Table Store

API Reference

API Reference

Operations

OperationsSummary

Single row operations

- GetRow
- PutRow
- UpdateRow
- DeleteRow

Multi-row operations

- GetRange
- BatchGetRow
- BatchWriteRow

Table Store stream operations

Table operations

GetRow

Action:

GetRow reads a single data row based on a given primary key.

Request structure:

```
message GetRowRequest {
required string table_name = 1;
required bytes primary_key = 2; // The Plainbuffer encoding is binary
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 bytes filter = 7;
optional string start_column = 8;
optional string end_column = 9;
optional bytes token = 10;
}
```

table_name:

Type: String

Required parameter: Yes

The name of the table containing the data to be read.

primary_key:

Type: 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.

columns_to_get:

Type: Repeated string

Required parameter: No

The names of all columns to be returned. If it is null, all columns of this row are returned.

If the specified column does not exist, data of the column is not 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.

filter:

Type: Bytes

Required parameter: No

The filtering condition expression.

It indicates the binary data after the Filter is serialized in Protobuf format.

start_column:

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 GetRowResponse {
required ConsumedCapacity consumed = 1;
required bytes row = 2; // The Plainbuffer encoding is binary
}
```

consumed:

Type: CapacityUnit

The capacity units consumed by this 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.

Capacity unit consumption:

If the requested row does not exist, one read capacity unit is consumed.

If the requested row exists, the number of read capacity units consumed is the sum of the size of all primary key columns in the row 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.

PutRow

Action:

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_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 PutRowResponse {
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 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.

It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

If return_content is not set or no value is returned, NULL is returned.

UpdateRow

Action:

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 time stamp 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_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 UpdateRowResponse {
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 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.

It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

If return_content is not set or no value is returned, NULL is returned.

DeleteRow

Action:

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.

It is in Plainbuffer format. For more information about encoding, see Plainbuffer encoding.

If return_content is not set or no value is returned, NULL is returned.

GetRange

Action:

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_primary_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_primary_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_primary_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_column:

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.

BatchGetRow

Action:

BatchGetRow reads several data rows from one or more tables. It is essentially a set of multiple GetRow operations. Each operation is independently executed, returns results independently, and independently consumes capacity units.

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

Request Structure:

```
message BatchGetRowRequest {
repeated TableInBatchGetRowRequest tables = 1;
}
```

tables

Type: repeated TableInBatchGetRowRequest

Required Parameter: Yes

Specifies the information of rows to be read.

The entire operation fails and returns an error if tables has any of the following conditions:

- Any table in tables does not exist.

- The name of any table in tables does not comply with the Table naming conventions.

The primary key is not specified for any row in tables, the primary key name does not comply with the conventions, or the primary key type is incorrect.

For any table in tables, a column name in columns_to_get does not comply with the Column naming conventions.

tables contains tables with identical names.

Any table in tables contains rows with identical primary keys.

In tables, the total number of RowInBatchGetRowRequest exceeds 100.

Any table in tables does not contain any RowInBatchGetRowRequest.

Columns_to_get for any table in tables exceeds 128 columns.

Response message structure:

```
message BatchGetRowResponse {
repeated TableInBatchGetRowResponse tables = 1;
}
```

tables

Type: repeated TableInBatchGetRowResponse

Corresponds to the data to be read in each table.

The TableInBatchGetRowResponse object order in the response message is the same as the TableInBatchGetRowRequest object order in BatchGetRowRequest. The order of each RowInBatchGetRowResponse under TableInBatchGetRowResponse is the same as that of RowInBatchGetRowRequest under TableInBatchGetRowRequest.

If a row does not exist or does not have data in the specified columns_to_get, a corresponding RowInBatchGetRowResponse still appears in TableInBatchGetRowResponse, but the row’ s primary_key_columns and attribute_columns are null.

If a row fails to be read, the is_ok value in RowInBatchGetRowResponse for the row is false and the row is null.

Note: At the row level, the BatchGetRow operation may partially fail. In this case, it still returns an HTTP status code of 200, but the application must check errors in RowInBatchGetRowResponse 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 a request time-out occurs and the results are undefined, a capacity unit may or may not be consumed.

In other situations, each RowInBatchGetRowRequest operation is viewed as one GetRow operation when write capacity units are counted.

BatchWriteRow

Action:

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 Table naming conventions.
- 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 Column naming conventions.
- 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 Restricted items.
- 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.

CreateTable

Action:

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

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. To request raising the limit, open a ticket.

Request structure:

```
message CreateTableRequest {
required TableMeta table_meta = 1;
required ReservedThroughput reserved_throughput = 2;
optional TableOptions table_options = 3;
optional StreamSpecification stream_spec = 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 Table naming rules, and the type value can only be STRING, INTEGER, or BINARY.

