# Alibaba Cloud Tablestore

## API Reference

Issue: 20190911

# 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:<br>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:<br>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:<br>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:<br>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 Operations

## 1.1 PutRow

PutRow inserts data in the specified row. If the row does not exist, a new row is added. If the row exists, the original row is overwritten.

Request message structure:

```
message PutRowRequest {
    required string table_name = 1;
    required bytes row = 2; // The Plainbuffer encoding is binary
    required Condition condition = 3;
    optional ReturnContent return_content = 4;
}
```

`table_name`:

- Type: String
- Required parameter: Yes
- The name of the table to write data to.

`row`:

- Type: Bytes
- Required parameter: Yes
- The data to be written into the row. It is in Plainbuffer format and includes the primary key and attribute column. For more information about encoding, see Plainbuffer encoding.

`condition`:

- Type: Condition
- Required parameter: Yes
- Determines whether to perform row existence check before writing data. It can be one of three values:
  - IGNORE indicates that the row existence check is not performed.
  - EXPECT_EXIST indicates that the row is expected to exist.
  - EXPECT_NOT_EXIST indicates that the row is not expected to exist.

- If the row is not expected to exist but it does, or conversely the row is expected to exist but it does not, insertion fails and an error is returned.

return_con tent :

- Type: ReturnContent
- Required parameter: No
- The data type after the row is successfully written. Currently, only the primary key can be returned, which is used for the auto-increment function of the primary key column.

Response message structure:

```
message PutRowResp onse {
    required ConsumedCa pacity consumed = 1 ;
    optional bytes row = 2 ;
}
```

consumed :

- Type: ConsumedCapacity
- The capacity units consumed by this operation.

Capacity unit consumption:

- When the row does not exist:
  - If the value of the specified condition check is IGNORE, then the number of consumed write capacity units is the sum of the size of the primary key of the row and the size of the attribute column to be inserted divided by 4 KB and rounded up.
  - If the value of the specified condition check is EXPECT_NOT_EXIST, both write capacity units and read capacity units are consumed. The number of consumed write capacity units is the sum of the size of the primary key of the row and the size of the attribute column to be inserted divided by 4 KB and rounded up, and the number of consumed read capacity units is the size of the primary key of the row divided by 4 KB and rounded up.
  - If the value of the specified condition check is EXPECT_EXIST, then insertion of the row fails, which consumes one write capacity unit and one read capacity unit.

- When the row exists:
  - If the value of the specified condition check is IGNORE, then the number of consumed write capacity units is the sum of the size of the primary key of the row and the size of the attribute column to be inserted divided by 4 KB and rounded up.
  - If the value of the specified condition check is EXPECT_EXIST, both write capacity units and read capacity units are consumed. The number of consumed write capacity units is the sum of the size of the primary key of the row and the size of the attribute column to be inserted divided by 4 KB and rounded up, and the number of consumed read capacity units is the size of the primary key of the row divided by 4 KB and rounded up.
  - If the value of the specified condition check is EXPECT_NOT_EXIST, then insertion of the row fails, which consumes one write capacity unit and one read capacity unit.
- Conditional Update:
  - If the operation succeeds, the consumed capacity units are calculated according to the preceding method.
  - If the operation fails, one write capacity unit and one read capacity unit are consumed.

  For more information about how to calculate the data size, see Billing.
- If an internal error code is returned (HTTP status code: 5XX), no capacity units are consumed. If other errors are returned, one write capacity unit is consumed.
- If the request times out and the results are undefined, a capacity unit may or may not be consumed.

`row :`

- Type: Bytes
- The returned data when return_content is set.
- If return_content is not set or no value is returned, NULL is returned.
- It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

## 1.2 UpdateRow

UpdateRow updates the data of the specified row. If the row does not exist, a new row is added. If the row exists, the values of the specified columns are added, modified, or deleted based on the request content.

Request message structure:

```
message UpdateRowRequest {
    required string table_name = 1;
    required bytes row_change = 2;
    required Condition condition = 3;
    optional ReturnContent return_content = 4;
}
```

table_name :

- Type: String
- Required parameter: Yes
- The name of the table whose data is to be updated.

row_change :

- Type: Bytes
- Required parameter: Yes
- The data to be updated. It is in Plainbuffer format and includes the primary key and attribute column. For more information about encoding, see Plainbuffer encoding.
- All the attribute columns to be updated for this row. Table Store adds, modifies , or deletes the values for the specified columns based on the content of each UpdateType in row_change.
- Columns that exist in the row but does not exist in row_change are not affected.
- UpdateType can have the following values:
  - PUT: The UpdateType value must be a valid attribute column value. This indicates that, if this column does not exist, a new one is added. If the column exists, it is overwritten.
  - DELETE: The UpdateType value must be null, while the timestamp must be specified. This indicates that, data of the specified version in the column is deleted.
  - DELETE_ALL: The UpdateType value and time stamp must be null. This indicates that, data of all versions in the column is deleted.

**Note:**

**Deleting all attribute columns of a row is not the same as deleting the row. If you want to delete the row, use the DeleteRow operation.**

```
condition :
```

- **Type: Condition**
- **Required parameter: Yes**
- **Determines whether to perform existence check before data updating. It can be one of two values:**
  - **IGNORE indicates that the row existence check is not performed.**
  - **EXPECT_EXIST indicates that the row is expected to exist.**
- **If this row is expected to exist but does not, the update operation fails and an error is returned. If the existence of the row is ignored, this operation is successful regardless of the row existence.**

```
return_con  tent :
```

- **Type: ReturnContent**
- **Required parameter: No**
- **The data type after the row is successfully written. Currently, only the primary key can be returned, which is used for the auto-increment function of the primary key column.**

Response message structure:

```
 message   UpdateRowR  esponse  {
     required   ConsumedCa  pacity   consumed  =  1 ;
     optional   bytes   row  =  2 ;
}
```

```
consumed :
```

- **Type: ConsumedCapacity**
- **The capacity units consumed by this operation.**

**Capacity unit consumption:**

- **When the row does not exist:**
  - **If the value of the specified condition check is IGNORE, then the number of consumed write capacity units is the sum of the size of the primary key of the**

row and the size of the attribute column to be updated divided by 4 KB and rounded up. If UpdateRow contains an attribute column to be deleted, only the column name length is included in the size of the attribute column.

- If the value of the specified condition check is EXPECT_EXIST, then insertion of the row fails, which consumes one write capacity unit and one read capacity unit.

· When the row exists:

- If the value of the specified condition check is IGNORE, then the number of consumed write capacity units is the sum of the size of the primary key of the row and the size of the attribute column to be updated divided by 4 KB and rounded up. If UpdateRow contains an attribute column to be deleted, only the column name length is included in the size of the attribute column.
- If the value of the specified condition check is EXPECT_EXIST, both write capacity units and read capacity units are consumed. The number of consumed write capacity units is the same as that when the value of the specified condition check is IGNORE, and the number of consumed read capacity units is the size of the primary key of the row divided by 4 KB and rounded up.

For more information about how to calculate the data size, see Billing.

· If the request times out and the results are undefined, a capacity unit may or may not be consumed.
· If an internal error code is returned (HTTP status code: 5XX), no capacity units are consumed. If other errors are returned, one write capacity unit and one read capacity unit are consumed.

`row :`

· Type: Bytes
· The returned data when return_content is set.
· If return_content is not set or no value is returned, NULL is returned.
· It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

## 1.3 DeleteRow

DeleteRow deletes a data row.

Request message structure:

```
message DeleteRowRequest {
    required string table_name = 1;
    required bytes primary_key = 2; // The Plainbuffer
  encoding is binary
    required Condition condition = 3;
    optional ReturnContent return_content = 4;
}
```

`table_name`:

- Type: String
- Required parameter: Yes
- The name of the table whose data is to be updated.

`primary_key`:

- Type: Bytes
- Required parameter: Yes
- The primary key of the row to be deleted, which is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

`condition`:

- Type: Condition
- Required parameter: Yes
- Determines whether to perform existence check before data updating. It can be one of two values:
  - IGNORE: indicates that the row existence check is not performed.
  - EXPECT_EXIST: indicates that the row is expected to exist.
- If this row is expected to exist but does not, the delete operation fails and an error is returned. If the existence of the row is ignored, this operation is successful regardless of the row's existence.

`return_content`:

- Type: ReturnContent
- Required parameter: No

- The data type after the row is successfully written. Currently, only the primary key can be returned, which is used for the auto-increment function of the primary key column.

Response message structure:

```
message DeleteRowResponse {
    required ConsumedCapacity consumed = 1;
    optional bytes row = 2;
}
```

consumed :

- Type: ConsumedCapacity
- The capacity units consumed by this operation.

Capacity unit consumption:

- When the row does not exist:
  - If the value of the specified condition check is IGNORE, then the number of consumed write capacity units is the size of the primary key of the row divided by 4 KB and rounded up.
  - If the value of the specified condition check is EXPECT_EXIST, then deletion of the row fails and one write capacity unit and one read capacity unit are consumed.
- When the row exists:
  - If the value of the specified condition check is IGNORE, then the number of consumed write capacity units is the size of the primary key of the row divided by 4 KB and rounded up.
  - If the value of the specified condition check is EXPECT_EXIST, both write capacity units and read capacity units are consumed. The number of consumed write capacity units and the number of consumed read capacity units are the size of the primary key of the row divided by 4 KB and rounded up, respectively.

