE_NAME , INDEX_NAME , searchQuer y );

     SearchRequ est . ColumnsToG et columnsToG et = new
 SearchRequ est . ColumnsToG et ();
     columnsToG et . setColumns ( Arrays . asList (" Col_GeoPoi nt
 ")); // Specify Col_GeoPoi nt as the column that you
 want to return .
     searchRequ est . setColumns ToGet ( columnsToG et );

     SearchResp onse resp = client . search ( searchRequ est );
     System . out . println (" TotalCount : " + resp . getTotalCo
 unt ()); // The total number of matched rows instead
 of the number of returned rows .
     System . out . println (" Row : " + resp . getRows ());
}
```

7.2.19 GeoPolygonQuery

You can use GeoPolygonQuery to query data that falls within a geographic polygon area. You can specify the geographic polygon area as a filtering condition in the query. Table Store returns the rows where the value of a field falls within the geographic polygon area.

Parameters

- **fieldName: the name of the target field.**
- **points: the coordinate points that compose the geographic polygon.**

Example

```
/**
 * Search the table for rows where the value of
   Col_GeoPoi nt falls within a specified geographic
 polygon area .
 * @ param client
 */
 public static void geoPolygon Query ( SyncClient client ) {
     SearchQuer y searchQuer y = new SearchQuer y ();
     GeoPolygon Query geoPolygon Query = new GeoPolygon Query
 (); // Set the query type to GeoPolygon Query .
     geoPolygon Query . setFieldNa me (" Col_GeoPoi nt ");
     geoPolygon Query . setPoints ( Arrays . asList (" 0 , 0 ," 5 ,
 5 ," 5 , 0 ")); // Specify coordinate s for vertices of
 the geographic polygon .
     searchQuer y . setQuery ( geoPolygon Query );

     SearchRequ est searchRequ est = new SearchRequ est (
 TABLE_NAME , INDEX_NAME , searchQuer y );

     SearchRequ est . ColumnsToG et columnsToG et = new
 SearchRequ est . ColumnsToG et ();
     columnsToG et . setColumns ( Arrays . asList (" Col_GeoPoi nt
 ")); // Specify Col_GeoPoi nt as the column that you
 want to return .
     searchRequ est . setColumns ToGet ( columnsToG et );

