Table Store

API Reference

API Reference

Operations

Operations

Single row operations:

- GetRow
- PutRow
- UpdateRow
- DeleteRow

Multi-row operations:

- GetRange
- BatchGetRow
- BatchWriteRow

Table operations:

- CreateTable
- ListTable

DeleteTable

UpdateTable

DescribeTable

GetRow

Action:

Reads a single data row based on a given primary key.

Request Structure:

```
message GetRowRequest {
required string table_name = 1;
repeated Column primary_key = 2;
repeated string columns_to_get = 3;
}
```

table_name:

- Type: string

- Required Parameter: Yes

- The name of the table holding the data to be read.

primary_key:

- Type: repeated Column

- Required Parameter: Yes

- All primary key columns of this row.

columns_to_get:

- Type: repeated string

- Required Parameter: No

- The names of all columns to be returned. If null, all columns of this row will be returned.

- If the specified column does not exist, data will not be returned for this column.

- If a duplicate column name is given, the return results will only include this column once.

- In columns_to_get, the number of strings should not exceed 128.

Response Message Structure:

```
message GetRowResponse {
required ConsumedCapacity consumed = 1;
required Row row = 2;
}
```

consumed:

- Type: CapacityUnit

- The capacity units consumed by this operation.

row:

- Type: Row

- The collection of column data to be returned for this row. Here, primary_key_columns and attribute_columns respectively store primary key columns and attribute columns that are read. Their order may not be consistent with that of columns_to_get in GetRowRequest.

- If this row does not exist, primary_key_columns and attribute_columns will be null.

Capacity Unit Consumption:

- If the requested row does not exist, 1 read capacity unit is consumed.

- If the requested row exists, the number of consumed read capacity units is the size of all data from this row divided by 1KB and rounded up. For information on data size calculation, please refer to Billing Explanation.

- If request timeout occurs 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), this operation will not consume capacity units. If other errors are returned, it will consume 1 read capacity unit.

Request Example:

```
GetRowRequest {
table_name: "consume_history"
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
```

```
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
columns_to_get: "CardID"
columns_to_get: "SellerID"
columns_to_get: "DeviceID"
columns_to_get: "OrderNumber"
columns_to_get: "Amount"
columns_to_get: "Remarks"
}
```

Response Example:

```
GetRowResponse {
consumed {
capacity_unit {
read: 1
}
}
row {
primary_key_columns {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
```

```
}
}
primary_key_columns {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key_columns {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key_columns {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
attribute_columns {
name: "Amount"
value {
type: DOUBLE
v_double: 2.5
```

```
}
}
attribute_columns {
name: "Remarks"
value {
type: STRING
v_string: "ice cream"
}
}
}
}
```

PutRow

Action:

Inserts data in the specified row. If this 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 Condition condition = 2;
repeated Column primary_key = 3;
repeated Column attribute_columns = 4;
}
```

table_name:

Type: string

Required Parameter: Yes

The name of the table to write data to.

condition:

Type: Condition

Required Parameter: Yes

Determines whether or not to perform row existence check before writing data. There are 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, insertion will fail and an error will be returned. Vice versa.

primary_key:

Type: repeated Column

Required Parameter: Yes

All primary key columns for the row to be written.

attribute_columns:

Type: repeated Column

Required Parameter: No

The attributes of the row to be written. If attribute_columns is null, this indicates the row to be written does not have any attribute columns.

In attribute_columns, the number of columns cannot exceed 128.

Response Message Structure:

```
message PutRowResponse {
required ConsumedCapacity consumed = 1;
}
```

consumed:

Type ConsumedCapacity

The capacity units consumed by this operation.

Capacity Unit Consumption:

If this row does not exist, the number of consumed write capacity units is the size of all data to be inserted divided by 1KB and rounded up. For information on data size calculation, please refer to Billing Explanation.

If this row exists, the number of consumed write capacity units is the sum of the size of all original row data and the size of modified row data respectively divided by 1KB and rounded up.

If an internal error code is returned (HTTP status code: 5XX), this operation will not consume capacity units. If other errors are returned, it will consume 1 write capacity unit.

If request timeout occurs and the results are undefined, a capacity unit may or may not be consumed.

Request Example:

```
PutRowRequest {
table_name: "consume_history"
condition {
row_existence: EXPECT_NOT_EXIST
}
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
```

```
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
attribute_columns {
name: "Amount"
value {
type: DOUBLE
v_double: 2.5
}
}
attribute_columns {
name: "Remarks"
value {
type: STRING
v_string: "ice cream"
}
}
}
```

Response Example:

```
PutRowResponse {
consumed {
capacity_unit {
write: 1
}
}
}
~~~~~~~~~~~~~~~~~
```

UpdateRow

Action:

Updates the data of the specified row. If this 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 Condition condition = 2;
repeated Column primary_key = 3;
repeated ColumnUpdate attribute_columns = 4;
}
```

table_name:

Type: string

Required Parameter: Yes

The name of the table to update.

condition:

Type: Condition

Required Parameter: Yes

Determines whether or not to perform existence check before updating data. There are 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 will fail and return an error. If the existence of the row is ignored, whether or not the row exists will not cause this operation to fail.

primary_key:

Type: repeated Column

Required Parameter: Yes

All primary key columns for the row to be updated.

attribute_columns:

Type: repeated ColumnUpdate

Required Parameter: Yes

All the attribute columns to be updated for this row. OTS will add, modify, or delete the values for the specified columns based on the content of each ColumnUpdate in attribute_columns.

Columns already existing in the row, but not in the attribute_columns list, will not be affected.

attribute_columns should contain at least one ColumnUpdate object. Otherwise, the request will fail and return an error.

If attribute_columns contains columns with the same name, the request will fail and return an error.

In attribute_columns, the number of ColumnUpdate objects cannot exceed 128. After the update is complete, the number of attribute columns for each row cannot exceed 128.

The ColumnUpdate type can have the following two values:

PUT: at this time, this ColumnUpdate 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: at this time, this ColumnUpdate value must be null. This indicates that this column is deleted.

If this row does not exist and attribute_columns contains a DELETE-type ColumnUpdate object, the row will still be non-existent after the UpdateRow operation is completed.

Note: Deleting all attribute columns of a row is not the same as deleting the row. If you wish to delete the row, please use the DeleteRow operation.

Response Message Structure:

```
message UpdateRowResponse {
required ConsumedCapacity consumed = 1;
}
```

consumed:

Type ConsumedCapacity

The capacity units consumed by this operation.

Capacity Unit Consumption:

If this row does not exist, the number of consumed write capacity units is the size of all data to be updated divided by 1KB and rounded up. For information on data size calculation, please refer to Billing Explanation.

If this row exists, the number of consumed write capacity units is the size of all original row data (0 if nonexistent) or the size of row data to be updated (whichever is bigger) divided by 1KB and rounded up.

If request timeout occurs 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), this operation will not consume capacity units. If other errors are returned, it will consume 1 write capacity unit.

Request Example:

```
UpdateRowRequest {
table_name: "consume_history"
condition {
row_existence: EXPECT_EXIST
}
primary_key {
name: "CardID"
```

```
value {
type: STRING
v_string: "2007035023"
}
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
attribute_columns {
type: PUT
name: "Amount"
value {
type: DOUBLE
v_double: 3.5
}
}
attribute_columns {
type: DELETE
name: "Remarks"
}
}
```

Response Example:

```
UpdateRowResponse {
consumed {
capacity_unit {
write: 1
}
}
}
```

DeleteRow

Action:

Deletes one row of data.

Request Message Structure:

```
message DeleteRowRequest {
required string table_name = 1;
required Condition condition = 2;
repeated Column primary_key = 3;
}
```

table_name:

Type: string

Required Parameter: Yes

The name of the table to update.

condition:

Type: Condition

Required Parameter: Yes

Determines whether or not to perform existence check before updating data. There are 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 will fail and return an error. If the existence of the row is ignored, whether or not the row exists will not cause this operation to fail.

primary_key:

Type: repeated Column

Required Parameter: Yes

All primary key columns for the row to be updated.

Response Message Structure:

```
message DeleteRowResponse {
required ConsumedCapacity consumed = 1;
}
```

consumed:

Type ConsumedCapacity

The capacity units consumed by this operation.

Capacity Unit Consumption:

If the row does not exist, 1 write capacity unit is consumed.

If this row does exist, the number of consumed write capacity units is the size of all original data divided by 1KB and rounded up. For information on data size calculation, please refer to Billing Explanation.

If an internal error code is returned (HTTP status code: 5XX), this operation will not consume capacity units. If other errors are returned, it will consume 1 write capacity unit.

If request timeout occurs and the results are undefined, a capacity unit may or may not be consumed.

Request Example:

```
DeleteRowRequest {
table_name: "consume_history"
```

```
condition {
row_existence: IGNORE
}
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
}
```

Response Example:

```
DeleteRowResponse {
consumed {
capacity_unit {
write: 1
}
}
}
```

GetRange

Action:

Reads the data in the specified primary key range.

Request Structure:

```
message GetRangeRequest {
required string table_name = 1;
required Direction direction = 2;
repeated string columns_to_get = 3;
optional int32 limit = 4;
repeated Column inclusive_start_primary_key = 5;
repeated Column exclusive_end_primary_key = 6;
}
```

table_name:

Type: string

Required Parameter: Yes

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

direction:

Type: Direction

Required Parameter: Yes

If the query direction is forward, inclusive_start_primary should be smaller than exclusive_end_primary and, in the response, the rows are sorted from smallest to largest primary key. If the query direction is backward, inclusive_start_primary should be larger than exclusive_end_primary and, in the response, the rows are sorted from largest to smallest primary key.

columns_to_get:

Type: repeated string

Required Parameter: No

The names of all columns to be returned. If null, all columns of each row in the returned results will be read.

If a duplicate column name is given, the return results will only include this column once.

In columns_to_get, the number of strings should not exceed 128.

limit:

Type: int32

Required Parameter: No

This value gives the maximum number of returned data rows to be read in a single operation. If the number of rows queried exceeds this value, the response will include a breakpoint to record the position where reading ended in this operation, making the next read easier. This value must be greater than 0.

Whether or not this value is set, OTS will return a maximum of 5000 data rows and the total data size cannot exceed 1M.

inclusive_start_primary_key:

Type: repeated Column

Required Parameter: Yes

The starting primary key for the range to be read. If this row exists, the response will certainly contain it.

exclusive_end_primary_key:

Type: repeated Column

Required Parameter: Yes

The ending primary key for the range to be read. Whether or not this line exists, the response will not contain it.

For GetRange, the column type in inclusive_start_primary_key and exclusive_end_primary_key can use specialized types for this operation: INF_MIN and INF_MAX. INF_MIN-type columns

are always smaller than other columns, while INF_MAX-type columns are always larger than other columns.

Response Message Structure:

```
message GetRangeResponse {
required ConsumedCapacity consumed = 1;
repeated Column next_start_primary_key = 2;
repeated Row rows = 3;
}
```

consumed:

Type: ConsumedCapacity

The capacity units consumed by this operation.

next_start_primary_key:

Type: repeated Column

The breakpoint information for this GetRange operation.

If null, this indicates that the response message for this GetRange operation contains all data in the request range.

If not null, this indicates that the response message for this GetRange operation includes only the data in the interval [inclusive_start_primary_key, next_start_primary_key). If you want the remaining data, you must use the next_start_primary_key as the inclusive_start_primary_key and retain exclusive_end_primary_key of the original request in a subsequent GetRange operation.

Note: In the OTS system, the number of data rows in the response message for a GetRange operation cannot exceed 5,000 and the size cannot exceed 1M. 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 or not the response has a next_start_primary_key and proceed accordingly.

rows:

Type: repeated Row

All data read. If the direction in the request is FORWARD, all rows will be sorted from smallest to largest primary key. If the direction in the request is BACKWARD, all rows will be sorted from largest to smallest primary key.

Here, the primary_key_columns and attribute_columns for each row only contain the columns specified in columns_to_get. 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 columns_to_get specified in the request does not contain any primary key columns, its primary key is in the query range. However, rows that do not contain any attribute column in columns_to_get will not appear in the response message.

Capacity Unit Consumption:

The number of read capacity units consumed by the GetRange operation is the size of all data rows in the query range divided by 1KB and rounded up. For information on data size calculation, please refer to Billing Explanation.

If request timeout occurs 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), this operation will not consume capacity units. If other errors are returned, it will consume 1 write capacity unit.

Request Example:

```
GetRangeRequest {
table_name: "consume_history"
direction: FORWARD
columns_to_get: "CardID"
columns_to_get: "SellerID"
columns_to_get: "DeviceID"
columns_to_get: "OrderNumber"
columns_to_get: "Amount"
columns_to_get: "Remarks"
limit: 2
inclusive_start_primary_key {
name: "CardID"
value {
type: STRING
```

```
v_string: "2007035023"
}
}
inclusive_start_primary_key {
name: "SellerID"
value {
type: INF_MIN
}
}
inclusive_start_primary_key {
name: "DeviceID"
value {
type: INF_MIN
}
}
inclusive_start_primary_key {
name: "OrderNumber"
value {
type: INF_MIN
}
}
exclusive_end_primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
exclusive_end_primary_key {
name: "SellerID"
value {
type: INF_MIN
}
}
exclusive_end_primary_key {
name: "DeviceID"
value {
type: INF_MIN
}
}
exclusive_end_primary_key {
name: "OrderNumber"
value {
type: INF_MIN
}
}
}
```

Response Example:

```
consumed {
capacity_unit {
read: 1
}
```

```
}
next_start_primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
next_start_primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00026"
}
}
next_start_primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "065499"
}
}
next_start_primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 166666
}
}
rows {
primary_key_columns {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key_columns {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key_columns {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key_columns {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
```

```
}
}
attribute_columns {
name: "Amount"
value {
type: DOUBLE
v_double: 2.5
}
}
attribute_columns {
name: "Remarks"
value {
type: STRING
v_string: "ice cream"
}
}
}
rows {
primary_key_columns {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key_columns {
name: "SellerID"
value {
type: STRING
v_string: "00026"
}
}
primary_key_columns {
name: "DeviceID"
value {
type: STRING
v_string: "065499"
}
}
primary_key_columns {
name: "OrderNumber"
value {
type: INTEGER
v_int: 153846
}
}
attribute_columns {
name: "Amount"
value {
type: DOUBLE
v_double: 0.5
}
}
}
```

BatchGetRow

Action:

Batch reads several data rows from one or more tables.

The BatchGetRow operation can be viewed as 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 effectively 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.

If “tables” has the conditions described below, the entire operation will fail and return an error.

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 the same name.

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 will still appear in TableInBatchGetRowResponse, but the row's primary_key_columns and attribute_columns will be null.

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

Note: The BatchGetRow operation may fail partly at the row level. In this case, it will still return an HTTP status code of 200. 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, it will not consume any capacity units.

If request timeout 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.

Request Example:

```
BatchGetRowRequest {
tables {
table_name: "consume_history"
rows {
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
}
rows {
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
```

```
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00026"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "065499"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 153846
}
}
}
columns_to_get: "CardID"
columns_to_get: "SellerID"
columns_to_get: "DeviceID"
columns_to_get: "OrderNumber"
columns_to_get: "Amount"
columns_to_get: "Remarks"
}
}
```

Response Example:

```
BatchGetRowResponse {
tables {
table_name: "consume_history"
rows {
is_ok: true
consumed {
capacity_unit {
read: 1
}
}
row {
primary_key_columns {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key_columns {
name: "SellerID"
value {
```

```
type: STRING
v_string: "00022"
}
}
primary_key_columns {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key_columns {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
attribute_columns {
name: "Amount"
value {
type: DOUBLE
v_double: 2.5
}
}
attribute_columns {
name: "Remarks"
value {
type: STRING
v_string: "ice cream"
}
}
}
}
rows {
is_ok: true
consumed {
capacity_unit {
read: 1
}
}
row {
primary_key_columns {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key_columns {
name: "SellerID"
value {
type: STRING
v_string: "00026"
}
}
```