Once a table is created, its Schema cannot be modified.

reserved_throughput:

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_options:

Type: TableOptions

Required parameter: Yes

Sets TimeToLive and MaxVersions.

stream_spec:

Type: StreamSpecification

Required parameter: No

Specifies whether to enable Stream-related attributes.

Response message structure:

```
message CreateTableResponse {
}
```

ListTable

Action:

ListTable obtains 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.

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.

DeleteTable

Action:

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.

UpdateTable

Action:

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 UpdateTableRequest {
required string table_name = 1;
optional ReservedThroughput reserved_throughput = 2;
optional TableOptions table_options = 3;
optional StreamSpecification stream_spec = 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_throughput:

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_options:

Type: TableOptions

Required parameter: Yes

Sets TimeToLive and MaxVersions.

StreamSpecification:

Type: StreamSpecification

Required parameter: No

Specifies whether to enable Stream-related attributes.

Response message structure:

```
message UpdateTableResponse {
required ReservedThroughputDetails reserved_throughput_details = 1;
required TableOptions table_options = 2;
}
```

capacity_unit_details:

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_options:

Type: TableOptions

The latest value of the table_options parameter after the modification.

DescribeTable

Action:

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

Request structure:

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

table_name:

Type: String

Required parameter: Yes

The name of the table to be queried.

Response message structure:

```
message DescribeTableResponse {
required TableMeta table_meta = 1;
required ReservedThroughputDetails reserved_throughput_details = 2;
required TableOptions table_options = 3;
optional StreamDetails stream_details = 5;
repeated bytes shard_splits = 6;
}
```

table_meta:

Type: TableMeta

The table’ s schema, which is the same as the Schema given at the time of table creation.

reserved_throughput_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_options:

Type: TableOptions

The latest value of the table_options parameter.

StreamSpecification:

Type: StreamSpecification

Required parameter: No

Specifies whether to enable Stream-related attributes.

shard_splits:

Type: bytes

The split points of all shards in the current table.

ComputeSplitPointsBySize

Action

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 100 MB
}
```

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.
* This means the actual size is difficult to determine.
*
* 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 is omitted to reduce transmission payloads.
*/
repeated bytes split_points = 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 is placed.
*/
message SplitLocation {
required string location = 1;
required sint64 repeat = 2;
}
repeated SplitLocation 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_points:

- 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)

    ("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.

ListStream

Action:

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

Request structure:

```
message ListStreamRequest {
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 ListStreamResponse {
repeated Stream streams = 1;
}

message Stream {
required string stream_id = 1;
required string table_name = 2;
required int64 creation_time = 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_time:

Type: required int64

The time when the current stream is enabled.

DescribeStream

Action:

DescribeStream obtains information about the shards of the current stream.

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

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.

GetShardIterator

Action:

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 GetShardIteratorResponse {
required string shard_iterator = 1;
}
```

shard_iterator:

Type: required string

The start iterator for reading the record of the current shard.

GetStreamRecord

Action:

GetStreamRecord obtains the incremental content of the current shard.

Request structure:

```
message GetStreamRecordRequest {
required string shard_iterator = 1;
optional int32 limit = 2;
}
```

shard_iterator:

Type: required string

The iterator for reading the current shard.

Response message structure:

```
message GetStreamRecordResponse {
message StreamRecord {
required ActionType action_type = 1;
required bytes record = 2;
}
repeated StreamRecord stream_records = 1;
optional raw_string next_shard_iterator = 2;
optional ConsumedCapacity consumed = 3;
}
```

StreamRecord:

Type: repeated StreamRecord

The record entry for reading the current shard.

shard_iterator:

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.

TableStore ProtocolBuffer Message Definitions

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

table_store.proto

```
package com.alicloud.openservices.tablestore.core.protocol;