     SearchResp onse resp = client . search ( searchRequ est );
     System . out . println (" TotalCount : " + resp . getTotalCo
 unt ()); // The total number of matched rows instead
 of the number of returned rows .
     System . out . println (" Row : " + resp . getRows ());
}
```

7.3 Limits

Mapping

Item	Maximum value	Description
Index fields	200	The number of indexed fields.
EnableSortAndAgg fields	100	The number of fields that support sorting and aggregation.
Nested levels	1	The maximum number of nested levels.
Nested fields	25	The number of nested fields.

Item	Maximum value	Description
Total length of primary key columns	1,000	The total length of all primary key columns is 1,000 bytes or less.
String length in primary key columns	1,000	The total length of all primary key columns is 1,000 bytes or less.
String length in each attribute column (Keyword index)	4 KB	None.
String length in each attribute column (Text index)	2 MB	The same as the length of each attribute column.
Query field length in a wildcard query	10	The field of Query type contains 10 characters or fewer.

Search

Item	Maximum value	Description
OFFSET + LIMIT	2,000	To display more than the specified number of rows on each page, specify the next_token parameter.
timeout	10s	None.
Capacity Unit (CU)	100,000	Scanning and analysis requests do not apply. For the parameter out of this limit, submit a ticket.

Index

Item	Maximum value	Description
Rate	50,000 rows /s	· Table Store requires load balancing when writing data to a table for the first time or in an instantaneous and high-concurrency manner. · For the parameter out of this limit, submit a ticket.

Item	Maximum value	Description
Synchronization latency	10s	· The value is less than 10s when the writing rate is steady. · Table Store synchronizes 99% of data in less than one minute. · A new index requires initialization for up to one minute.
Number of rows	10 billion	For the parameter out of this limit, submit a ticket.
Total size	10 TB	For the parameter out of this limit, submit a ticket.

Other limits

Item	Value
Applicable regions	China (Beijing), China (Shanghai), China (Hangzhou), China (Shenzhen), Singapore, India (Mumbai), Hong Kong, and China (Zhangjiakou-Beijing Winter Olympics).

Note:

If these limits are not applicable to your services, submit a ticket. Describe the scenario, restriction item, requirement, and reason in the ticket. Then, your requirement is considered in follow-up development.

8 Global secondary indexes

8.1 Overview

Before you use the Global Secondary Index structure, you need to understand the following terms, limits, and notes.

Terms

Term	Description
Index	You can create an index for some columns in a primary table. The index is read-only.
Pre-defined column	Table Store uses a schema-free model. You can write the unlimited number of columns to a row. You do not need to specify a fixed number of attributes in a schema. You can also pre-define some columns and specify their data types when you create a table.
Single-column index	You can create an index only for one column.
Composite index	You can create an index for multiple columns in a table. A composite index can have Indexed columns 1 and 2.
Indexed attribute column	You can map pre-defined columns in a primary table to non-primary key columns in an index.
Autocomplete	Table Store automatically adds the primary key column that you have not specified in a primary table to an index when you create the index.

Limits

- You can create a maximum of five indexes in a primary table. You cannot create more indexes if you have created five indexes.

- An index contains a maximum of four indexed columns. These indexed columns include a combination of primary keys and pre-defined columns of the primary table. If you specify more indexed columns, you cannot create the index.
- An index contains a maximum of eight attribute columns. If you specify more attribute columns, you cannot create the index.
- You can specify an indexed column as Integer, String, or Binary type. The constraint of Indexed columns is the same as that for primary keys of the primary table.
- If an index combines multiple columns, the size limit for the index is the same as that for primary keys of the primary table.
- When you specify the column of String or Binary type as an attribute column of an index, the limits for the attribute column are the same as those for the attribute column of the primary table.
- You cannot create an index in a table that has Time To Live (TTL) configured. If you want to index a table that has TTL configured, use DingTalk to request technical support.
- You cannot create an index in a table that has Max Versions configured. If a table has Max Versions configured, you cannot create any index for the table. If you index the table, you cannot use the Max Versions feature.
- You cannot customize versions when writing data to an indexed primary table. Otherwise, you cannot write data to the primary table.
- You cannot use the Stream feature in an index.
- An indexed primary table cannot contain repeated rows that have the same primary key during the same BatchWrite operation. Otherwise, you cannot write data to the primary table.

Notes

- Table Store automatically adds the primary key column that you have not specified to the index. When you scan an index, you must specify the range of primary key columns. The range can be from negative infinity to positive infinity. For example, a primary table includes Primary keys `PK0` and `PK1` and Pre-defined column `Defined0`.

  When you create an index for the `Defined0` column, Table Store generates an index that has Primary keys `Defined0`, `PK0`, and `PK1`. When you create an

index for the `Defined0` and `PK1` columns, Table Store generates an index that has Primary keys `Defined0`, `PK1`, and `PK0`. When you create an index for the `primary key` columns, Table Store generates an index that has Primary keys `PK1` and `PK0`. When you create an index, you can only specify the column that you want to index. Table Store automatically adds the target columns to the index. For example, a primary table contains Primary keys PK0 and PK1 and Pre-defined column Defined0.

- When you create an index for the Defined0 column, Table Store generates the index that has Primary keys Defined0, PK0, and PK1.
- When you create an index for the PK1 column, Table Store generates the index that has Primary keys PK1 and PK0.

· You can specify pre-defined columns as attribute columns in the primary table. When you specify a pre-defined attribute as an indexed attribute column, you can search this index for the attribute value instead of searching the primary table. However, this increases storage costs. If you do not specify a pre-defined attribute as an indexed attribute column, you have to search the primary table. You can choose between these methods based on your requirements.

· We recommend that you do not specify a column related to the time or date as the first primary key column of an index. This type of column may slow down index updates. We recommend that you hash the column related to the time or date and create an index for the hashed column. To solve related issues, use DingTalk to request technical support.

· We recommend that you do not define an attribute of low cardinality, even an attribute that contains enumerated values, as the first primary key column of an index. For example, the `gender` attribute restricts the horizontal scalability of the index and leads to poor write performance.

8.2 Introduction

A global secondary index in Table Store has the following features:

· Supports asynchronous data synchronization between a table and table indexes. Under normal network conditions, the data synchronization latency is in milliseconds.

- Supports single-field indexes, compound indexes, and covered indexes. Pre-defined attributes are attributes specified in advance in a table. You can create an index on any pre-defined attribute or on a table primary key. In addition, you can specify a table pre-defined attributes as index attributes or choose not to specify attributes. If you specify pre-defined attributes as the index attributes, you can directly query this index to retrieve data from the base table instead of querying the table. For example, a base table includes three primary keys PK0, PK1, and PK2. Additionally, the table have three pre-defined attributes Defined0, Defined1, and Defined2.
  - You can create an index on PK2 without specifying an attribute.
  - You can create an index on PK2 and specify Defined0 as an attribute.
  - You can create an index on PK3 and PK2 without specifying an attribute.
  - You can create an index on PK3 and PK2 and specify Defined0 as an attribute.
  - You can create an index on PK2, PK1, and PK3 and specify Defined0, Defined1, and Defined2 as an attribute.
  - You can create an index on Defined0 without specifying an attribute.
  - You can create an index on Define0 and PK1 and specify Defined1 as an attribute .
  - You can create an index on Define1 and Define0 without specifying an attribute.
  - You can create an index on Define1 and Define0 and specify Defined2 as an attribute.
- Supports sparse indexes. You can specify a base table pre-defined attribute as an index attribute. This row will be indexed even when all primary keys exist despite the pre-defined attribute being excluded from the base table row. However, this row will not be indexed when a row excludes one or more indexed attributes. For example, a base table includes three primary keys that are PK0, PK1, and PK2. Additionally, the table have three pre-defined attributes Defined0, Defined1, and Defined2. You can create an index on Defined0 and Defined1, and specify Defined2 as an attribute.
  - An index will include a row in a base table that excludes the Defined2 attribute and includes pre-defined attributes Defined0 and Defined1.
  - This row is excluded from the index when a base table row excludes Defined1 but includes the pre-defined attributes Defined0 and Defined2.

- Supports creating and deleting indexes on an existing base table. In later versions , existing data in a base table will be copied to an index when you create this index on the base table.
- When you query an index, the query is not automatically performed on the base table of the created index. You need to query the base table. This feature will be supported in later versions.

The Table Store global secondary index feature is now available in China (Zhangjiako u) region. You can contact Table Store technical support by DingTalk for a trial or enter the ID 111789671 to join the DingTalk group for further information.

8.3 Scenarios

The global secondary index is a new Table Store feature. When you create a table , the primary index is composed of all the primary keys. Table Store uses primary keys to uniquely identify each row in a table. However, you need to query a table by attributes, primary keys, or primary keys that are not from the first column in more scenarios. Due to insufficient indexes, you can only fetch the results by scanning the entire table and setting filter conditions. If you obtain few results after querying a table with large data volume, the query can cause excessive consumption of resources .

The Table Store Global secondary index feature is similar to that of *DynamoDB GSI* and *HBase Phoenix*. You can create an index with one or more specified attributes. In addition, you can sort data in the created index by specified attributes. Every data you write to a base table will be asynchronously synchronized to the created index on the base table. You only have to write data to a base table, and can query indexes created on this base table. This configuration greatly improves query performance in most scenarios. For example, you can create a base table for a common phone log query as follows:

CellNumber	StartTime (Unix timestamps)	CalledNumber	Duration	BaseStationNumber
123456	1532574644	654321	60	1
234567	1532574714	765432	10	1
234567	1532574734	123456	20	3

CellNumber	StartTime (Unix timestamps)	CalledNumber	Duration	BaseStatio nNumber
345678	1532574795	123456	5	2
345678	1532574861	123456	100	2
456789	1532584054	345678	200	3

- `CellNumber` and `StartTime` are primary keys that represent `a calling number` and `the start time of a call`, respectively.
- `CalledNumb er`, `Duration`, and `BaseStatio nNumber` are pre-defined attributes that represent `a called number`, `call duration`, and `the base station number`.

When you end a phone call, the call information is written to this table. You can create global secondary indexes on `CalledNumb er` and `BaseStatio nNumber` respectively to meet various query requirements. For more information about how to create an index, see example in *Appendix*.

If you have the following query requirements:

- You want to fetch the rows where the CellNumber value matches `234567`.

  You can sort data by primary keys in Table Store. In addition, you can call the `getRange` method to scan data sequentially. When you call the `getRange` method, you need to specify `234567` both as the minimum and maximum values for PK0 (CellNumber). Meanwhile, you need to specify `0` as the minimum value of PK1 (StartTime) and specify `INT_MAX` as the maximum value of PK1. Then you can query the base table.