```
primary_key_columns {
name: "DeviceID"
value {
type: STRING
v_string: "065499"
}
}
primary_key_columns {
name: "OrderNumber"
value {
type: INTEGER
v_int: 153846
}
}
attribute_columns {
name: "Amount"
value {
type: DOUBLE
v_double: 0.5
}
}
}
}
}
}
```

BatchWriteRow

Action:

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

The BatchWriteRow operation can be viewed as a set of multiple PutRow, UpdateRow, and DeleteRow operations. Each operation is executed and 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 effectively 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.

In the following cases, a general error will be returned:

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 Upper Limit.

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 will be false.

Note: The BatchWriteRow operation may fail partly at the row level. In this case, it will still return an HTTP status code of 200. 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, it will not consume any capacity units.

If request timeout occurs and the results are undefined, a capacity unit may or may not be consumed.

In other situations, each PutRowInBatchWriteRowRequest, UpdateRowInBatchWriteRowRequestDelete, and RowInBatchWriteRowRequest operation is

viewed as a corresponding write operation when the number of write capacity units are counted.

Request Example:

```
BatchWriteRowRequest {
tables {
table_name: "consume_history"
put_rows {
condition {
row_existence: IGNORE
}
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00022"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "061104"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 142857
}
}
attribute_columns {
name: "Amount"
value {
type: DOUBLE
v_double: 2.5
}
}
attribute_columns {
name: "Remarks"
value {
type: STRING
v_string: "ice cream"
}
```

```
}
}
update_rows {
condition {
row_existence: EXPECT_EXIST
}
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
}
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00026"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "065499"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 153846
}
}
attribute_columns {
type: PUT
name: "Amount"
value {
type: DOUBLE
v_double: 1
}
}
attribute_columns {
type: DELETE
name: "Remarks"
}
}
delete_rows {
condition {
row_existence: IGNORE
}
primary_key {
name: "CardID"
value {
type: STRING
v_string: "2007035023"
```

```
}
}
primary_key {
name: "SellerID"
value {
type: STRING
v_string: "00026"
}
}
primary_key {
name: "DeviceID"
value {
type: STRING
v_string: "065499"
}
}
primary_key {
name: "OrderNumber"
value {
type: INTEGER
v_int: 166666
}
}
}
}
}
```

Response Example:

```
tables {
table_name: "consume_history"
put_rows {
is_ok: true
consumed {
capacity_unit {
write: 1
}
}
}
update_rows {
consumed {
capacity_unit {
write: 1
}
}
}
delete_rows {
consumed {
capacity_unit {
write: 1
}
}
}
}
```

CreateTable

Action:

Creates a table based on the given table structure information.

Request Structure:

```
message CreateTableRequest {
required TableMeta table_meta = 1;
required ReservedThroughput reserved_throughput = 2;
}
```

table_meta:

Type: TableMeta

Required Parameter: Yes

The structure information of the table to be created. Here, table_name should be unique for this instance; in primary_key, the number of ColumnSchema should be between 1-4; and, in primary_key, the ColumnSchema names should comply with Table Naming Conventions and the type values can only be STRING or INTEGER.

After the table is created, the table’ s schema cannot be modified.

reserved_throughput:

Type: ProvisionedThroughput

Required Parameter: Yes

Sets the initial reserved read/write throughput for the table to be created. For any table, neither the reserved write nor read throughput can exceed 5,000.

The reserved read/write throughput settings for a table can be changed on the fly through UpdateTable.

Response Message Structure:

```
message CreateTableResponse {
}
```

Considerations:

Successfully created tables cannot immediately provide read/write services. Generally, read/write operations can be performed on the new table about one minute after creation.

There cannot be more than 10 tables under a single instance. If you need to increase the table count upper limit for a single instance, please use manual services to do it.

Request Example:

```
CreateTableRequest {
table_meta {
table_name: "consume_history"
primary_key {
name: "CardID"
type: STRING
}
primary_key {
name: "SellerID"
type: STRING
}
primary_key {
name: "DeviceID"
type: STRING
}
primary_key {
name: "OrderNumber"
type: INTEGER
}
}
capacity_unit {
read: 100
write: 100
}
}
```

Response Example:

```
CreateTableResponse {
}
```

ListTable

Action:

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.

Considerations:

If a table has just been created, its table name will appear in ListTableResponse, but this does not necessarily mean that this table can be read or written at the time.

Request Example:

```
ListTableRequest {
}
```

Response Example:

```
ListTableResponse {
table_names: "consume_history"
table_names: "card_info"
```

```
table_names: "seller_info"
}
```

DeleteTable

Action:

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, this indicates the table has been deleted.

Request Example:

```
DeleteTableRequest {
table_name: "consume_history"
}
```

Response Example:

```
DeleteTableResponse {
```

```
}
```

UpdateTable

Action:

Updates the reserved read or write throughput settings of the specified table. The new settings will take effect within one minute of a successful update.

Request Structure:

```
message UpdateTableRequest {
required string table_name = 1;
required ReservedThroughput reserved_throughput = 2;
}
```

table_name:

Type: string

Required Parameter: Yes

The name of the table for which to change the reserved read/write throughput settings.

reserved_throughput:

Type: ProvisionedThroughput

Required Parameter: Yes

The reserved read/write throughput settings for the table to change. These settings will take effect in one minute.

You may update either the table’ s reserved read or write throughput settings, or both.

In capacity_unit, read and write should at least have a non-null value. Otherwise, the request will fail and return an error.

Response Message Structure:

```
message UpdateTableResponse {
required ReservedThroughputDetails reserved_throughput_details = 1;
}
```

capacity_unit_details:

Type: ProvisionedThroughputDetails

After the update, besides containing the current reserved read/write throughput settings, the table’ s reserved read/write throughput settings information will also contain the time these settings were last updated and the number of times they have been lowered for the current date.

Tips

The minimum time interval between adjustments to a table’ s reserved read/write throughput is 10 minutes. If the current UpdateTable operation comes less than 10 minutes after the last one, it will be rejected.

Within a single calendar day (from 00:00:00 to 00:00:00 the next day in UTC time), each table’ s reserved read/write throughput may be raised an unlimited number of times, but lowered no more than 4 times. Lowering either the reserved read throughput or write throughput is viewed as lowering the reserved read/write throughput.

Request Example:

```
UpdateTableRequest {
table_name: "consume_history"
capacity_unit {
read: 10
write: 150
}
}
```

Response Example:

```
UpdateTableResponse {
capacity_unit_details {
```

```
capacity_unit {
read: 10
write: 150
}
last_increase_time: 1407507306
last_decrease_time: 1407507306
number_of_decreases_today: 2
}
}
```

DescribeTable

Action:

Queries the structure information and reserved read/write throughput settings information for 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;
}
```

table_meta:

Type: TableMeta

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

reserved_throughput_details:

Type: ProvisionedThroughputDetails

This table’ s reserved read/write throughput settings information. Besides containing the current reserved read/write throughput settings, this also contains the time these settings were last updated and the number of times they have been lowered for the current date.

Request Example:

```
DescribeTableRequest {
table_name: "consume_history"
}
```

Response Example:

```
DescribeTableResponse {
table_meta {
table_name: "consume_history"
primary_key {
name: "CardID"
type: STRING
}
primary_key {
name: "SellerID"
type: STRING
}
primary_key {
name: "DeviceID"
type: STRING
}
primary_key {
name: "OrderNumber"
type: INTEGER
}
}
capacity_unit_details {
capacity_unit {
read: 10
write: 150
}
last_increase_time: 1407507306
last_decrease_time: 1407507306
number_of_decreases_today: 2
}
}
```

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 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 the 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_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 will be 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), 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.

Action:

Obtains the information about the stream 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.

Action:

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.

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.

Action:

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.