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

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

enum PrimaryKeyOption {
AUTO_INCREMENT = 1;
}

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

message TableOptions {
optional int32 time_to_live = 1; // It can be dynamically modified
optional int32 max_versions = 2; // It can be dynamically modified
optional int64 deviation_cell_version_in_sec = 5; // It can be dynamically modified
}

message TableMeta {
required string table_name = 1;
repeated PrimaryKeySchema primary_key = 2;
}

enum RowExistenceExpectation {
IGNORE = 0;
EXPECT_EXIST = 1;
EXPECT_NOT_EXIST = 2;
}

message Condition {
required RowExistenceExpectation row_existence = 1;
optional bytes column_condition = 2;
}

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

message ReservedThroughputDetails {
required CapacityUnit capacity_unit = 1; // It indicates the value of the current reserved throughput of the table
required int64 last_increase_time = 2; // It indicates the last time that the reserved throughput was raised
optional int64 last_decrease_time = 3; // It indicates the last time that the reserved throughput was lowered
}
```

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

message ConsumedCapacity {
required CapacityUnit capacity_unit = 1;
}

/* ############################################### CreateTable
############################################### */
/**
* table_meta is used to store unmodifiable schema attributes in a table. The ReservedThroughput and TableOptions
attributes that can be modified are independently stored as the parameters for UpdateTable.
* message CreateTableRequest {
* required TableMeta table_meta = 1;
* required ReservedThroughput reserved_throughput = 2;
* required TableOptions table_options = 3;
* }
*/
message CreateTableRequest {
required TableMeta table_meta = 1;
required ReservedThroughput reserved_throughput = 2;
optional TableOptions table_options = 3;
}

message CreateTableResponse {
}

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


/* ############################################### UpdateTable
############################################### */
message UpdateTableRequest {
required string table_name = 1;
optional ReservedThroughput reserved_throughput = 2;
optional TableOptions table_options = 3;
}

message UpdateTableResponse {
required ReservedThroughputDetails reserved_throughput_details = 1;
required TableOptions table_options = 2;
}
/*
##########################################################################################
#################### */

/* ############################################### DescribeTable
############################################### */
message DescribeTableRequest {
required string table_name = 1;
}
```

```
message DescribeTableResponse {
required TableMeta table_meta = 1;
required ReservedThroughputDetails reserved_throughput_details = 2;
required TableOptions table_options = 3;
repeated bytes shard_splits = 6;
}
/*
######################################################################################
##################### */

/* ############################################# ListTable
############################################# */
message ListTableRequest {
}

/**
* Only a simple name list is returned currently.
*/
message ListTableResponse {
repeated string table_names = 1;
}
/*
######################################################################################
################# */

/* ############################################# DeleteTable
############################################# */
message DeleteTableRequest {
required string table_name = 1;
}

message DeleteTableResponse {
}

/* ############################################# UnloadTable
############################################# */
message UnloadTableRequest {
required string table_name = 1;
}

message UnloadTableResponse {

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

/**
* The minimum and maximum values of the time stamp are 0 and INT64.MAX, respectively.
* 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_time.
*/
message TimeRange {
optional int64 start_time = 1;
optional int64 end_time = 2;
optional int64 specific_time = 3;
}

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

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

message ReturnContent {
optional ReturnType return_type = 1;
}

/**
* 1. You can specify the time stamp range or time to read the columns of the specified version.
* 2. Intra-row breakpoints are not supported currently.
*/
message GetRowRequest {
required string table_name = 1;
required bytes primary_key = 2; // encoded as InplaceRowChangeSet, 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_versions = 5;
optional bytes filter = 7;
optional string start_column = 8;
optional string end_column = 9;
optional bytes token = 10;
}

message GetRowResponse {
required ConsumedCapacity consumed = 1;
required bytes row = 2; // encoded as InplaceRowChangeSet
optional bytes next_token = 3;
}
/*
#######################################################################################
############## */

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

message UpdateRowResponse {
required ConsumedCapacity 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 PutRowRequest {
required string table_name = 1;
required bytes row = 2; // encoded as InplaceRowChangeSet
required Condition condition = 3;
optional ReturnContent return_content = 4;
}

message PutRowResponse {
required ConsumedCapacity consumed = 1;
optional bytes row = 2;
}
/*
#########################################################################################
############## */

/* ############################################### DeleteRow
############################################### */
/**
* Table Store only supports deleting all versions of all columns in a specified row.
*/
message DeleteRowRequest {
required string table_name = 1;
required bytes primary_key = 2; // encoded as InplaceRowChangeSet, but only has primary key
required Condition condition = 3;
optional ReturnContent return_content = 4;
}

message DeleteRowResponse {
required ConsumedCapacity consumed = 1;
optional bytes row = 2;
}
/*
#########################################################################################
################# */

/* ############################################### BatchGetRow
############################################### */
message TableInBatchGetRowRequest {
required string table_name = 1;
repeated bytes primary_key = 2; // encoded as InplaceRowChangeSet, 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_versions = 6;
optional bytes filter = 8;
optional string start_column = 9;
optional string end_column = 10;
}

message BatchGetRowRequest {
repeated TableInBatchGetRowRequest tables = 1;
```

```
}

message RowInBatchGetRowResponse {
required bool is_ok = 1;
optional Error error = 2;
optional ConsumedCapacity consumed = 3;
optional bytes row = 4; // encoded as InplaceRowChangeSet
optional bytes next_token = 5;
}

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

message BatchGetRowResponse {
repeated TableInBatchGetRowResponse tables = 1;
}
/*
#######################################################################################
################### */

/* ############################################# BatchWriteRow
############################################# */

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

message RowInBatchWriteRowRequest {
required OperationType type = 1;
required bytes row_change = 2; // encoded as InplaceRowChangeSet
required Condition condition = 3;
optional ReturnContent return_content = 4;
}

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

message BatchWriteRowRequest {
repeated TableInBatchWriteRowRequest tables = 1;
}

message RowInBatchWriteRowResponse {
required bool is_ok = 1;
optional Error error = 2;
optional ConsumedCapacity consumed = 3;
optional bytes row = 4;
}

message TableInBatchWriteRowResponse {
required string table_name = 1;
```

```
repeated RowInBatchWriteRowResponse rows = 2;
}

message BatchWriteRowResponse {
repeated TableInBatchWriteRowResponse tables = 1;
}
/*
##############################################################################################
####################### */

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

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; // encoded as InplaceRowChangeSet, but only has primary key
required bytes exclusive_end_primary_key = 8; // encoded as InplaceRowChangeSet, but only has primary key
optional bytes filter = 10;
optional string start_column = 11;
optional string end_column = 12;
optional bytes token = 13;
}

message GetRangeResponse {
required ConsumedCapacity consumed = 1;
required bytes rows = 2; // encoded as InplaceRowChangeSet
optional bytes next_start_primary_key = 3; // If it is null, all data has been read. It is encoded as
InplaceRowChangeSet, but only has primary key
optional bytes next_token = 4;
}

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

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 typically split_size specified in the request.
* This means the actual size is difficult to determine.
```

```
*
* 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 is omitted to reduce transmission payloads.
*/
repeated bytes split_points = 3;

/**
* Locations where splits lies in.
*
* Due to the inherent design of TableStore, locations are not an absolute concept.
* If a location is not suitable to be seen, an empty string is placed.
*/
message SplitLocation {
required string location = 1;
required sint64 repeat = 2;
}
repeated SplitLocation locations = 4;
}
```