```
 private   static   void   getRangeFr  omMainTabl  e ( SyncClient
 client ,  long   cellNumber )
{
     RangeRowQu  eryCriteri  a   rangeRowQu  eryCriteri  a  =  new
 RangeRowQu  eryCriteri  a ( TABLE_NAME );

    //  You   can   specify   primary   keys .
     PrimaryKey  Builder   startPrima  ryKeyBuild  er  =
 PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
     startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
 PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Value . fromLong ( cellNumber
 ));
     startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
 PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Value . fromLong ( 0 ));
```

```
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

   // You can specify primary keys .
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . fromLong ( cellNumber ));
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX );
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());

    rangeRowQu eryCriteri a . setMaxVers ions ( 1 );

    String strNum = String . format ("% d ", cellNumber );
    System . out . println (" A cell number " + strNum + "
makes the following calls :");
    while ( true ) {
        GetRangeRe sponse getRangeRe sponse = client .
getRange ( new GetRangeRe quest ( rangeRowQu eryCriteri a ));
        for ( Row row : getRangeRe sponse . getRows ()) {
            System . out . println ( row );
        }

        // If the value of nextStartP rimaryKey is not
  null , you can continue to read data from the
base table .
        if ( getRangeRe sponse . getNextSta rtPrimaryK ey () !
= null ) {
            rangeRowQu eryCriteri a . setInclusi veStartPri
maryKey ( getRangeRe sponse . getNextSta rtPrimaryK ey ());
        } else {
            break ;
        }
    }
}
```

- If you want to fetch the rows where the value of CalledNumber is `123456`.

  Table Store sorts all rows by primary keys. Because CalledNumber is a pre-defined attribute, you cannot directly query a table by this attribute. Therefore, you can query an index that is created on `CalledNumb er`.

  `IndexOnBeC alledNumbe r`:

PK0	PK1	PK2
CalledNumber	CellNumber	StartTime
123456	234567	1532574734
123456	345678	1532574795
123456	345678	1532574861
654321	123456	1532574644
765432	234567	1532574714

PK0	PK1	PK2
345678	456789	1532584054

Note:

Table Store will auto complement primary keys of an index. When building this index, Table Store adds all primary keys of a base table to an index created on this base table. Therefore, the index includes three primary keys.

Because `IndexOnBeC alledNumbe r` is an index that is created on CalledNumber, you can directly query this index to fetch results.

```
private static void getRangeFr omIndexTab le ( SyncClient
client , long cellNumber ) {
    RangeRowQu eryCriteri a rangeRowQu eryCriteri a = new
RangeRowQu eryCriteri a ( INDEX0_NAM E );

   // You can specify primary keys .
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_1 , PrimaryKey Value . fromLong ( cellNumber
));
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN );
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . INF_MAX );
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

   // You can specify primary keys .
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_1 , PrimaryKey Value . fromLong ( cellNumber ));
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . INF_MAX );
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX );
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());

    rangeRowQu eryCriteri a . setMaxVers ions ( 1 );

    String strNum = String . format ("% d ", cellNumber );
    System . out . println (" A cell number " + strNum + "
was called by the following numbers ");
    while ( true ) {
        GetRangeRe sponse getRangeRe sponse = client .
getRange ( new GetRangeRe quest ( rangeRowQu eryCriteri a ));
        for ( Row row : getRangeRe sponse . getRows ()) {
            System . out . println ( row );
       }

       // If the value of nextStartP rimaryKey is not
  null , you can continue to read data from the
base table .
```

```
        if ( getRangeRe sponse . getNextSta rtPrimaryK ey () !
= null ) {
            rangeRowQu eryCriteri a . setInclusi veStartPri
maryKey ( getRangeRe sponse . getNextSta rtPrimaryK ey ());
        } else {
            break ;
        }
    }
}
```

- If you want to fetch the rows where the value of BaseStationNumber matches `002` and the value of StartTime matches `1532574740` .

  This query specifies both `BaseStatio nNumber` and `StartTime` as conditions. Therefore, you can create a compound index on the `BaseStatio nNumber` and `StartTime` .

  `IndexOnBas eStation1` :

PK0	PK1	PK2
BaseStationNumber	StartTime	CellNumber
001	1532574644	123456
001	1532574714	234567
002	1532574795	345678
002	1532574861	345678
003	1532574734	234567
003	1532584054	456789

  You can query the `IndexOnBas eStation1` index.

```
private static void getRangeFr omIndexTab le ( SyncClient
client ,
                                        long baseStatio
nNumber ,
                                        long startTime ) {
    RangeRowQu eryCriteri a rangeRowQu eryCriteri a = new
RangeRowQu eryCriteri a ( INDEX1_NAM E );

   // You can specify primary keys .
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_3 , PrimaryKey Value . fromLong ( baseStatio
nNumber ));
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . fromLong ( startTime
));
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN );
```

```
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

   // You can specify primary keys .
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_3 , PrimaryKey Value . fromLong ( baseStatio nNumber ));
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX );
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . INF_MAX );
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());

    rangeRowQu eryCriteri a . setMaxVers ions ( 1 );

    String strBaseSta tionNum = String . format ("% d ",
baseStatio nNumber );
    String strStartTi me = String . format ("% d ", startTime
);
    System . out . println (" All called numbers forwarded
  by the base station " + strBaseSta tionNum + " that
start from " + strStartTi me + " are listed as follows
:");
    while ( true ) {
        GetRangeRe sponse getRangeRe sponse = client .
getRange ( new GetRangeRe quest ( rangeRowQu eryCriteri a ));
        for ( Row row : getRangeRe sponse . getRows ()) {
            System . out . println ( row );
        }

        // If the nextStartP rimaryKey value is not
null , you can continue to read data from the base
  table .
        if ( getRangeRe sponse . getNextSta rtPrimaryK ey () !
= null ) {
            rangeRowQu eryCriteri a . setInclusi veStartPri
maryKey ( getRangeRe sponse . getNextSta rtPrimaryK ey ());
        } else {
            break ;
        }
    }
}
```

- **If you want to fetch the rows where the value of BaseStationNumber `003` matches the StartTime value range from `1532574861` to `1532584054`. Only the Duration will be displayed in the rows.**

  In this query, you specify both `BaseStatio nNumber` and `StartTime` as conditions. Only `Duration` appears in the result set. You can issue a query on the last index, and then fetch Duration by querying the base table.