  For more information about how to calculate the data size, see Billing.
- If an internal error code is returned (HTTP status code: 5XX), this operation does not consume capacity units. If other errors are returned, one write capacity unit is consumed.
- If the request times out and the results are undefined, a capacity unit may or may not be consumed.

row :

- Type: bytes
- The returned data when return_content is set.
- If return_content is not set or no value is returned, NULL is returned.
- It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

## 1.4 GetRange

GetRange reads data within the specified primary key range.

Request structure:

```
message GetRangeRequest {
    required string table_name = 1;
    required Direction direction = 2;
    repeated string columns_to_get = 3; // If it is
not specified, all columns are read
    optional TimeRange time_range = 4;
    optional int32 max_versions = 5;
    optional int32 limit = 6;
    required bytes inclusive_start_primary_key = 7; //
The Plainbuffer encoding is binary
    required bytes exclusive_end_primary_key = 8; // The
Plainbuffer encoding is binary
    optional bytes filter = 10;
    optional string start_column = 11;
    optional string end_column = 12;
}
```

table_name :

- Type: String
- Required parameter: Yes
- The name of the table containing the data to be read.

direction :

- Type: Direction
- Required parameter: Yes

- The query direction.
  - If the query direction is FORWARD, the value of inclusive_start_primary must be smaller than that of exclusive_end_primary, and the rows in the response are sorted from the smallest to the largest primary keys.
  - If the query direction is BACKWARD, the value of inclusive_start_primary must be larger than that of exclusive_end_primary, and the rows in the response are sorted from the largest to the smallest primary keys.

`columns_to_get`:

- Type: Repeated string
- Required parameter: No
- The names of all columns to be returned. If it is null, all columns of each row in the read results are returned.
- If a duplicate column name is given, the returned results only include this column once.
- In columns_to_get, the maximum number of strings is 128.

`time_range`:

- Type: TimeRange
- Required parameter: Either time_range or max_versions is required.
- The range of time stamps to read versions of data.
- The minimum and maximum values of the time stamp are 0 and INT64. MAX, respectively.
- To query data of a time range, specify start_time and end_time.
- To query data of a specific time stamp, specify specific_time.
- Example: If the value of time_range is (100, 200), the time stamp of the returned column data must be within [100, 200).

`max_versions`:

- Type: int32
- Required parameter: Either time_range or max_versions is required.
- The maximum number of versions of data to be returned.
- Example: If the value of max_versions is 2, data of a maximum of two versions is returned for each column.

`limit :`

- Type: int32
- Required parameter: No
- The maximum number of rows of data to be returned. If the number of rows queried exceeds this value, the response includes a breakpoint to record the position where reading ended in the operation so that the next read can start at this position. This value must be greater than 0.
- Whether or not this value is set, Table Store returns a maximum of 5,000 data rows and the total data size never exceeds 4 MB.

`inclusive_ start_prim ary_key :`

- Type: Bytes
- Required parameter: Yes
- The row that contains the starting primary key for the range to be read. If this row exists, the response returns it. It is encoded in Plainbuffer format. For more information, see Plainbuffer encoding.

`exclusive_ end_primar y_key :`

- Type: Bytes
- Required parameter: Yes
- The row that contains the ending primary key for the range to be read. The response does not contain the row whether the row exists or not. It is encoded in Plainbuffer format. For more information, see Plainbuffer encoding.
- For GetRange, Column in inclusive_start_primary_key or exclusive_end_primar y_key can be specialized to INF_MIN or INF_MAX. If Column is set to INF_MIN, the value of the Column is always smaller than that of other Columns. If Column is set to INF_MAX, its value is always larger than other Columns.

`filter :`

- Type: Bytes
- Required parameter: No
- The filtering condition expression.
- It indicates the binary data after the Filter is serialized in Protobuf format.

`start_colu mn :`

- Type: String
- Required parameter: No
- The starting column to be read, which is used for reading a wide row.
- The returned results contain the current starting column.
- The column names are sorted lexicographically.
- Example: If a table contains columns “a”, “b”, and “c” and the value of start_column is “b”, the reading starts from column “b”, and columns “b” and “c” are returned.

`end_column :`

- Type: String
- Required parameter: No
- The ending column to be read, which is used for reading a wide row.
- The returned results do not contain the current ending column.
- The column names are sorted lexicographically.
- Example: If a table contains columns “a”, “b”, and “c” and the value of end_column is “b”, the reading ends at column “b”, and column “a” is returned.

Response message structure:

```
message GetRangeResponse {
    required ConsumedCapacity consumed = 1;
    required bytes rows = 2;
    optional bytes next_start_primary_key = 3;
}
```

`consumed :`

- Type: ConsumedCapacity
- The capacity units consumed by this operation.

`rows :`

- Type: Bytes
- Rows are encoded in Plainbuffer format. For more information, see Plainbuffer encoding.
- All data read. If the direction in the request is FORWARD, all rows are sorted from the smallest to the largest primary keys. If the direction in the request is BACKWARD, all rows are sorted from the largest to the smallest primary keys.

- primary_key_columns and attribute_columns for each row only contain the columns specified in columns_to_get. This means their order may not be consistent with that of columns_to_get in the request, nor may the order of primary_key_columns be consistent with the order specified when the table was created.
- If the specified columns_to_get in the request does not contain any primary key column, the primary key is within the query range. However, the row that does not contain any attribute column in columns_to_get is not contained in the response message.

`next_start_primary_key:`

- Type: Bytes
- The breakpoint information for this GetRange operation. It is encoded in Plainbuffer format. For more information, see Plainbuffer encoding.
- If it is null, the response message for this GetRange operation contains all data in the request range.
- If it is not null, the response message for this GetRange operation includes only the data in the range [inclusive_start_primary_key, next_start_primary_key). If you want the remaining data, you must use next_start_primary_key as inclusive_start_primary_key, and retain exclusive_end_primary_key in the original request to perform the subsequent GetRange operation.

 Note:
In Table Store, the number of data rows in the response message for a GetRange operation cannot exceed 5,000, and the size cannot exceed 4 MB. Furthermore, the volume of returned data cannot exceed the reserved read throughput that currently remains. Even if no limit is set in the GetRange request, a next_start_primary_key may still appear in the response. Therefore, when using GetRange, you must check whether the response has a next_start_primary_key to be processed.

Capacity unit consumption:

- The number of read capacity units consumed by the GetRange operation is the sum of the size of primary keys of all data rows in the query range, and the size of actually read attribute columns divided by 4 KB, and then rounded up. For more information about how to calculate the data size, see Billing.

- If the request times out and the results are undefined, a capacity unit may or may not be consumed.
- If an internal error code is returned (HTTP status code: 5XX), no capacity units are consumed. If other errors are returned, one read capacity unit is consumed.

## 1.5 BatchWriteRow

BatchWriteRow inserts, modifies, or deletes several data rows in one or more tables.

It is essentially a set of multiple PutRow, UpdateRow, and DeleteRow operations . Each operation is executed, returns results independently, and independently consumes capacity units.

Compared to the execution of a large number of write operations, the use of the BatchWriteRow operation can reduce the request response time and increase the data write rate.

Request structure:

```
message BatchWriteRowRequest {
    repeated TableInBatchWriteRowRequest tables = 1;
}
```

tables :

- Type: repeated TableInBatchWriteRowRequest
- Required parameter: Yes
- Specifies the information of rows that require write operations.

- An error is returned if any of the following conditions occur:
  - Any table in `tables` does not exist.
  - `tables` contains tables with the same name.
  - The name of any table in `tables` does not comply with the #unique_18.
  - The primary key is not specified for any row in `tables`, the primary key column name does not comply with the conventions, or the primary key column type is incorrect.
  - For any attribute column in `tables`, the column name does not comply with the #unique_18.
  - Any row in `tables` has an attribute column with the same name as the primary key column.
  - The value of any primary key or attribute column in `tables` exceeds the #unique_19.
  - Any table in `tables` contains rows with identical primary keys.
  - The total number of rows for all `tables` in tables exceeds 200 or the total size of the data contained exceeds 1M.
  - If any table in `tables` contains no rows, the OTSParameterInvalidException error is returned.
  - In `tables`, any PutRowInBatchWriteRowRequest contains more than 1024 columns.
  - In `tables`, any UpdateRowInBatchWriteRowRequest contains more than 1024 ColumnUpdate objects.

Response message structure:

```
message BatchWriteRowResponse {
    repeated TableInBatchWriteRowResponse tables = 1;
}
```

`tables`:

- Type: TableInBatchWriteRowResponse
- The response information corresponding to the operations for each table, including whether or not execution was successful, error codes, and the capacity units consumed.

- The TableInBatchWriteRowResponse object order in the response message is the same as the TableInBatchWriteRowRequest object order in BatchWriteRowRequest . The RowInBatchWriteRowResponse object orders contained in put_rows, update_rows, and delete_rows in each TableInBatchWriteRowRequest are the same as the respective PutRowInBatchWriteRowRequest, UpdateRowInBatchWriteRowRequest, and DeleteRowInBatchWriteRowRequest object orders contained in put_rows, update_rows, and delete_rows in TableInBatchWriteRowRequest.
- If a row fails to be read, the row's is_ok value in RowInBatchWriteRowResponse is false.