table_store_filter.proto

```


package com.alicloud.openservices.tablestore.core.protocol;

enum FilterType {
FT_SINGLE_COLUMN_VALUE = 1;
FT_COMPOSITE_COLUMN_VALUE = 2;
FT_COLUMN_PAGINATION = 3;
}

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;
}

message SingleColumnValueFilter {
required ComparatorType comparator = 1;
required string column_name = 2;
required bytes column_value = 3;
required bool filter_if_missing = 4;
required bool latest_version_only = 5;
}

enum LogicalOperator {
LO_NOT = 1;
LO_AND = 2;
LO_OR = 3;
}

message CompositeColumnValueFilter {
```

```
required LogicalOperator combinator = 1;
repeated Filter sub_filters = 2;
}

message ColumnPaginationFilter {
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
}
```

Data Types

DataType Summary

- CapacityUnit
- ColumnPaginationFilter
- ComparatorType
- CompositeColumnValueFilter
- Condition
- ConsumedCapacity
- Direction
- Error
- Filter

FilterType

LogicalOperator

OperationType

PrimaryKeyOption

PrimaryKeySchema

PrimaryKeyType

ReservedThroughput

ReservedThroughputDetails

ReturnContent

ReturnType

RowExistenceExpectation

RowInBatchGetRowResponse

RowInBatchWriteRowRequest

RowInBatchWriteRowResponse

SingleColumnValueFilter

StreamDetails

StreamRecord

StreamSpecification

TableInBatchGetRowRequest

TableInBatchGetRowResponse

TableInBatchWriteRowRequest

TableInBatchWriteRowResponse

TableMeta

TableOptions

TimeRange

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

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

ColumnPaginationFilter

ColumnPaginationFilter specifies the filtering conditions for reading a wide row. It is applicable to the

Filter function.