```
private static void getRowFrom IndexAndMa inTable (
SyncClient client ,
                                            long baseStatio
nNumber ,
                                            long startTime ,
                                            long endTime ,
```

```
                                                    String   colName )
{
    RangeRowQu  eryCriteri  a   rangeRowQu  eryCriteri  a  =  new
RangeRowQu  eryCriteri  a ( INDEX1_NAM  E );

   //  You   can   specify   primary   keys .
    PrimaryKey  Builder   startPrima  ryKeyBuild  er  =
PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
DEFINED_CO  L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio
nNumber ));
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Value . fromLong ( startTime
));
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Value . INF_MIN );
    rangeRowQu  eryCriteri  a . setInclusi  veStartPri  maryKey (
startPrima  ryKeyBuild  er . build ());

   //  You   can   specify   primary   keys .
    PrimaryKey  Builder   endPrimary  KeyBuilder  =  PrimaryKey
Builder . createPrim  aryKeyBuil  der ();
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( DEFINED_CO
L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio  nNumber ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_2 ,  PrimaryKey  Value . fromLong ( endTime ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_1 ,  PrimaryKey  Value . INF_MAX );
    rangeRowQu  eryCriteri  a . setExclusi  veEndPrima  ryKey (
endPrimary  KeyBuilder . build ());

    rangeRowQu  eryCriteri  a . setMaxVers  ions ( 1 );

    String   strBaseSta  tionNum  =  String . format ("% d ",
baseStatio  nNumber );
    String   strStartTi  me  =  String . format ("% d ",  startTime
);
    String   strEndTime  =  String . format ("% d ",  endTime );

    System . out . println (" The   list   of   calls   forwarded
  by   the   base   station " +  strBaseSta  tionNum  + " from "
+  strStartTi  me  + " to " +  strEndTime  + " is   listed   as
follows :");
    while  ( true ) {
        GetRangeRe  sponse   getRangeRe  sponse  =  client .
getRange ( new   GetRangeRe  quest ( rangeRowQu  eryCriteri  a ));
        For  ( Row   row :  fig .  getrows  ()){
            PrimaryKey   curIndexPr  imaryKey  =  row . getPrimary
Key ();
            PrimaryKey  Column   mainCalled  Number  =  curIndexPr
imaryKey . getPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_1 );
            PrimaryKey  Column   callStartT  ime  =  curIndexPr
imaryKey . getPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_2 );
            PrimaryKey  Builder   mainTableP  KBuilder  =
PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
            mainTableP  KBuilder . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_1 ,  mainCalled  Number . getValue ());
            mainTableP  KBuilder . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_2 ,  callStartT  ime . getValue ());
            PrimaryKey   mainTableP  K  =  mainTableP  KBuilder .
build (); //  You   can   specify   primary   keys   for   the
base   table .

          //  You   can   query   the   base   table .
```

```
            SingleRowQ ueryCriter ia  criteria = new
SingleRowQ ueryCriter ia ( TABLE_NAME , mainTableP K );
            criteria . addColumns ToGet ( colName ); // You can
  read the Duration attribute value of the base
table .
          // You can specify 1 to indicate the
latest data version will be read .
            criteria . setMaxVers ions ( 1 );
            GetRowResp onse getRowResp onse = client . getRow
( new GetRowRequ est ( criteria ));
            Row mainTableR ow = getRowResp onse . getRow ();

            System . out . println ( mainTableR ow );
        }

        // If the nextStartP rimaryKey value is not
null , you can continue to read data from the base
  table .
        if ( getRangeRe sponse . getNextSta rtPrimaryK ey () !
=  null ) {
            rangeRowQu eryCriteri a . setInclusi veStartPri
maryKey ( getRangeRe sponse . getNextSta rtPrimaryK ey ());
        } else {
            break ;
        }
    }
}
```

To improve query performance, you can create a compound index on `BaseStatio nNumber` and `StartTime`. You can specify `Duration` as an attribute of this index.

The following index is created.

`IndexOnBas eStation2`:

PK0	PK1	PK2	Defined0
BaseStatio nNumber	StartTime	CellNumber	Duration
001	1532574644	123456	600
001	1532574714	234567	10
002	1532574795	345678	5
002	1532574861	345678	100
003	1532574734	234567	20
003	1532584054	456789	200

You can query the `IndexOnBas eStation2` index:

```
private static void getRangeFr omIndexTab le ( SyncClient
client ,
```

```
                                            long   baseStatio
nNumber ,
                                            long   startTime ,
                                            long   endTime ,
                                            String   colName ) {
    RangeRowQu  eryCriteri  a   rangeRowQu  eryCriteri  a  =  new
RangeRowQu  eryCriteri  a ( INDEX2_NAM  E );

   //  You   can   specify   primary   keys .
    PrimaryKey  Builder   startPrima  ryKeyBuild  er  =
PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
DEFINED_CO  L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio
nNumber ));
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Value . fromLong ( startTime
));
    startPrima  ryKeyBuild  er . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Value . INF_MIN );
    rangeRowQu  eryCriteri  a . setInclusi  veStartPri  maryKey (
startPrima  ryKeyBuild  er . build ());

   //  You   can   specify   primary   keys .
    PrimaryKey  Builder   endPrimary  KeyBuilder  =  PrimaryKey
Builder . createPrim  aryKeyBuil  der ();
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( DEFINED_CO
L_NAME_3 ,  PrimaryKey  Value . fromLong ( baseStatio  nNumber ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_2 ,  PrimaryKey  Value . fromLong ( endTime ));
    endPrimary  KeyBuilder . addPrimary  KeyColumn ( PRIMARY_KE
Y_NAME_1 ,  PrimaryKey  Value . INF_MAX );
    rangeRowQu  eryCriteri  a . setExclusi  veEndPrima  ryKey (
endPrimary  KeyBuilder . build ());

   //  You   can   specify   the   attribute   name   to   read .
    rangeRowQu  eryCriteri  a . addColumns  ToGet ( colName );

    rangeRowQu  eryCriteri  a . setMaxVers  ions ( 1 );

    String   strBaseSta  tionNum  =  String . format ("% d ",
baseStatio  nNumber );
    String   strStartTi  me  =  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 " +  strStartTi  me  + " to " +  strEndTime  + " is   listed
as   follows :");
    while  ( true ) {
        GetRangeRe  sponse   getRangeRe  sponse  =  client .
getRange ( new   GetRangeRe  quest ( rangeRowQu  eryCriteri  a ));
        for  ( Row   row  :  getRangeRe  sponse . getRows ()) {
            System . out . println ( row );
       }

       //  If   the   nextStartP  rimaryKey   value   is   not
null ,  you   can   continue   to   read   data   from   the   base
  table .
        if  ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey () !
=  null ) {
            rangeRowQu  eryCriteri  a . setInclusi  veStartPri
maryKey ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey ());
       }  else  {
```