 Note:
At the row level, the BatchWriteRow operation may partially fail. In this case, it still returns an HTTP status code of 200, but the application must check errors in RowInBatchWriteRowResponse to confirm the execution results for each row and then proceed accordingly.

Capacity unit consumption:

- If the entire operation fails, no capacity units are consumed.
- If request time-out occurs and the results are undefined, a capacity unit may or may not be consumed.
- In other situations, each PutRowInBatchWriteRowRequest, UpdateRowInBatchWriteRowRequestDelete, and RowInBatchWriteRowRequest operation corresponds to a separate write operation when the number of write capacity units are counted. For more information, see PutRow, UpdateRow and DeleteRow .

## 1.6 CreateTable

CreateTable creates a table based on the given table structure information.

Request structure:

```
message CreateTabl eRequest {
    required TableMeta table_meta = 1 ;
    required ReservedTh roughput reserved_t hroughput = 2 ;
    optional TableOptio ns table_opti ons = 3 ;
    optional StreamSpec ification stream_spe c = 5 ;
}
```

table_meta :

- Type: TableMeta
- Required parameter: Yes
- The structure information of the table to be created. The table_name must be unique within the instance. In primary_key, the number of ColumnSchema must range from one to four, the ColumnSchema names must comply with #unique_18, and the type value can only be STRING, INTEGER, or BINARY.
- Once a table is created, its Schema cannot be modified.

```
reserved_t  hroughput :
```

- Type: ReservedThroughput
- Required parameter: Yes
- The initial reserved read/write throughput for the table to be created. The maximum reserved throughput is 5,000 for each table.
- The reserved read/write throughput settings for a table can be modified dynamically by using the UpdateTable operation.

```
table_opti  ons :
```

- Type: TableOptions
- Required parameter: Yes
- Sets TimeToLive and MaxVersions.

```
stream_spe  c :
```

- Type: StreamSpecification
- Required parameter: No
- Specifies whether to enable Stream-related attributes.

Response message structure:

```
 message   CreateTabl  eResponse  {
}
```

Note:

- No read/write operation can be performed on a table immediately after it is created. Generally, a read/write operation can be performed on a new table one minute after it is created.
- A single instance can contain up to 64 tables.

## 1.7 ListTable

ListTable obtains the names of all tables under the current instance.

Request structure:

```
message ListTableRequest {
}
```

Response message structure:

```
message ListTableResponse {
    repeated string table_names = 1;
}
```

table_names:

- Type: repeated string
- The names of all tables under the current instance.

> **Note:**
> If a table has been newly created, its table name appears in ListTableResponse. However, no read/write operation can be performed on a table immediately after it is created. Generally, a read/write operation can be performed on a new table one minute after it is created.

## 1.8 DeleteTable

DeleteTable deletes the specified table under this instance.

Request structure:

```
message DeleteTableRequest {
    required string table_name = 1;
}
```

table_name:

- Type: string
- Required Parameter: Yes
- The name of the table to be deleted.

Response message structure:

```
message DeleteTableResponse {
```

```
}
```

If the DeleteTable response does not contain an error, then the table has been deleted .

## 1.9 UpdateTable

UpdateTable updates the reserved read or write throughput settings of the specified table. New settings take effect within one minute of a successful update.

Request structure:

```
message UpdateTabl eRequest {
    required string table_name = 1 ;
    optional ReservedTh roughput reserved_t hroughput = 2 ;
    optional TableOptio ns table_opti ons = 3 ;
    optional StreamSpec ification stream_spe c = 4 ;
}
```

`table_name :`

- Type: String
- Required parameter: Yes
- The name of the table for which the reserved read/write throughput settings are to be modified.

`reserved_t hroughput :`

- Type: ReservedThroughput
- Required parameter: Yes
- The reserved read/write throughput settings to be modified for the table. These settings take effect within one minute.
- You may modify the table’s reserved read/write throughput settings as required.
- In capacity_unit, read and write must at least have a non-null value. Otherwise, the request fails and an error is returned.

`table_opti ons :`

- Type: TableOptions
- Required parameter: Yes
- Sets TimeToLive and MaxVersions.

`StreamSpec ification`

- Type: StreamSpecification
- Required parameter: No
- Specifies whether to enable Stream-related attributes.

Response message structure:

```
message UpdateTabl eResponse {
    required ReservedTh roughputDe tails reserved_t
hroughput_ details = 1 ;
    required TableOptio ns table_opti ons = 2 ;
}
```

```
capacity_u nit_detail s :
```

- Type: ReservedThroughputDetails
- After the update, in addition to containing the current reserved read/write throughput settings, the table's reserved read/write throughput settings information also contains the time these settings were last updated and the number of times they have been lowered for the current date.

Note:

- The minimum time interval between adjustments to a table's reserved read/write throughput is two minutes. If the current UpdateTable operation is requested within two minutes of the previous operation, it is rejected.
- A table's reserved read/write throughput can be raised or lowered an unlimited number of times within a single day (from 00:00:00 to 00:00:00 the next day in UTC time).

```
table_opti ons :
```

- Type: TableOptions
- The latest value of the table_options parameter after the modification.

## 1.10 DescribeTable

DescribeTable queries the structure information and the reserved read/write throughput value of the specified table.

Request structure:

```
message DescribeTa bleRequest {
    required string table_name = 1 ;
```

```
}
```

```
table_name :
```

- Type: String
- Required parameter: Yes
- The name of the table to be queried.

Response message structure:

```
message DescribeTa bleRespons e {
    required TableMeta table_meta = 1 ;
    required ReservedTh roughputDe tails reserved_t
hroughput_ details = 2 ;
    required TableOptio ns table_opti ons = 3 ;
    optional StreamDeta ils stream_det ails = 5 ;
    repeated bytes shard_spli ts = 6 ;
}
```

```
table_meta :
```

- Type: TableMeta
- The table's schema, which is the same as the Schema given at the time of table creation.

```
reserved_t hroughput_ details :
```

- Type: ReservedThroughputDetails
- The table's reserved read/write throughput settings information. In addition to the current reserved read/write throughput settings, it also contains the time these settings were last updated and the number of times they have been lowered for the current date.

```
table_opti ons :
```

- Type: TableOptions
- The latest value of the table_options parameter.

```
StreamSpec ification :
```

- Type: StreamSpecification
- Required parameter: No
- Specifies whether to enable Stream-related attributes.

```
shard_spli ts :
```

- Type: bytes

- The split points of all shards in the current table.

## 1.11 ComputeSplitPointsBySize

Logically splits data in a table into several shards whose sizes are close to the specified size, and returns the split points between the shards and prompt about machines where the shards are located. This API is generally used for execution plans like concurrency of plans on a computing engine.

### Request structure

```
message ComputeSplitPointsBySizeRequest {
    required string table_name = 1;
    required int64 split_size = 2; // in 100MB
}
```

`table_name`:

- Type: string
- Required parameter: Yes
- The name of the table holding the data to be split.

`split_size`:

- Type: int64
- Required parameter: Yes
- The approximate size of each shard, in the unit of 100 MB.

### Response message structure

```
message ComputeSplitPointsBySizeResponse {
    required ConsumedCapacity consumed = 1;
    repeated PrimaryKeySchema schema = 2;

   /**
    * Split points between splits in an increasing
order
    *
    * A split is a consecutive range of primary
keys,
    * whose data size is about split_size specified in
  the request.
    * The size could be hard to be precise.
    *
    * A split point is an array of primary-key
column w.r.t. table schema,
    * which is never longer than that of table
schema.
    * Tailing -inf will be omitted to reduce
transmission payloads.
    */
```

```
    repeated bytes split_poin ts = 3 ;

   /**
    * Locations where splits lies in .
    *
    * By the managed nature of TableStore , these
locations are no more than hints .
    * If a location is not suitable to be seen , an
  empty string will be placed .
    */
    message SplitLocat ion {
        required string location = 1 ;
        required sint64 repeat = 2 ;
    }
    repeated SplitLocat ion locations = 4 ;
}
```

`consumed :`

- Type: ConsumedCapacity
- The service capacity units consumed by this request.

`schema :`

- Type: PrimaryKeySchema
- The table’s schema, same as the schema given at the time of table creation.

`split_poin ts :`

- Type: repeated bytes
- The split points between shards. The split points must increase monotonically between the shards. Each split point is a Plainbuffer-encoded line and contains only primary keys. The last -inf of each split point is not transmitted for a smaller volume of data transmitted.

`locations :`

- Type: repeated SplitLocation
- The prompt about the machines where the split points are located. It can be null.

For example, a table contains three columns of primary keys, where the first column is of string type. After this API is called, five shards are obtained, which are: `(- inf ,- inf ,- inf )` to `(" a ",- inf ,- inf )`, `(" a ",- inf ,- inf )` to `(" b ",- inf ,- inf )`, `(" b ",- inf ,- inf )` to `(" c ",- inf ,- inf )`, `(" c ",- inf ,- inf )` to `(" d ",- inf ,- inf )` and `(" d ",- inf ,- inf )` to `(+ inf ,+ inf ,+ inf )`. The first three shards are located in “machine-A”, while the last two in “machine-B”. In this case, split_points is (example) `[(" a "),(" b "),(" c`

`"),(" d ")]`, while locations is (example) `" machine - A "* 3 , " machine - B "* 2 `.