Data structure

```
message ColumnPaginationFilter {
required int32 offset = 1;
required int32 limit = 2;
}
```

offset:

Type: int32

The position of the starting column, that is, the first column to be read

limit:

Type: int32

The number of columns to be read

Related operations

Filter

GetRow

GetRange

BatchGetRow

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;
}
```

CompositeColumnValueFilter

CompositeColumnValueFilter specifies a group of conditions, for example, column_a > 5 AND column_b = 10. It is applicable to the ConditionUpdate and Filter functions.

Data structure

```
message CompositeColumnValueFilter {
required LogicalOperator combinator = 1;
repeated Filter sub_filters = 2;
}
```

combinator:

Type: LogicalOperator

The logical operator.

sub_filter:

Type: Filter

The sub-condition expression.

Related operations

ConditionUpdate

PutRow

UpdateRow

DeleteRow

BatchWriteRow

Filter

GetRow

GetRange

BatchGetRow

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 RowExistenceExpectation row_existence = 1;
optional bytes column_condition = 2;
}
```

row_existence:

Type: RowExistenceExpectation

The row existence check settings for this row.

column_condition:

Type: Bytes

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

Related operations

PutRow

UpdateRow

DeleteRow

BatchWriteRow

ConsumedCapacity

ConsumedCapacity indicates capacity units consumed by an operation.

Data structure

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

capacity_unit:

Type: CapacityUnit

The value of capacity units consumed by this operation.

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

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. For more information, see Error messages.

message:

Type: string

The error message for the current individual row operation. For more information, see Error messages.

Related operations

BatchGetRow

BatchWriteRow

Filter

Filter specifies the column-based conditions. It is applicable to the Conditional Update and Filter functions.

Data structure

```
message Filter {
required FilterType type = 1;
required bytes filter = 2;
}
```

type:

Type: FilterType

Column condition type.

filter:

Type: bytes

The binary data of the condition statement in CompositeColumnValueFilter, ColumnPaginationFilter, or SingleColumnValueFilter type after being serialized in Protobuf format.

Related operations

ConditionUpdate

PutRow

UpdateRow

DeleteRow

BatchWriteRow

Filter

GetRow

GetRange

BatchGetRow

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_VALUE = 1;
FT_COMPOSITE_COLUMN_VALUE = 2;
FT_COLUMN_PAGINATION = 3;
}
```

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 LogicalOperator {
LO_NOT = 1;
LO_AND = 2;
LO_OR = 3;
}
```

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;
}
```

PartitionRange

PartitionRange specifies the shard range.

Data structure

```
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.

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_name formated_value
cell_value = tag_cell_value formated_value
cell_op = tag_cell_op cell_op_value
cell_ts = tag_cell_ts cell_ts_value
row_checksum = tag_row_checksum row_crc8
cell_checksum = tag_cell_checksum row_crc8

formated_value = value_type value_len value_data
value_type = int8
value_len = int32

cell_op_value = delete_all_version | delete_one_version
cell_ts_value = 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_name = 0x04 (1byte)
tag_cell_value = 0x05 (1byte)
tag_cell_op = 0x06 (1byte)
tag_cell_ts = 0x07 (1byte)
tag_delete_marker = 0x08 (1byte)
tag_row_checksum = 0x09 (1byte)
tag_cell_checksum = 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_INCREMENT = 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.

C++ implementation:

```
void GetChecksum(uint8_t* crc, const InplaceCell& cell)
{
Crc8(crc, cell.GetName());
Crc8(crc, cell.GetValue().GetInternalSlice());
Crc8(crc, cell.GetTimestamp());
Crc8(crc, cell.GetOpType());
}

void GetChecksum(uint8_t* crc, const InplaceRow& row)
{
const std::deque<InplaceCell>& pk = row.GetPrimaryKey();
for (size_t i = 0; i < pk.size(); i++) {
uint8_t* cellcrc;
*cellcrc = 0;
GetChecksum(cellcrc, pk[i]);
Crc8(crc, *cellcrc);
}
for (int i = 0; i < row.GetCellCount(); i++) {
uint8_t* cellcrc;
*cellcrc = 0;
GetChecksum(cellcrc, row.GetCell(i));
Crc8(crc, *cellcrc);
}

uint8_t del = 0;
if (row.HasDeleteMarker()) {
del = 1;
}
Crc8(crc, del);
}
```