```
                break ;
            }
        }
}
```

Hence, if you do not specify `Duration` as an index attribute, you have to retrieve Duration by querying the base table. However, when you specify `Duration` as an index attribute, this attribute data is stored in the base table and the index. The configuration improves query performance at the cost of disk space consumption.

- If you want to fetch the following values from a result set: total call duration, the average call duration, the maximum call duration, and the minimum call duration.This result set is a value of BaseStationNumber `003` with a StartTime value range from `1532574861` to `1532584054` .

  Compared to the last query, return is not required for each call duration. However, return is required for duration statistics. You can fetch results using the same method as the last query. Then you can perform Duration calculations to obtain the required result. In addition, you can execute SQL statements in SQL-on-OTS to obtain statistics. For more information about how to activate SQL-on-OTS, see *OLAP on Table Store: serverless SQL big data analysis on Data Lake Analytics*. You can use most MySQL syntax in SQL-on-OTS. Additionally, with SQL-on-OTS, you can easily process complicated calculations that are applicable to your business.

8.4 Java SDK for global secondary indexes

In this section, you can call the `createTable` method and the `scanFromIndex` method in the Java SDK to perform the following operations.

- You can create a base table and an index on this base table at the same time.

```
private static void createTabl e ( SyncClient client ) {
    TableMeta tableMeta = new TableMeta ( TABLE_NAME );
    tableMeta . addPrimary KeyColumn ( new PrimaryKey Schema (
PRIMARY_KE Y_NAME_1 , PrimaryKey Type . STRING )); // You can
  specify a primary key for a base table .
    tableMeta . addPrimary KeyColumn ( new PrimaryKey Schema
( PRIMARY_KE Y_NAME_2 , PrimaryKey Type . INTEGER )); // Set
primary key for the base table
    tableMeta . addDefined Column ( new DefinedCol umnSchema (
DEFINED_CO L_NAME_1 , DefinedCol umnType . STRING )); // You
can specify a pre - defined attribute for the base
table .
    tableMeta . addDefined Column ( new DefinedCol umnSchema (
DEFINED_CO L_NAME_2 , DefinedCol umnType . INTEGER )); // You
can specify a pre - defined attribute for the base
table .
```

```
    tableMeta . addDefined  Column ( new   DefinedCol  umnSchema (
DEFINED_CO  L_NAME_3 ,  DefinedCol  umnType . INTEGER )); //  You
can   specify   a   pre - defined   attribute   for   the   base
table .

    int   timeToLive  = - 1 ; //  You   can   specify  - 1   as
the   Time   To   Live  ( TTL )  value   so   the   data   never
expires .
    int   maxVersion  s  =  1 ; //  The   maximum   version
number .  You   can   only   specify   1   as   the   version
value   when   a   base   table   have   one   or   more   indexes
.

    TableOptio  ns   tableOptio  ns  =  new   TableOptio  ns (
timeToLive ,  maxVersion  s );

    ArrayList < IndexMeta >  indexMetas  =  new   ArrayList <
IndexMeta >();
    IndexMeta   indexMeta  =  new   IndexMeta ( INDEX_NAME ); //
You   can   create   an   index .
    indexMeta . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_1 ); //
You   can   specify   DEFINED_CO  L_NAME_1   of   the   base
table   as   an   index   primary   key .
    indexMeta . addDefined  Column ( DEFINED_CO  L_NAME_2 ); //
You   can   specify   DEFINED_CO  L_NAME_2   of   the   base
table   as   an   index   primary   key .
    indexMetas . add ( indexMeta ); //  You   can   add   the
index   to   the   base   table .

    CreateTabl  eRequest   request  =  new   CreateTabl  eRequest (
tableMeta ,  tableOptio  ns ,  indexMetas ); //  You   can   create
  the   base   table .

    client . createTabl  e ( request );
}
```

- **You can create an index on a base table.**

```
private   static   void   createInde  x ( SyncClient   client ) {
    IndexMeta   indexMeta  =  new   IndexMeta ( INDEX_NAME ); //
You   can   create   an   index .
    indexMeta . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_2 ); //
You   can   specify   DEFINED_CO  L_NAME_2   as   the   first
attribute   of   an   index   primary   key .
    indexMeta . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_1 ); //
You   can   specify   DEFINED_CO  L_NAME_1   as   the   second
attribute   of   an   index   primary   key .
    CreateInde  xRequest   request  =  new   CreateInde  xRequest (
TABLE_NAME ,  indexMeta ,  false ); //  You   can   create   an
index   on   a   base   table .
    client . createInde  x ( request ); //  You   can   create   an
  index .
}
```

Note:

At the moment, existing data in the base table will not be copied to the index when you create an index on a base table. The newly created index only includes

incremental data after you create this index. For more information about incremental data, contact Table Store technical support with DingTalk.

- **You can delete an index.**

```
private static void deleteInde x ( SyncClient client ) {
    DeleteInde xRequest request = new DeleteInde xRequest (
TABLE_NAME , INDEX_NAME ); // You can specify the names
of a base table and an index .
    client . deleteInde x ( request ); // You can delete an
  index .
}
```

- **You can read data from an index.**

  If an index includes an attribute that will be returned in results, you can directly retrieve data from the index.

```
private static void scanFromIn dex ( SyncClient client ) {
    RangeRowQu eryCriteri a rangeRowQu eryCriteri a = new
RangeRowQu eryCriteri a ( INDEX_NAME ); // You can specify
the name of an index .

   // You can specify the start primary key .
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_1 , PrimaryKey Value . INF_MIN ); // You
can specify the minimum value for an index primary
  key .
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN ); // You
can specify the minimum value for a base table
primary key .
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . INF_MIN ); // You
can specify the minimum value for a base table
primary key .
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

   // You can specify the end primary key .
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_1 , PrimaryKey Value . INF_MAX ); // You can specify
  the maximum value for an index attribute .
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . INF_MAX ); // You can specify
  the maximum value for a base table primary key .
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX ); // You can specify
  the maximum value for a base table primary key .
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());

    rangeRowQu eryCriteri a . setMaxVers ions ( 1 );

    System . out . println (" The results returned from an
  index are as follows :");
```

```
    while ( true ) {
        GetRangeRe sponse getRangeRe sponse = client .
getRange ( new GetRangeRe quest ( rangeRowQu eryCriteri a ));
        for ( Row row : getRangeRe sponse . getRows ()) {
            System . out . println ( row );
        }

        // If the nextStartP rimaryKey value is not
null , you can continue to read data from the base
  table .
        if ( getRangeRe sponse . getNextSta rtPrimaryK ey () !
= null ) {
            rangeRowQu eryCriteri a . setInclusi veStartPri
maryKey ( getRangeRe sponse . getNextSta rtPrimaryK ey ());
        } else {
            break ;
        }
    }
}
```

If an index does not include an attribute that will be returned in results, you must query the base table.