Capacity unit consumption

The number of consumed read service capacity units is the same as that of the shards . No write service capacity unit is consumed.

## 1.12 ListStream

ListStream obtains information about the stream that is enabled for all tables on the current instance.

Request structure:

```
message ListStream Request {
  optional string table_name = 1 ;
}
```

`table_name :`

- Type: optional string
- The name of the table for which the current stream is enabled.

Response message structure:

```
message ListStream Response {
    repeated Stream streams = 1 ;
}

message Stream {
    required string stream_id = 1 ;
    required string table_name = 2 ;
    required int64 creation_t ime = 3 ;
}
```

`stream_id :`

- Type: required string
- The ID of the current stream.

`table_name :`

- Type: required string
- The name of the table for which the current stream is enabled.

`creation_t ime :`

- Type: required int64

- The time when the current stream is enabled.

## 1.13 DescribeStream

DescribeStream obtains information about the shards of the current stream.

Request structure:

```
message DescribeStreamRequest {
    required string stream_id = 1;
    optional string inclusive_start_shard_id = 2;
    optional int32 shard_limit = 3;
}
```

```
stream_id:
```

- Type: required string
- The ID of the current stream.

```
inclusive_start_shard_id:
```

- Type: required string
- The ID of the start shard in a query.

```
shard_limit:
```

- Type: required string
- The upper limit of the returned shard quantity in a single query.

Response message structure:

```
message DescribeStreamResponse {
    required string stream_id = 1;
    required int32 expiration_time = 2;
    required string table_name = 3;
    required int64 creation_time = 4;
    required StreamStatus stream_status = 5;
    repeated StreamShard shards = 6;
    optional string next_shard_id = 7;
}

message StreamShard {
    required string shard_id = 1;
    optional string parent_id = 2;
    optional string parent_sibling_id = 3;
}
```

```
stream_id:
```

- Type: required string
- The ID of the current stream.

expiration_time :

- Type: required int32
- The expiration time of the stream.

table_name :

- Type: required string
- The name of the table for which the current stream is enabled.

creation_time :

- Type: required int32
- The time when the current stream is enabled.

stream_status :

- Type: required StreamStatus
- The status of the current stream, which can be enabling or active.

shards :

- Type: required StreamShard
- The information about the streamShard, including the shard ID, parent shard ID , and information about the neighbor shard of the parent shard (applicable when the parent shard is merged).

next_shard_id :

- Type: optional string
- The start ID of the next shard in a paging query.

Note:

Data reading for the parent shard must be completed before data of the current shard is read.

## 1.14 GetShardIterator

GetShardIterator obtains the start iterator for reading the record information of the current shard.

Request structure:

```
message GetShardIteratorRequest {
    required string stream_id = 1;
```

```
    required   string   shard_id  =  2 ;
}
```

stream_id :

- Type: required string
- The ID of the current stream.

shard_id :

- Type: required string
- The ID of the current shard.

Response message structure:

```
 message   GetShardIt  eratorResp  onse  {
    required   string   shard_iter  ator  =  1 ;
}
```

shard_iter  ator :

- Type: required string
- The start iterator for reading the record of the current shard.

## 1.15 GetStreamRecord

GetStreamRecord obtains the incremental content of the current shard.

Request structure:

```
 message   GetStreamR  ecordReque  st  {
    required   string   shard_iter  ator  =  1 ;
    optional   int32   limit  =  2 ;
}
```

shard_iter  ator :

- Type: required string
- The iterator for reading the current shard.

Response message structure:

```
 message   GetStreamR  ecordRespo  nse  {
    message   StreamReco  rd  {
        required   ActionType   action_typ  e  =  1 ;
        required   bytes   record  =  2 ;
    }
    repeated   StreamReco  rd   stream_rec  ords  =  1 ;
    optional   raw_string   next_shard  _iterator  =  2 ;
    optional   ConsumedCa  pacity   consumed  =  3 ;
```

```
}
```

StreamReco rd :

- Type: repeated StreamRecord
- The record entry for reading the current shard.

shard_iter ator :

- Type: required string
- The iterator for reading the current shard in the next operation.

consumed :

- Type: ConsumedCapacity
- The value of capacity units consumed by this operation.
- The sum of data in all rows of a table is divided by 4 KB, and then rounded up. Stream read CUs are equivalent to the rounded up value. For more information about how to calculate a row's data volume, see Data storage.

## 1.16 Table Store ProtocolBuffer Message Definitions

The detailed definitions of table_store.proto and table_store_filter.proto are as follows:

table_store.proto

```
package com . alicloud . openservic es . tablestore . core .
protocol ;

message Error {
    required string code = 1 ;
    optional string message = 2 ;
}

enum PrimaryKey Type {
    INTEGER = 1 ;
    STRING = 2 ;
    BINARY = 3 ;
}

enum PrimaryKey Option {
    AUTO_INCRE MENT = 1 ;
}

message PrimaryKey Schema {
    required string name = 1 ;
    required PrimaryKey Type type = 2 ;
    optional PrimaryKey Option option = 3 ;
}

message TableOptio ns {
```

```
    optional   int32   time_to_li  ve  =  1 ; //  It   can   be
 dynamicall  y   modified
    optional   int32   max_versio  ns  =  2 ; //  It   can   be
 dynamicall  y   modified
  optional   int64   deviation_  cell_versi  on_in_sec  =  5 ; //  It
   can   be   dynamicall  y   modified
}

 message   TableMeta  {
    required   string   table_name  =  1 ;
    repeated   PrimaryKey  Schema   primary_ke  y  =  2 ;
}

 enum   RowExisten  ceExpectat  ion  {
    IGNORE  =  0 ;
    EXPECT_EXI  ST  =  1 ;
    EXPECT_NOT  _EXIST  =  2 ;
}

 message   Condition  {
    required   RowExisten  ceExpectat  ion   row_existe  nce  =  1 ;
    optional   bytes   column_con  dition       =  2 ;
}

 message   CapacityUn  it  {
    optional   int32   read  =  1 ;
    optional   int32   write  =  2 ;
}

 message   ReservedTh  roughputDe  tails  {
    required   CapacityUn  it   capacity_u  nit  =  1 ; //  It
 indicates   the   value   of   the   current   reserved   throughput
   of   the   table
    required   int64   last_incre  ase_time  =  2 ; //  It
 indicates   the   last   time   that   the   reserved   throughput
 was   raised
    optional   int64   last_decre  ase_time  =  3 ; //  It
 indicates   the   last   time   that   the   reserved   throughput
 was   lowered
}

 message   ReservedTh  roughput  {
    required   CapacityUn  it   capacity_u  nit  =  1 ;
}

 message   ConsumedCa  pacity  {
    required   CapacityUn  it   capacity_u  nit  =  1 ;
}

/* ##############################################   CreateTabl  e
 ############################################## */
/**
 *  table_meta   is   used   to   store   unmodifiab  le   schema
   attributes   in   a   table .  The   ReservedTh  roughput   and
   TableOptio  ns   attributes   that   can   be   modified   are
 independen  tly   stored   as   the   parameters   for   UpdateTabl
 e .
 *  message   CreateTabl  eRequest  {
 *          required   TableMeta   table_meta  =  1 ;
 *          required   ReservedTh  roughput   reserved_t  hroughput
 =  2 ;
 *          required   TableOptio  ns   table_opti  ons  =  3 ;
 * }
 */
```

```
 message   CreateTabl  eRequest  {
     required   TableMeta   table_meta  =  1 ;
     required   ReservedTh  roughput   reserved_t  hroughput  =  2 ;
     optional   TableOptio  ns   table_opti  ons  =  3 ;
}

 message   CreateTabl  eResponse  {
}

/*
 ################################################################################
 */


/* ##############################################   UpdateTabl  e
 ############################################## */
 message   UpdateTabl  eRequest  {
     required   string   table_name  =  1 ;
     optional   ReservedTh  roughput   reserved_t  hroughput  =  2 ;
     optional   TableOptio  ns   table_opti  ons  =  3 ;
}

 message   UpdateTabl  eResponse  {
     required   ReservedTh  roughputDe  tails   reserved_t
 hroughput_  details  =  1 ;
     required   TableOptio  ns   table_opti  ons  =  2 ;
}
/*
 ################################################################################
 */

/* ##############################################   DescribeTa  ble
 ############################################## */
 message   DescribeTa  bleRequest  {
     required   string   table_name  =  1 ;
}

 message   DescribeTa  bleRespons  e  {
     required   TableMeta   table_meta  =  1 ;
     required   ReservedTh  roughputDe  tails   reserved_t
 hroughput_  details  =  2 ;
     required   TableOptio  ns   table_opti  ons  =  3 ;
     repeated   bytes   shard_spli  ts  =  6 ;
}
/*
 ################################################################################
 */

/* ##############################################   ListTable
 ############################################## */
 message   ListTableR  equest  {
}

/**
 *  Only   a   simple   name   list   is   returned   currently .
 */
 message   ListTableR  esponse  {
     repeated   string   table_name  s  =  1 ;
}
/*
 ################################################################################
 */
```

```
/* ##############################################   DeleteTabl  e
 ############################################## */
 message   DeleteTabl  eRequest  {
     required   string   table_name  =  1 ;
}

 message   DeleteTabl  eResponse  {
}

/* ##############################################   UnloadTabl  e
 ############################################## */
 message   UnloadTabl  eRequest  {
     required   string   table_name  =  1 ;
}

 message   UnloadTabl  eResponse  {

}
/*
 ##########################################################################################
 */

/**
 *  The   minimum   and   maximum   values   of   the   time   stamp
 are   0   and   INT64 .  MAX ,  respective  ly .
 *  1 .  To   query   data   of   a   time   range ,  specify
 start_time   and   end_time .
 *  2 .  To   query   data   of   a   specific   time   stamp ,
 specify   specific_t  ime .
 */
 message   TimeRange  {
     optional   int64   start_time  =  1 ;
     optional   int64   end_time  =  2 ;
     optional   int64   specific_t  ime  =  3 ;
}

/* ##############################################   GetRow
 ############################################## */

 enum   ReturnType  {
     RT_NONE  =  0 ;
     RT_PK  =  1 ;
}

 message   ReturnCont  ent  {
     optional   ReturnType   return_typ  e  =  1 ;
}

/**
 *  1 .  You   can   specify   the   time   stamp   range   or   time
   to   read   the   columns   of   the   specified   version .
 *  2 .  Intra - row   breakpoint  s   are   not   supported
 currently .
 */
 message   GetRowRequ  est  {
     required   string   table_name  =  1 ;
     required   bytes   primary_ke  y  =  2 ; //  encoded   as
 InplaceRow  ChangeSet ,  but   only   has   primary   key
     repeated   string   columns_to  _get  =  3 ; //  If   it   is
 not   specified ,  all   columns   are   read
     optional   TimeRange   time_range  =  4 ;
     optional   int32   max_versio  ns  =  5 ;
     optional   bytes   filter  =  7 ;
     optional   string   start_colu  mn  =  8 ;
```

```
    optional   string   end_column  =  9 ;
    optional   bytes   token  =  10 ;
}

 message   GetRowResp  onse  {
    required   ConsumedCa  pacity   consumed  =  1 ;
    required   bytes   row  =  2 ; //  encoded   as   InplaceRow
 ChangeSet
    optional   bytes   next_token  =  3 ;
}
/*
 ###############################################################################
 */

/* ##############################################   UpdateRow
 ############################################## */
 message   UpdateRowR  equest  {
    required   string   table_name  =  1 ;
    required   bytes   row_change  =  2 ;
    required   Condition   condition  =  3 ;
    optional   ReturnCont  ent   return_con  tent  =  4 ;
}

 message   UpdateRowR  esponse  {
    required   ConsumedCa  pacity   consumed  =  1 ;
    optional   bytes   row  =  2 ;
}
/*
 ###############################################################################
 */

/* ##############################################   PutRow
 ############################################## */
/**
 *  You   can   set   a   time   stamp   for   each   column   instead
   of   setting   a   unified   time   stamp   for   the   entire
 row .
 */
 message   PutRowRequ  est  {
    required   string   table_name  =  1 ;
    required   bytes   row  =  2 ; //  encoded   as   InplaceRow
 ChangeSet
    required   Condition   condition  =  3 ;
    optional   ReturnCont  ent   return_con  tent  =  4 ;
}

 message   PutRowResp  onse  {
    required   ConsumedCa  pacity   consumed  =  1 ;
    optional   bytes   row  =  2 ;
}
/*
 ###############################################################################
 */

/* ##############################################   DeleteRow
 ############################################## */
/**
 *  Table   Store   only   supports   deleting   all   versions   of
 all   columns   in   a   specified   row .
 */
 message   DeleteRowR  equest  {
    required   string   table_name  =  1 ;
    required   bytes   primary_ke  y  =  2 ; //  encoded   as
 InplaceRow  ChangeSet ,  but   only   has   primary   key
```

```
    required   Condition   condition  =  3 ;
    optional   ReturnCont  ent   return_con  tent  =  4 ;
}

 message   DeleteRowR  esponse  {
    required   ConsumedCa  pacity   consumed  =  1 ;
    optional   bytes   row  =  2 ;
}
/*
 ################################################################################
 */

/* ##############################################   BatchGetRo  w
 ############################################## */
 message   TableInBat  chGetRowRe  quest  {
    required   string   table_name  =  1 ;
    repeated   bytes   primary_ke  y  =  2 ; //  encoded   as
 InplaceRow  ChangeSet ,  but   only   has   primary   key
    repeated   bytes   token  =  3 ;
    repeated   string   columns_to  _get  =  4 ;  //  If   it   is
 not   specified ,  all   columns   are   read
    optional   TimeRange   time_range  =  5 ;
    optional   int32   max_versio  ns  =  6 ;
    optional   bytes   filter  =  8 ;
    optional   string   start_colu  mn  =  9 ;
    optional   string   end_column  =  10 ;
}

 message   BatchGetRo  wRequest  {
    repeated   TableInBat  chGetRowRe  quest   tables  =  1 ;
}

 message   RowInBatch  GetRowResp  onse  {
    required   bool   is_ok  =  1 ;
    optional   Error   error  =  2 ;
    optional   ConsumedCa  pacity   consumed  =  3 ;
    optional   bytes   row  =  4 ; //  encoded   as   InplaceRow
 ChangeSet
    optional   bytes   next_token  =  5 ;
}

 message   TableInBat  chGetRowRe  sponse  {
    required   string   table_name  =  1 ;
    repeated   RowInBatch  GetRowResp  onse   rows  =  2 ;
}

 message   BatchGetRo  wResponse  {
    repeated   TableInBat  chGetRowRe  sponse   tables  =  1 ;
}
/*
 ################################################################################
 */

/* ##############################################   BatchWrite  Row
 ############################################## */

 enum   OperationT  ype  {
    PUT  =  1 ;
    UPDATE  =  2 ;
    DELETE  =  3 ;
}

 message   RowInBatch  WriteRowRe  quest  {
    required   OperationT  ype   type  =  1 ;
```

```
    required bytes row_change = 2 ; // encoded as
 InplaceRow ChangeSet
    required Condition condition = 3 ;
    optional ReturnCont ent return_con tent = 4 ;
}

 message TableInBat chWriteRow Request {
    required string table_name = 1 ;
    repeated RowInBatch WriteRowRe quest rows = 2 ;
}

 message BatchWrite RowRequest {
    repeated TableInBat chWriteRow Request tables = 1 ;
}

 message RowInBatch WriteRowRe sponse {
    required bool is_ok = 1 ;
    optional Error error = 2 ;
    optional ConsumedCa pacity consumed = 3 ;
    optional bytes row = 4 ;
}

 message TableInBat chWriteRow Response {
    required string table_name = 1 ;
    repeated RowInBatch WriteRowRe sponse rows = 2 ;
}

 message BatchWrite RowRespons e {
    repeated TableInBat chWriteRow Response tables = 1 ;
}
/*
 ################################################################################
 */

/* ############################################ GetRange
 ############################################ */
 enum Direction {
    FORWARD = 0 ;
    BACKWARD = 1 ;
}

 message GetRangeRe quest {
    required string table_name = 1 ;
    required Direction direction = 2 ;
    repeated string columns_to _get = 3 ; // If it is
 not specified , all columns are read
    optional TimeRange time_range = 4 ;
    optional int32 max_versio ns = 5 ;
    optional int32 limit = 6 ;
    required bytes inclusive_ start_prim ary_key = 7 ; //
 encoded as InplaceRow ChangeSet , but only has primary
 key
    required bytes exclusive_ end_primar y_key = 8 ; //
 encoded as InplaceRow ChangeSet , but only has primary
 key
    optional bytes filter = 10 ;
    optional string start_colu mn = 11 ;
    optional string end_column = 12 ;
    optional bytes token = 13 ;
}

 message GetRangeRe sponse {
    required ConsumedCa pacity consumed = 1 ;
```

```
    required bytes rows = 2 ; // encoded as InplaceRow
 ChangeSet
    optional bytes next_start _primary_k ey = 3 ; // If it
   is null , all data has been read . It is encoded
 as InplaceRow ChangeSet , but only has primary key
    optional bytes next_token = 4 ;
}

/* ################### ComputeSpl itPointsBy Size
 ################### */
 message ComputeSpl itPointsBy SizeReques t {
    required string table_name = 1 ;
    required int64 split_size = 2 ; // in 100MB
}

 message ComputeSpl itPointsBy SizeRespon se {
    required ConsumedCa pacity consumed = 1 ;
    repeated PrimaryKey Schema schema = 2 ;

   /**
    * Split points between splits , in the increasing
 order
    *
    * A split is a consecutiv e range of primary
 keys ,
    * whose data size is about split_size specified in
   the request .
    * The size could be hard to be precise .
    *
    * A split point is an array of primary - key
 column w . r . t . table schema ,
    * which is never longer than that of table
 schema .
    * Tailing - inf will be omitted to reduce
 transmissi on payloads .
    */
    repeated bytes split_poin ts = 3 ;

   /**
    * Locations where splits lies in .
    *
    * By the managed nature of TableStore , these
 locations are no more than hints .
    * If a location is not suitable to be seen , an
   empty string will be placed .
    */
     message SplitLocat ion {
        required string location = 1 ;
        required sint64 repeat = 2 ;
    }
     repeated SplitLocat ion locations = 4 ;
}
```