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:

```
    <Header starting>[0x75]
<Primary key column starting>[0x1]
<Cell1>[0x3][0x4][3][pk1][0x5][3][5][iampk]
<Cell2>[0x3][0x4][3][pk2][0x5][0][100]
<Attribute column starting>[0x2]
<Cell1>[0x3][0x4][7][column1][0x5][0x3][3][bad][0x7][1001]
<Cell2>[0x3][0x4][7][column2][0x5][0x0][128][0x7][1002]
<Cell3>[0x3][0x4][7][column3][0x5][0x1][34.2][0x7][1003]
<Cell4>[0x3][0x4][7][column4][0x5][0x6][1]
```

PrimaryKeyOption

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

Enumeration value list

```
enum PrimaryKeyOption {
AUTO_INCREMENT = 1;
}
```

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.

PrimaryKeyType

PrimaryKeyType specifies the type of the primary key.

Enumeration data type

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

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

ReservedThroughputDetails

Data structure

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

```
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

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.

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.

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_EXIST = 1;
EXPECT_NOT_EXIST = 2;
}
```

Related operations

PutRow

UpdateRow

DeleteRow

BatchWriteRow

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

RowInBatchWriteRowRequest

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

Data structure

```
message RowInBatchWriteRowRequest {
required OperationType type = 1;
required bytes row_change = 2; // encoded as InplaceRowChangeSet
required Condition condition = 3;
optional ReturnContent return_content = 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_content:

Type: ReturnContent

The type of the returned data.

Related operations

BatchWriteRow

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

SingleColumnValueFilter

SingleColumnValueFilter specifies a single condition, for example, column_a > 5. It is applicable to the ConditionUpdate and Filter functions.

Data structure

```
message SingleColumnValueFilter {
required ComparatorType comparator = 1;
required string column_name = 2;
required bytes column_value = 3;
required bool filter_if_missing = 4;
required bool latest_version_only = 5;
}
```

comparator:

Type: ComparatorType

The comparison type.

column_name:

Type: String

The column name.

column_value:

Type: Bytes

The value of the column after being encoded in Plainbuffer format.

filter_if_missing:

Type: Bool

Determines whether the condition is filtered if the column of a row does not exist. For example, the condition is column_a > 0, and the value of filter_if_missing is true. If column_a does not exist for a row, the condition column_a > 0 is true.

latest_version_only:

Type: Bool

Determines whether the condition is valid only for the latest version. If the value is true, the value of the latest version is checked if it meets the condition. If the value is false, the values of all versions are checked if they meet the conditions.

Related operations

ConditionUpdate

PutRow

UpdateRow

DeleteRow

BatchWriteRow

Filter

GetRow

GetRange

BatchGetRow

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

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_type:

Type: Required ActionType

The operation type of the row.

record:

Type: Required bytes

The data content of the row.

Related operations

GetStreamRecord

StreamSpecification

StreamSpecification indicates the stream of a table.

Data structure

```
message StreamSpecification {
required bool enable_stream = 1;
optional int32 expiration_time = 2;
}
```

enable_stream:

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

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_versions:

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_blocks:

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_column:

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

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

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

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_rows:

Type: RowInBatchWriteRowResponse

The results of the UpdateRow operation for the table.

delete_rows:

Type: RowInBatchWriteRowResponse

The results of the DeleteRow operation for the table.

Related operations

BatchWriteRow

TableMeta

TableMeta indicates the structure information of a table.

Data structure

```
message TableMeta {
required string table_name = 1;
repeated PrimaryKeySchema primary_key = 2;
}
```

table_name:

Type: string

The name of the table.

primary_key:

Type: repeated PrimaryKeySchema

All primary key columns for the table.

Related operations

CreateTable

DescribeTable

TableOptions

TableOptions indicates the table parameters, including TimeToLive and MaxVersions.

Data structure

```
message TableOptions {
optional int32 time_to_live = 1; // It can be dynamically modified
optional int32 max_versions = 2; // It can be dynamically modified
optional int64 deviation_cell_version_in_sec = 5; // It can be dynamically modified
}
```

time_to_live:

Type: int32

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

max_versions:

Type: int32

The maximum number of versions stored in this table.

deviation_cell_version_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].

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_time = 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_time:

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.