```
private static void scanFromIn dex ( SyncClient client ) {
    RangeRowQu eryCriteri a rangeRowQu eryCriteri a = new
RangeRowQu eryCriteri a ( INDEX_NAME ); // You can specify
the index name .

   // You can specify the start primary key .
    PrimaryKey Builder startPrima ryKeyBuild er =
PrimaryKey Builder . createPrim aryKeyBuil der ();
    startPrima ryKeyBuild er . addPrimary KeyColumn (
DEFINED_CO L_NAME_1 , PrimaryKey Value . INF_MIN ); // You
  can specify the minimum value for an indexed
attribute of an index .
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_1 , PrimaryKey Value . INF_MIN ); // You
can specify the minimum value for a primary key
of a base table .
    startPrima ryKeyBuild er . addPrimary KeyColumn (
PRIMARY_KE Y_NAME_2 , PrimaryKey Value . INF_MIN ); // You
can specify the minimum value for a primary key
of a base table .
    rangeRowQu eryCriteri a . setInclusi veStartPri maryKey (
startPrima ryKeyBuild er . build ());

   // You can specify the end primary key .
    PrimaryKey Builder endPrimary KeyBuilder = PrimaryKey
Builder . createPrim aryKeyBuil der ();
    endPrimary KeyBuilder . addPrimary KeyColumn ( DEFINED_CO
L_NAME_1 , PrimaryKey Value . INF_MAX ); // You can specify
  the maximum value for an indexed attribute of an
  index .
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_1 , PrimaryKey Value . INF_MAX ); // You can specify
  the maximum value for a base table primary key .
    endPrimary KeyBuilder . addPrimary KeyColumn ( PRIMARY_KE
Y_NAME_2 , PrimaryKey Value . INF_MAX ); // You can specify
  the maximum value for a base table primary key .
    rangeRowQu eryCriteri a . setExclusi veEndPrima ryKey (
endPrimary KeyBuilder . build ());
```

```
    rangeRowQu  eryCriteri  a . setMaxVers  ions ( 1 );

    while  ( true ) {
        GetRangeRe  sponse  getRangeRe  sponse  =  client .
getRange ( new  GetRangeRe  quest ( rangeRowQu  eryCriteri  a ));
        for  ( Row  row  :  getRangeRe  sponse . getRows ()) {
            PrimaryKey  curIndexPr  imaryKey  =  row . getPrimary
Key ();
            PrimaryKey  Column  pk1  =  curIndexPr  imaryKey .
getPrimary  KeyColumn ( PRIMARY_KE  Y_NAME1 );
            PrimaryKey  Column  pk2  =  curIndexPr  imaryKey .
getPrimary  KeyColumn ( PRIMARY_KE  Y_NAME2 );
            PrimaryKey  Builder  mainTableP  KBuilder  =
PrimaryKey  Builder . createPrim  aryKeyBuil  der ();
            mainTableP  KBuilder . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME1 ,  pk1 . getValue ());
            mainTableP  KBuilder . addPrimary  KeyColumn (
PRIMARY_KE  Y_NAME2 ,  ke2 . getValue ());
            PrimaryKey  mainTableP  K  =  mainTableP  KBuilder .
build (); //  You  can  specify  the  index  primary  keys
for  a  base  table .

           //  You  can  query  a  base  table .
            SingleRowQ  ueryCriter  ia  criteria  =  new
SingleRowQ  ueryCriter  ia ( TABLE_NAME ,  mainTableP  K );
            criteria . addColumns  ToGet ( DEFINED_CO  L_NAME3
); //  You  can  read  the  DEFINED_CO  L_NAME3  attribute
from  the  base  table .
           //  You  can  retrieve  the  latest  data  version
.
            criteria . setMaxVers  ions ( 1 );
            GetRowResp  onse  getRowResp  onse  =  client . getRow
( new  GetRowRequ  est ( criteria ));
            Row  mainTableR  ow  =  getRowResp  onse . getRow ();
            System . out . println ( row );
        }