table_store_filter.proto

```
 package com . alicloud . openservic es . tablestore . core .
 protocol ;

 enum FilterType {
    FT_SINGLE_ COLUMN_VAL UE = 1 ;
    FT_COMPOSI TE_COLUMN_ VALUE = 2 ;
    FT_COLUMN_ PAGINATION = 3 ;
}
```

```
 enum   Comparator  Type  {
     CT_EQUAL  =  1 ;
     CT_NOT_EQU  AL  =  2 ;
     CT_GREATER  _THAN  =  3 ;
     CT_GREATER  _EQUAL  =  4 ;
     CT_LESS_TH  AN  =  5 ;
     CT_LESS_EQ  UAL  =  6 ;
}

 message   SingleColu  mnValueFil  ter  {
     required   Comparator  Type   comparator  =  1 ;
     required   string   column_nam  e  =  2 ;
     required   bytes   column_val  ue  =  3 ;
     required   bool   filter_if_  missing  =  4 ;
     required   bool   latest_ver  sion_only  =  5 ;
}

 enum   LogicalOpe  rator  {
     LO_NOT  =  1 ;
     LO_AND  =  2 ;
     LO_OR  =  3 ;
}

 message   CompositeC  olumnValue  Filter  {
     required   LogicalOpe  rator   combinator  =  1 ;
     repeated   Filter   sub_filter  s  =  2 ;
}

 message   ColumnPagi  nationFilt  er  {
     required   int32   offset  =  1 ;
     required   int32   limit  =  2 ;
}

 message   Filter  {
     required   FilterType   type  =  1 ;
     required   bytes   filter  =  2 ;  //  Serialized   string   of
 filter   of   the   type
}
```

# 2 Data Types

## 2.1 ActionType

ActionType specifies the operation type in the response message of the `GetStreamRecord` operation. Operation types include:

- PUT_ROW, which indicates that the operation type is PutRow.
- UPDATE_ROW, which indicates that the operation type is UpdateRow.
- DELETE_ROW, which indicates that the operation type is DeleteRow.

### Enumeration value list

```
enum ActionType {
    PUT_ROW = 1;
    UPDATE_ROW = 2;
    DELETE_ROW = 3;
}
```

### Related operations

GetStreamRecord

## 2.2 CapacityUnit

CapacityUnit indicates the value of the capacity units consumed in an operation, or a table's reserved read/write throughput.

### Data structure

```
message CapacityUnit {
    optional int32 read = 1;
    optional int32 write = 2;
}
```

`read`:

- Type: int32
- The read capacity units consumed by this operation or the reserved read throughput for this table.

`write`:

- Type: int32

- The write capacity units consumed by this operation or the reserved write throughput for this table.

Related operations

UpdateRow

BatchWriteRow

## 2.3 ComparatorType

ComparatorType is an relational operator. It includes the following enumeration types:

- CT_EQUAL, which indicates equal.
- CT_NOT_EQUAL, which indicates not equal.
- CT_GREATER_THAN, which indicates greater than.
- CT_GREATER_EQUAL, which indicates not less than.
- CT_LESS_THAN, which indicates less than.
- CT_LESS_EQUAL, which indicates not greater than.

Enumeration value list

```
enum ComparatorType {
    CT_EQUAL = 1;
    CT_NOT_EQUAL = 2;
    CT_GREATER_THAN = 3;
    CT_GREATER_EQUAL = 4;
    CT_LESS_THAN = 5;
    CT_LESS_EQUAL = 6;
}
```

## 2.4 PrimaryKeyType

PrimaryKeyType specifies the type of the primary key.

Enumeration data type

- INTEGER: Integer
- STRING: String
- BINARY: Binary

```
enum PrimaryKeyType {
    INTEGER = 1;
    STRING = 2;
    BINARY = 3;
```

```
}
```

## 2.5 Condition

Condition specifies the row judgment conditions in PutRow, UpdateRow, and DeleteRow. It currently contains row_existence and column_condition.

### Data structure

```
message Condition {
    required RowExisten ceExpectat ion row_existe nce = 1 ;
    optional bytes column_con dition = 2 ;
}
```

`row_existe nce :`

- Type: RowExistenceExpectation
- The row existence check settings for this row.

`column_con dition :`

- Type: Bytes
- The column condition settings. It indicates the bytes after the Filter is serialized in Protobuf format.

### Related operations

PutRow

UpdateRow

DeleteRow

BatchWriteRow

## 2.6 ConsumedCapacity

ConsumedCapacity indicates capacity units consumed by an operation.

### Data structure

```
message ConsumedCa pacity {
    required CapacityUn it capacity_u nit = 1 ;
}
```

`capacity_u nit :`

- Type: CapacityUnit

- The value of capacity units consumed by this operation.

## 2.7 Direction

Direction indicates the data query order in the GetRange operation. The two Direction operations are:

- FORWARD, which indicates that the query proceeds from the smallest to the largest primary key.
- BACKWARD, which indicates that the query proceeds from the largest to the smallest primary key.

### Enumeration value list

```
enum Direction {
    FORWARD = 0;
    BACKWARD = 1;
}
```

### Related operations

GetRange

## 2.8 Error

Error can indicate an error message in the response returned when an operation fails, and indicate an error for an individual row request in the BatchGetRow and BatchWriteRow operations.

### Data structure

```
Error {
    required string code = 1;
    optional string message = 2;
}
```

`code :`

- Type: string
- The error code for the current individual row operation.

`message :`

- Type: string
- The error message for the current individual row operation.

Related operations

BatchGetRow

BatchWriteRow

## 2.9 FilterType

FilterType specifies the type for condition updating or filtering. The types include:

- FT_SINGLE_COLUMN_VALUE, which is a single-column condition.
- FT_COMPOSITE_COLUMN_VALUE, which is a multi-column composite condition.
- FT_COLUMN_PAGINATION, which is a condition for wide-row read.

Enumeration value list

```
enum FilterType {
    FT_SINGLE_ COLUMN_VAL UE = 1 ;
    FT_COMPOSI TE_COLUMN_ VALUE = 2 ;
    FT_COLUMN_ PAGINATION = 3 ;
}
```

## 2.10 LogicalOperator

LogicalOperator includes the following enumeration types:

- LO_NOT, which indicates NOT.
- LO_AND, which indicates AND.
- LO_OR, which indicates OR.

Enumeration value list

```
enum LogicalOpe rator {
    LO_NOT = 1 ;
    LO_AND = 2 ;
    LO_OR = 3 ;
}
```

## 2.11 OperationType

In UpdateRow, OperationType indicates the modification method for one column. The types of operations include:

- PUT, which indicates that a column is inserted or this column’s data is overwritten.
- UPDATE, which indicates that a column is updated.

- DELETE, which indicates that a column is deleted.

Enumeration value list

```
enum OperationType {
    PUT = 1;
    UPDATE = 2;
    DELETE = 3;
}
```

## 2.12 PartitionRange

PartitionRange specifies the shard range.

```
message PartitionRange {
    required bytes begin = 1; // encoded as SQLVariant
    required bytes end = 2; // encoded as SQLVariant
}
```

`begin :`

- Type: Bytes
- The starting key of the shard. The shard is a left-closed right-open interval, which includes the starting key. It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

`end :`

- Type: Bytes
- The ending key of the shard. The shard is a left-closed right-open interval, which does not include the ending key. It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

## 2.13 PlainBuffer

The PlainBuffer format is defined in Table Store to indicate row data, because it delivers better performance for serialization, and small object resolution, compared to Protocol Buffer.

Format definition

```
plainbuffer = tag_header row1 [ row2 ] [ row3 ]
row = ( pk [ attr ] | [ pk ] attr | pk attr ) [ tag_delete
_marker ] row_checksum ;
pk = tag_pk cell_1 [ cell_2 ] [ cell_3 ]
attr = tag_attr cell1 [ cell_2 ] [ cell_3 ]
cell = tag_cell cell_name [ cell_value ] [ cell_op ] [ cell_ts
] cell_checksum
```

```
cell_name = tag_cell_n ame formated_v alue
cell_value = tag_cell_v alue formated_v alue
cell_op = tag_cell_o p cell_op_va lue
cell_ts = tag_cell_t s cell_ts_va lue
row_checks um = tag_row_ch ecksum row_crc8
cell_check sum = tag_cell_c hecksum row_crc8

formated_v alue = value_type value_len value_data
value_type = int8
value_len = int32

cell_op_va lue = delete_all _version | delete_one _version
cell_ts_va lue = int64
delete_all _version = 0x01 ( 1byte )
delete_one _version = 0x03 ( 1byte )
```

### Tag value

```
tag_header = 0x75 ( 4byte )
tag_pk = 0x01 ( 1byte )
tag_attr = 0x02 ( 1byte )
tag_cell = 0x03 ( 1byte )
tag_cell_n ame = 0x04 ( 1byte )
tag_cell_v alue = 0x05 ( 1byte )
tag_cell_o p = 0x06 ( 1byte )
tag_cell_t s = 0x07 ( 1byte )
tag_delete _marker = 0x08 ( 1byte )
tag_row_ch ecksum = 0x09 ( 1byte )
tag_cell_c hecksum = 0x0A ( 1byte )
```

### ValueType value

The values of value_type in formated_value are as follows:

```
VT_INTEGER = 0x0
VT_DOUBLE = 0x1
VT_BOOLEAN = 0x2
VT_STRING = 0x3
VT_NULL = 0x6
VT_BLOB = 0x7
VT_INF_MIN = 0x9
VT_INF_MAX = 0xa
VT_AUTO_IN CREMENT = 0xb
```

### Calculate the checksum

The basic logic for calculating the checksum is as follows:

- Calculate the name/value/type/time stamp of each cell.
- Calculate the delete in the row. If delete mark exists in the row, supplement a single-byte 1; otherwise, supplement a single-byte 0.
- Because the checksum is calculated for each cell, the cell checksum is used to calculate the row checksum, that is, the CRC operation is only performed on the checksums of cells in the row, not data in the row.

Java implementation:

**Note:**

The following code is from `$ tablestore - 4 . 2 . 1 - sources / com / alicloud / openservic es / tablestore / core / protocol / PlainBuffe rCrc8 . java` . For more information, see Java SDK.

```
public static byte getChecksu m ( byte crc , PlainBuffe
rCell cell ) throws IOExceptio n {

    if ( cell . hasCellNam e ()) {
        crc = crc8 ( crc , cell . getNameRaw Data ());
    }

    if ( cell . hasCellVal ue ()) {
        if ( cell . isPk ()) {
            crc = cell . getPkCellV alue (). getChecksu m ( crc
);
        } else {
            crc = cell . getCellVal ue (). getChecksu m ( crc );
        }
    }

    if ( cell . hasCellTim estamp ()) {
        crc = crc8 ( crc , cell . getCellTim estamp ());
    }

    if ( cell . hasCellTyp e ()) {
        crc = crc8 ( crc , cell . getCellTyp e ());
    }

    return crc ;
}

public static byte getChecksu m ( byte crc , PlainBuffe
rRow row ) throws IOExceptio n {
    for ( PlainBuffe rCell cell : row . getPrimary Key ()) {
        crc = crc8 ( crc , cell . getChecksu m ());
    }

    for ( PlainBuffe rCell cell : row . getCells ()) {
        crc = crc8 ( crc , cell . getChecksu m ());
    }

    byte del = 0 ;
    if ( row . hasDeleteM arker ()) {
        del = ( byte ) 0x1 ;
    }
    crc = crc8 ( crc , del );

    return crc ;
```

```
}
```

Example

A data row contains two primary key columns and four data columns. The primary key types are string and integer, and the data types are string, int, and double. The versions are 1001, 1002, and 1003. A column is also contained to delete all versions.

- Primary key column:
  - [pk1:string:iampk]
  - [pk2:integer:100]
- Attribute column:
  - [column1:string:bad:1001]
  - [column2:integer:128:1002]
  - [column3:double:34.2:1003]
  - [column4:del_all_versions]

Encoding:

```
 < HHeader   starting >[ 0x75 ]
 < rimary   key   column   starting >[ 0x1 ]
 < Cell1 >[ 0x3 ][ 0x4 ][ 0x3 ][ 3 ][ pk1 ][ 0x5 ][ 0x3 ][ 5 ][ iampk
][ _cell_chec  ksum ]
   < Cell2 >[ 0x3 ][ 0x4 ][ 0x3 ][ 3 ][ pk2 ][ 0x5 ][ 0x0 ][ 8 ][ 100
][ _cell_chec  ksum ]
 < Attribute   column   starting >[ 0x2 ]
   < Cell1 >[ 0x3 ][ 0x4 ][ 0x3 ][ 7 ][ column1 ][ 0x5 ][ 0x3 ][ 3 ][
bad ][ 0x7 ][ 1001 ][ _cell_chec  ksum ]
   < Cell2 >[ 0x3 ][ 0x4 ][ 0x3 ][ 7 ][ column2 ][ 0x5 ][ 0x0 ][ 8 ][
128 ][ 0x7 ][ 1002 ][ _cell_chec  ksum ]
   < Cell3 >[ 0x3 ][ 0x4 ][ 0x3 ][ 7 ][ column3 ][ 0x5 ][ 0x1 ][ 8 ][
34 . 2 ][ 0x7 ][ 1003 ][ _cell_chec  ksum ]
   < Cell4 >[ 0x3 ][ 0x4 ][ 0x3 ][ 7 ][ column4 ][ 0x6 ][ 1 ][
_cell_chec  ksum ]
 [ _row_check  _sum ]
```

## 2.14 PrimaryKeyOption

PrimaryKeyOption specifies the attribute value of the primary key. Currently, it only supports AUTO_INCREMENT.

Enumeration value list

```
enum   PrimaryKey  Option  {
    AUTO_INCRE  MENT  =  1 ;
```

```
}
```

## 2.15 RowInBatchGetRowResponse

RowInBatchGetRowResponse indicates a data row in the response message of the BatchGetRow operation.

### Data structure

```
message RowInBatchGetRowResponse {
    required bool is_ok = 1 [default = true];
    optional Error error = 2;
    optional ConsumedCapacity consumed = 3;
    optional bytes row = 4;
  optional bytes next_token = 5;
}
```

`is_ok`:

- Type: Bool
- Determines whether the operation for this row is successful. If the value is true, the row is read successfully and error is invalid. If the value is false, this row fails to be read and row is invalid.

`error`:

- Type: Error
- The error message for this row operation.

`consumed`:

- Type: ConsumedCapacity
- The capacity units consumed by this row operation.

`row`:

- Type: Bytes
- The read data is encoded in Plainbuffer format. For more information, see Plainbuffer encoding.
- If this row does not exist, null is returned.

`next_token`:

- Type: Bytes
- It indicates the starting position of a wide row to be read during the next operation of which is unavailable currently.

### Related operations

BatchGetRow

## 2.16 PrimaryKeySchema

PrimaryKeySchema specifies the attribute value of the primary key.

### Data structure

```
message PrimaryKeySchema {
    required string name = 1;
    required PrimaryKeyType type = 2;
    optional PrimaryKeyOption option = 3;
}
```

`name`:

- Type: String
- The name of the column.

`type`:

- Type: PrimaryKeyType
- The type of the column.

`option`:

- Type: PrimaryKeyOption
- The additional attribute value of the column.

## 2.17 ReservedThroughput

ReservedThroughput indicates the provisioned throughput capacity reserved for read/write value set for a table.

### Data structure

```
message ReservedThroughput {
    required CapacityUnit capacity_unit = 1;
}
```

`capacity_unit`:

- Type: CapacityUnit
- The current reserved read/write throughput value.

### Related operations

CreateTable

UpdateRow

DescribeTable

## 2.18 ReservedThroughputDetails

ReservedThroughputDetails indicates the provisioned throughput capacity reserved for read/write information for a table.

### Data structure

```
message ReservedThroughputDetails {
    required CapacityUnit capacity_unit = 1;
    required int64 last_increase_time = 2;
    optional int64 last_decrease_time = 3;
    required int32 number_of_decreases_today = 4;
}
```

`capacity_unit :`

- Type: CapacityUnit
- The provisioned throughput capacity reserved for read/write value for the specified table.

`last_increase_time :`

- Type: int64
- The last time the provisioned throughput capacity reserved for read/write settings were raised for this table, expressed in UTC time at second level.

`last_decrease_time :`

- Type: int64
- The last time the provisioned throughput capacity reserved for read/write settings were lowered for this table, expressed in UTC time at second level.

`number_of_decreases_today :`

- Type: int32
- The number of times the table's provisioned throughput capacity reserved for read/write settings were lowered on the current calendar day.

Related operations

UpdateTable

DescribeTable

## 2.19 ReturnContent

ReturnContent specifies the content of the returned data.

Data structure

```
message ReturnContent {
    optional ReturnType return_type = 1;
}
```

return_type:

- Type: ReturnType
- The type of the returned data.

## 2.20 ReturnType

ReturnType specifies the type of the returned data.

Enumeration data type

```
enum ReturnType {
    RT_NONE = 0;
    RT_PK = 1;
}
```

- RT_NONE: The default value. It indicates that no value is returned.
- RT_PK: It indicates that the primary key is returned.

## 2.21 RowExistenceExpectation

RowExistenceExpectations includes conditions for row existence in enumeration type. The conditions include:

- IGNORE, which indicates that the row existence check is not performed.
- EXPECT_EXIST, which indicates that the row is expected to exist.
- EXPECT_NOT_EXIST, which indicates that this row is not expected to exist.

Enumeration value list

```
enum RowExistenceExpectation {
```

```
    IGNORE  =  0 ;
    EXPECT_EXI  ST  =  1 ;
    EXPECT_NOT  _EXIST  =  2 ;
}
```

### Related operations

PutRow

UpdateRow

DeleteRow

BatchWriteRow

## 2.22 RowInBatchWriteRowRequest

RowInBatchWriteRowRequest indicates the information of the row to be written, updated, or deleted in the BatchWriteRow operation.

### Data structure

```
 message   RowInBatch  WriteRowRe  quest  {
    required   OperationT  ype   type  =  1 ;
    required   bytes   row_change  =  2 ; //  encoded   as
 InplaceRow  ChangeSet
    required   Condition   condition  =  3 ;
    optional   ReturnCont  ent   return_con  tent  =  4 ;
}
```

`type :`

- Type: OperationType
- The operation type.

`row_change :`

- Type: Bytes
- The row data, which includes the primary key columns and attribute columns. The columns are encoded in Plainbuffer format. For more information, see Plainbuffer encoding.

`condition :`

- Type: Condition
- The value of conditional update, including the row existence condition and column -based condition.