        //  If  the  value  of  nextStartP  rimaryKey  is  not
  null ,  you  can  continue  to  read  data  from  the
base  table .
         if  ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey () !
=  null ) {
            rangeRowQu  eryCriteri  a . setInclusi  veStartPri
maryKey ( getRangeRe  sponse . getNextSta  rtPrimaryK  ey ());
        }  else  {
            break ;
        }
    }
}
```

8.5 APIs

CreateTable

You can call the `CreateTable` method to create a table, and an index with pre-defined attributes at the same time.

When you write data to a base table, an index on this base table is also updated. For more information, see *CreateTable*.

CreateIndex

You can call the `CreateIndex` method to create an index on a base table.

> Note:
> The current version does not support copying existing base table data to the index when you call the CreateIndex method to create an index on a base table. This feature will be supported by later versions.

DeleteIndex

You can call the `DeleteIndex` method to delete indexes on a base table. The other indexes on this table will not be affected.

DeleteTable

You can call the `DeleteTable` method to delete a base table and all indexes on this table. For more information, see *DeleteTable*.

8.6 Billing rules

To use secondary indexes, index tables are needed. Therefore, additional storage space is required to store index tables. When the system inserts data to a primary table, it may also need to write the index tables created on the primary table at the same time. During this process, read and write CUs are consumed. This topic describes the billing rules for secondary indexes.

> Note:
> Capacity units (CUs) are read and write throughput units. They are the smallest units used to measure the costs of read and write operations For example, when the system reads 4 KB from one row per second, one read CU is consumed.

To use secondary indexes, index tables are needed. Therefore, additional storage space is required to store index tables. When the system inserts data to a primary table, it may also need to write the index tables created on the primary table at the same time. During this process, read and write CUs are consumed.

Secondary index billing includes the following parts: the number of read and write CUs consumed to write index tables, the amount of data stored in the index tables, and the amount of data that is read from the index tables.

Billing item	Description
Data storage	The storage space used to store a primary table and its index tables.
Read CUs consumed to write index tables	The number of CUs that are consumed by read operations to delete, insert, or update index rows.
Write CUs consumed to write index tables	The number of CUs that are consumed to insert or update index rows.
CUs consumed by regular read operations	The number of CUs that are consumed to read data from a primary table or index tables using an API.
CUs consumed by regular write operations	The number of CUs that are consumed to insert data to a primary table using an API.

Billing rules for storing, writing, and reading an index table:

- The billing rules for storing and reading an index table are the same as those of a primary table. For more information, see *Billing items and pricing*.
- CUs are consumed based on the following rules when the system writes an index table:
  - Write CUs are consumed only when an index row is inserted or updated.
  - Read CUs are consumed when an index row is deleted, updated, or inserted. The number of read CUs equals the amount of data read from the corresponding indexed columns in the primary table.

Calculate the number of read CUs consumed to write index tables

When you create secondary indexes on the primary table, read CUs are consumed based on the following rules:

- When you use the PUT operation to insert a data row to the primary table:
  - The PUT operation does not insert data to the indexed attribute columns in the primary table, which means that no index row is inserted. In this case, one read CU is consumed.
  - The PUT operation inserts data to the indexed attribute columns in the primary table, which means that new index rows are inserted. In this case, one read CU is consumed.
- When you use the PUT operation to overwrite a row in the primary table:
  - The PUT operation does not update the indexed attribute columns in the primary table. In this case, one read CU is consumed.
  - The PUT operation updates the indexed attribute columns in the primary table. In this case, the read CUs are consumed as follows:

    Divide the total amount of data read from the indexed attribute columns by four, excluding primary key columns. The number of consumed CUs equals the calculated value rounded up to the nearest integer. If the total amount is 0 KB, one CU is consumed.
- When you use the UPDATE operation to insert a data row to the primary table:
  - If the UPDATE operation does not insert data to the indexed columns in the primary table, no read CU is consumed.
  - If the UPDATE operation inserts data to the indexed columns in the primary table, one read CU is consumed.
- When you use the UPDATE operation to update a row in the primary table:
  - If the UPDATE operation does not insert data to the indexed attribute columns in the primary table, no read CU is consumed.
  - If the UPDATE operation inserts data to the indexed attribute columns in the primary table, read CUs are consumed based on the following rules:

    Divide the total amount of data read from the indexed columns by four, excluding the primary key columns. The number of consumed CUs equals the calculated value rounded up to the nearest integer. If the total amount is 0 KB, one CU is consumed.

- When you use the Delete operation to delete a row in the primary table, read CUs are consumed based on the following rules:

  Divide the total amount of data read from the indexed columns by four, excluding the primary key columns. The number of consumed CUs equals the calculated value rounded up to the nearest integer. If the total amount is 0 KB, one CU is consumed.
- If the primary table uses primary key auto increment, inserting data to the primary table does not consume any read CUs. Updating a row in a primary table that uses primary key auto increment consumes read CUs. CUs are calculated based on the same rules as those of the UPDATE operation.

   Note:
  We recommend that you use primary key auto increment to insert data to a primary table to decrease the number of CUs that are consumed by index tables.

  For primary tables that do not use primary key auto increment, one read CU is consumed if a read operation is performed on the indexed columns, even if no data is retrieved. For primary tables that use primary key auto increment, no read operation is performed on the indexed columns when you insert data. Therefore, no read CU is consumed.

Calculate the number of write CUs

When you insert data to the primary table and create secondary indexes, write CUs are consumed. Write CUs are consumed based on the following rules:

- If you insert a row to the primary table and no data in the index table is updated, no write CUs are consumed.
- If you insert a row to the primary table and a new index row is inserted to the index table, write CUs are consumed. The number of the write CUs is determined by the size of the inserted index row.
- If you insert a row to the primary table and an index row is deleted from the index table, write CUs are consumed. The number of the write CUs is determined by the size of the deleted index row.
- If you insert a row to the primary table and an index row in the index table is updated, write CUs are consumed. The number of the write CUs is determined by the size of the updated index row.

- If you insert a row to the primary table, an index row is deleted from the index table, and another index row is inserted to the index table, write CUs are consumed . The number of the write CUs is determined by the total size of the deleted and inserted index rows.

The detailed rules are as follows:

- When you use the PUT operation to insert a data row to a primary table:
  - The PUT operation does not insert data to the indexed attribute columns in the primary table, which means that no index row is inserted. In this case, no read CU is consumed.
  - The PUT operation inserts data to the indexed attribute columns in the primary table, which means that new index rows are inserted. The write CUs consumed for each index table are calculated as follows:

    Divide the total amount of data in the inserted index row by four. The number of consumed CUs equals the calculated value rounded up to the nearest integer.
- When you use the PUT operation to overwrite a row in the primary table:
  - The PUT operation only updates the indexed primary key columns in the primary table. In this case, no write CUs are consumed.
  - The PUT operation updates the indexed columns in the primary table. The write CUs are consumed based on the following rules:

    All indexes updated by the PUT operation consume a certain number of write CUs, except sparse indexes.
- When you use the UPDATE operation to insert a data row to the primary table:
  - If the UPDATE operation does not insert data to the indexed columns in the primary table, no write CUs are consumed.
  - If the UPDATE operation inserts data to the indexed columns in the primary table, the write CUs consumed for each index table are calculated as follows:
    - If the UPDATE operation inserts a new index row, write CUs are consumed . Divide the total size of the data in the index row by four. The number of consumed CUs equals the calculated value rounded up to the nearest integer.
    - If no index row is inserted, no write CUs are consumed.

- When you use the UPDATE operation to update a row in the primary table:
  - If the UPDATE operation does not update the indexed attribute columns, no write CUs are consumed.
  - If the UPDATE operation updates the indexed attribute columns, write CUs consumed for each index table are calculated based on the following rules:
    - If the index table already contains an index row created based on the row to be updated, delete CUs are consumed. The number of the delete CUs is determined by the size of the indexed primary keys in the deleted index row.
    - If a new index row is inserted based on the updated row, write CUs are consumed. The number of the write CUs is determined by the size of the indexed primary keys in the inserted index row.
    - If the UPDATE operation only updates the attribute data in the existing index row but no new index row is inserted, update CUs are consumed.

    Divide the total amount of data in the index row by four. The number of consumed CUs equals the calculated value rounded up to the nearest integer.
- When you use the DELETE operation to delete a row in the primary table, write CUs are consumed based on the following rules:

  If an index table already contains an index row created based on the row to be deleted, write CUs are consumed. Divide the total amount of the data in the corresponding indexed columns by four, excluding the primary key columns. The consumed write CUs equal the calculated value rounded up to the nearest integer.
- If you insert data to a primary table that uses primary key auto increment, write CUs are consumed. The write CUs are calculated based on the same rules as those of the PUT operation. If you update a row in a primary table that uses primary key auto increment, write CUs are consumed. The write CUs are calculated based on the same rules as those of the UPDATE operation.

Measure index table size

The size of an index table is measured based on the same rule as that of a primary table. The size of an index table equals the total size of all rows. The total size of the rows equals the total size of primary keys and attribute data. For more information, see *Data storage*.

Calculate the number of CUs consumed to read an index table

When you use an SDK, the console, or other methods, such as a DLA, to read an index table, read CUs are consumed. The number of read CUs are calculated based on the same rules as those of reading a primary table.

Examples

The following example uses a primary table that has two index tables to describe how CUs are consumed under different conditions.

The primary table Table contains two primary key columns PK0 and PK1, and three predefined columns Col0, Col1, and Col2. Two index tables, Index0 and Index1, are created on the primary table. Index0 contains three primary keys Col0, PK0, and PK1 and one attribute column Col2. Index1 contains four primary keys Col1, Col0, PK0, and PK1, and no attribute columns. Use the UPDATE operation to update PK0 and PK1.

- If the target row does not exist in the primary table:
  - Updating Col3 does not consume read or write CUs.
  - Updating Col1 consumes the following CUs:
    - One read CU
    - No write CUs
  - Updating Col0 and Col1 consumes the following CUs:
    - One read CU
    - Index0 consumes write CUs. The number of the write CUs is determined by the total amount of data inserted to Col0, PK0, and PK1. Index1 consumes write CUs. The number of the write CUs is determined by the total amount of data inserted to Col0, Col1, PK0, and PK1.
- If the target row already exists in the primary table:
  - Updating Col3 does not consume read or write CUs.
  - Updating Col2 consumes the following CUs:
    - Read CUs are consumed. The number of the read CUs is determined by the amount of data read from Col0. If the UPDATE operation inserts data to Col0, one CU is consumed.
    - For Index0, if the UPDATE operation insets data to Col0, Index0 does not consume write CUs. If the UPDATE operation updates the data in Col0, Index0

consumed write CUs. The number of the write CUs is determined by the total amount of data inserted to Col0, PK0, PK1, and Col2. Index1 does not consume write CUs.

- Updating Col1 consumes the following CUs:
  - Read CUs are consumed. The number of the read CUs is determined by the amount of data read from Col0 and Col1. If the total amount is 0 KB, one CU is consumed.
  - Index0 does not consume write CUs. For Index1, if an index row is inserted, write CUs are consumed. The number of the write CUs is determined by the amount of data read from Col0 and inserted to Col1, PK0, and PK1. For Index1 , if no data in Col0 is updated, no index row is inserted and no write CUs are consumed. If the data in Col0 and Col1 is updated, write CUs are consumed to delete the corresponding index row. The number of write CUs is determined by the total amount of data read from Col0, Col1, PK0, and PK1.

8.7 Appendix

You can create tables and indexes as follows:

```
private   static   final   String   TABLE_NAME  = " CallRecord  Table
";
    private   static   final   String   INDEX0_NAM  E  = "
IndexOnBeC  alledNumbe  r ";
    private   static   final   String   INDEX1_NAM  E  = "
IndexOnBas  eStation1 ";
    private   static   final   String   INDEX2_NAM  E  = "
IndexOnBas  eStation2 ";
    private   static   final   String   PRIMARY_KE  Y_NAME_1  = "
CellNumber ";
    private   static   final   String   PRIMARY_KE  Y_NAME_2  = "
StartTime ";
    private   static   final   String   DEFINED_CO  L_NAME_1  = "
CalledNumb  er ";
    private   static   final   String   DEFINED_CO  L_NAME_2  = "
Duration ";
    private   static   final   String   DEFINED_CO  L_NAME_3  = "
BaseStatio  nNumber ";

    private   static   void   createTabl  e ( SyncClient   client ) {
        TableMeta   tableMeta  =  new   TableMeta ( TABLE_NAME );
        tableMeta . addPrimary  KeyColumn ( new   PrimaryKey  Schema
( PRIMARY_KE  Y_NAME_1 ,  PrimaryKey  Type . INTEGER ));
        tableMeta . addPrimary  KeyColumn ( new   PrimaryKey  Schema
( PRIMARY_KE  Y_NAME_2 ,  PrimaryKey  Type . INTEGER ));
        tableMeta . addDefined  Column ( new   DefinedCol  umnSchema
( DEFINED_CO  L_NAME_1 ,  DefinedCol  umnType . INTEGER ));
        tableMeta . addDefined  Column ( new   DefinedCol  umnSchema
( DEFINED_CO  L_NAME_2 ,  DefinedCol  umnType . INTEGER ));
```

```
        tableMeta . addDefined  Column ( new   DefinedCol  umnSchema
( DEFINED_CO  L_NAME_3 ,  DefinedCol  umnType . INTEGER ));

        int   timeToLive  = - 1 ; //  The   time   before   the
data   expires . You   can   specify  - 1   as   the   Time   To
Live  ( TTL )  value   so   the   data   never   expires .  Unit :
seconds .  You   must   specify  - 1   as   the   TTL   value   when
  a   table   has   one   or   more   indexes .
        int   maxVersion  s  =  1 ; //   The   maximum   number   of
  versions .  You   must   specify   1   as   the   value   when   a
  table   has   one   or   more   indexes .

        TableOptio  ns   tableOptio  ns  =  new   TableOptio  ns (
timeToLive ,  maxVersion  s );

        ArrayList < IndexMeta >  indexMetas  =  new   ArrayList <
IndexMeta >();
        IndexMeta   indexMeta0  =  new   IndexMeta ( INDEX0_NAM  E );
        indexMeta0 . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_1 );
        indexMetas . add ( indexMeta0 );
        IndexMeta   indexMeta1  =  new   IndexMeta ( INDEX1_NAM  E );
        indexMeta1 . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_3 );
        indexMeta1 . addPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_2 );
        indexMetas . add ( indexMeta1 );
        IndexMeta   indexMeta2  =  new   IndexMeta ( INDEX2_NAM  E );
        indexMeta2 . addPrimary  KeyColumn ( DEFINED_CO  L_NAME_3 );
        indexMeta2 . addPrimary  KeyColumn ( PRIMARY_KE  Y_NAME_2 );
        indexMeta2 . addDefined  Column ( DEFINED_CO  L_NAME_2 );
        indexMetas . add ( indexMeta2 );

        CreateTabl  eRequest   request  =  new   CreateTabl  eRequest
( tableMeta ,  tableOptio  ns ,  indexMetas );

        client . createTabl  e ( request );
   }
```