`return_con  tent :`

- Type: ReturnContent
- Required parameter: Yes
- The data type after the row is successfully written. Currently, only the primary key can be returned, which is used for the auto-increment function of the primary key column.

### Related operations

BatchWriteRow

## 2.23 RowInBatchWriteRowResponse

RowInBatchWriteRowResponse indicates the write operation result for a row in the response message of the BatchWriteRow operation.

### Data structure

```
message RowInBatchWriteRowResponse {
    required bool is_ok = 1 [default = true];
    optional Error error = 2;
    optional ConsumedCapacity consumed = 3;
}
```

`is_ok`:

- Type: Bool
- Determines whether the operation for this row is successful. If the value is true, the row is written successfully and error is invalid. If the value is false, the row fails to be written.

`error`:

- Type: Error
- The error message for this row operation.

`consumed`:

- Type: ConsumedCapacity
- The capacity units consumed by this row operation.

### Related operations

BatchWriteRow

## 2.24 StreamDetails

StreamDetails indicates the stream of a table.

### Data structure

```
message StreamDetails {
    required bool enable_stream = 1;
    optional string stream_id = 2;
    optional int32 expiration_time = 3;
    optional int64 last_enable_time = 4;
}
```

`enable_stream`:

- Type: Required bool
- Determines whether the stream is enabled for the table.

`stream_id`:

- Type: Optional string
- The ID of the stream of the table.

`expiration_time`:

- Type: Optional int32
- The expiration time of the stream of the table.

`last_enable_time`:

- Type: Optional int64
- The time for enabling the stream of the table.

### Related operations

DescribeTable

## 2.25 StreamRecord

StreamRecord indicates a data row in the response message of the GetStreamRecord operation.

### Data structure

```
message StreamRecord {
       required ActionType action_type = 1;
       required bytes record = 2;
```

```
}
```

action_typ e :

- Type: Required ActionType
- The operation type of the row.

record :

- Type: Required bytes
- The data content of the row.

### Related operations

GetStreamRecord

## 2.26 StreamSpecification

StreamSpecification indicates the stream of a table.

### Data structure

```
message StreamSpec ification {
    required bool enable_str eam = 1 ;
    optional int32 expiration _time = 2 ;
}
```

enable_str eam :

- Type: Bool
- Determines whether the stream is enabled for the table.

expiration _time :

- Type: int32
- The expiration time of the stream of the table.

### Related operations

CreateTable

DescribeTable

UpdateTable

# 2.27 TableInBatchGetRowRequest

TableInBatchGetRowRequest indicates the request information of the table to be read in the BatchGetRow operation.

### Data structure

```
message TableInBatchGetRowRequest {
    required string table_name = 1;
    repeated bytes primary_key = 2; // Plainbuffer encoding
    repeated bytes token = 3;
    repeated string columns_to_get = 4; // If it is not specified, all columns are read
    optional TimeRange time_range = 5;
    optional int32 max_versions = 6;
    optional bool cache_blocks = 7 [default = true]; // Whether the read data enters the BlockCache
    optional bytes filter = 8;
    optional string start_column = 9;
    optional string end_column = 10;
}
```

`table_name`:

- Type: String
- The name of the table.

`primary_key`:

- Type: Repeated bytes
- Required parameter: Yes
- All primary key columns in the row, including the primary key names and values. The primary key columns are encoded in Plainbuffer format. For more information, see Plainbuffer encoding.

`token`:

- Type: Repeated bytes
- Required parameter: No
- It specifies the starting position of a wide row to be read next time, which is unavailable currently.

`columns_to_get`:

- Type: Repeated string
- The names of all columns to be returned from this table.

`time_range :`

- Type: TimeRange
- Required parameter: Either time_range or max_versions must exist, or both.
- The range of time stamps to read versions of data.
- The time stamp is in the unit of millisecond. The minimum and maximum values of the time stamp is 0 and INT64. MAX, respectively.
- To query data of a time range, specify start_time and end_time.
- To query data of a specific time stamp, specify specific_time.
- Example: If the value of time_range is (100, 200), the time stamp of the returned column data must be within [100, 200).

`max_versio ns :`

- Type: int32
- Required parameter: Either of max_versions and time_range must exist at least.
- The maximum number of versions of data to be returned.
- Example: If the value of max_versions is 2, data of a maximum of two versions is returned for each column.

`cache_bloc ks :`

- Type: Bool
- Required parameter: No
- Whether the read data enters the BlockCache.
- The default value is true.
- This parameter cannot be set to false currently.

`filter :`

- Type: Bytes
- Required parameter: No
- The filtering condition expression.
- It indicates the binary data after the Filter is serialized in Protobuf format.

`start_colu mn :`

- Type: String
- Required parameter: No
- The starting column to be read, which is used for reading a wide row.

- The returned results contain the current starting column.
- The column names are sorted lexicographically.
- Example: If a table contains columns “a”, “b”, and “c” and the value of start_column is “b”, the reading starts from column “b”, and columns “b” and “c” are returned.

`end_column :`

- Type: String
- Required parameter: No
- The ending column to be read, which is used for reading a wide row.
- The returned results do not contain the current ending column.
- The column names are sorted lexicographically.
- Example: If a table contains columns “a”, “b”, and “c” and the value of end_column is “b”, the reading ends at column “b”, and column “a” is returned.

### Related operations

BatchGetRow

## 2.28 TableInBatchGetRowResponse

TableInBatchGetRowResponse indicates data in a table in the messages returned by the BatchGetRow operation.

### Data structure

```
message TableInBatchGetRowResponse {
    required string table_name = 1;
    repeated RowInBatchGetRowResponse rows = 2;
}
```

`table_name :`

- Type: string
- The name of the table.

`rows :`

- Type: repeated RowInBatchGetRowResponse
- All row data read in the table.

### Related operations

BatchGetRow

## 2.29 TableInBatchWriteRowRequest

TableInBatchWriteRowRequest indicates the request information of the table to be written in the BatchWriteRow operation.

### Data structure

```
message TableInBatchWriteRowRequest {
    required string table_name = 1;
    repeated RowInBatchWriteRowRequest rows = 2;
}
```

table_name :

- Type: string
- The name of the table.

rows :

- Type: repeated RowInBatchWriteRowRequest
- The information of the row to be inserted, updated, or deleted in the table.

### Related operations

BatchWriteRow

## 2.30 TableInBatchWriteRowResponse

TableInBatchWriteRowResponse indicates the write results for a table in the BatchWriteRow operation.

### Data structure

```
message TableInBatchWriteRowResponse {
    required string table_name = 1;
    repeated RowInBatchWriteRowResponse put_rows = 2;
    repeated RowInBatchWriteRowResponse update_rows = 3;
    repeated RowInBatchWriteRowResponse delete_rows = 4;
}
```

table_name :

- Type: string
- The name of the table.

put_rows :

- Type: RowInBatchWriteRowResponse
- The results of the PutRow operation for the table.

update_row s :

- Type: RowInBatchWriteRowResponse
- The results of the UpdateRow operation for the table.

delete_row s :

- Type: RowInBatchWriteRowResponse
- The results of the DeleteRow operation for the table.

### Related operations

BatchWriteRow

## 2.31 TableMeta

TableMeta indicates the structure information of a table.

### Data structure

```
message TableMeta {
    required string table_name = 1 ;
    repeated PrimaryKey Schema primary_ke y = 2 ;
}
```

table_name :

- Type: string
- The name of the table.

primary_ke y :

- Type: repeated PrimaryKeySchema
- All primary key columns for the table.

### Related operations

CreateTable

DescribeTable

## 2.32 TableOptions

TableOptions indicates the table parameters, including TimeToLive and MaxVersions.

### Data structure

```
message TableOptio ns {
    optional int32 time_to_li ve = 1 ; // It can be
dynamicall y modified
    optional int32 max_versio ns = 2 ; // It can be
dynamicall y modified
    optional int64 deviation_ cell_versi on_in_sec = 5 ; //
It can be dynamicall y modified
}
```

`time_to_li ve :`

- Type: int32
- The retention time of data stored in this table (unit: second).

`max_versio ns :`

- Type: int32
- The maximum number of versions stored in this table.

`deviation_ cell_versi on_in_sec :`

- Type: int64
- The maximum version deviation. This parameter is used to forbid writing data that deviates from the expected output. For example, if the value of deviation_cell_version_in_sec is 1000 and the current time stamp is 10000, the allowed time stamp range to be written is [10000 – 1000, 10000 + 1000].

## 2.33 TimeRange

TimeRange specifies the time stamp range or time stamp value during data query.

### Data structure

```
message TimeRange {
    optional int64 start_time = 1 ;
    optional int64 end_time = 2 ;
    optional int64 specific_t ime = 3 ;
}
```

`start_time :`

- Type: int64

- The starting time stamp (unit: millisecond). The minimum and maximum values of the time stamp are 0 and INT64. MAX, respectively.

`end_time :`

- Type: int64
- The ending time stamp (unit: millisecond). The minimum and maximum values of the time stamp are 0 and INT64. MAX, respectively.

`specific_t ime :`

- Type: int64
- The specified time stamp. You can set either specific_time, or [start_time, end_time ). The unit for the time stamp must be millisecond. The minimum and maximum values of the time stamp are 0 and INT64. MAX, respectively.