Action:

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 string next_shard_iterator = 2;
}
```

StreamRecord:

Type: repeated StreamRecord

The record entry for reading the current shard.

shard_iterator:

Type: required string

The iterator for reading the current shard next time.

TableStore ProtocolBuffer Message Definitions

```
package com.aliyun.cloudservice.ots2;

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


enum ColumnType {
INF_MIN = 0; // only for GetRange
INF_MAX = 1; // only for GetRange
INTEGER = 2;
STRING = 3;
BOOLEAN = 4;
DOUBLE = 5;
BINARY = 6;
}

message ColumnSchema {
required string name = 1;
required ColumnType type = 2;
}

message ColumnValue {
required ColumnType type = 1;
optional int64 v_int = 2;
optional string v_string = 3;
optional bool v_bool = 4;
optional double v_double = 5;
optional bytes v_binary = 6;
}

message Column {
required string name = 1;
required ColumnValue value = 2;
}

message Row {
repeated Column primary_key_columns = 1;
repeated Column attribute_columns = 2;
}

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

```
repeated ColumnSchema primary_key = 2;
}

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

message Condition {
required RowExistenceExpectation row_existence = 1;
}

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

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;
}

message ReservedThroughput {
required CapacityUnit capacity_unit = 1;
}

message ConsumedCapacity {
required CapacityUnit capacity_unit = 1;
}

/* CreateTable */
message CreateTableRequest {
required TableMeta table_meta = 1;
required ReservedThroughput reserved_throughput = 2;
}

message CreateTableResponse {
}

/* UpdateTable */

message UpdateTableRequest {
required string table_name = 1;
required ReservedThroughput reserved_throughput = 2;
}

message UpdateTableResponse {
required ReservedThroughputDetails reserved_throughput_details = 1;
}

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

```
}

message DescribeTableResponse {
required TableMeta table_meta = 1;
required ReservedThroughputDetails reserved_throughput_details = 2;
}

/* ListTable */
message ListTableRequest {
}

message ListTableResponse {
repeated string table_names = 1;
}

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

message DeleteTableResponse {
}

/* GetRow */
message GetRowRequest {
required string table_name = 1;
repeated Column primary_key = 2;
repeated string columns_to_get = 3;
}

message GetRowResponse {
required ConsumedCapacity consumed = 1;
required Row row = 2;
}

/* UpdateRow */
enum OperationType {
PUT = 1;
DELETE = 2;
}

message ColumnUpdate {
required OperationType type = 1;
required string name = 2;
optional ColumnValue value = 3;
}

message UpdateRowRequest {
required string table_name = 1;
required Condition condition = 2;
repeated Column primary_key = 3;
repeated ColumnUpdate attribute_columns = 4;
}

message UpdateRowResponse {
required ConsumedCapacity consumed = 1;
```

```
}

/* PutRow */
message PutRowRequest {
required string table_name = 1;
required Condition condition = 2;
repeated Column primary_key = 3;
repeated Column attribute_columns = 4;
}

message PutRowResponse {
required ConsumedCapacity consumed = 1;
}

/* DeleteRow */
message DeleteRowRequest {
required string table_name = 1;
required Condition condition = 2;
repeated Column primary_key = 3;
}

message DeleteRowResponse {
required ConsumedCapacity consumed = 1;
}

/* BatchGetRow */
message RowInBatchGetRowRequest {
repeated Column primary_key = 1;
}

message TableInBatchGetRowRequest {
required string table_name = 1;
repeated RowInBatchGetRowRequest rows = 2;
repeated string columns_to_get = 3;
}

message BatchGetRowRequest {
repeated TableInBatchGetRowRequest tables = 1;
}

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

message TableInBatchGetRowResponse {
required string table_name = 1;
repeated RowInBatchGetRowResponse rows = 2; // same indices w.r.t. request
}

message BatchGetRowResponse {
repeated TableInBatchGetRowResponse tables = 1; // same indices w.r.t. request
}
```

```
/* BatchWriteRow */
message PutRowInBatchWriteRowRequest {
required Condition condition = 1;
repeated Column primary_key = 2;
repeated Column attribute_columns = 3;
}

message UpdateRowInBatchWriteRowRequest {
required Condition condition = 1;
repeated Column primary_key = 2;
repeated ColumnUpdate attribute_columns = 3;
}

message DeleteRowInBatchWriteRowRequest {
required Condition condition = 1;
repeated Column primary_key = 2;
}

message TableInBatchWriteRowRequest {
required string table_name = 1;
repeated PutRowInBatchWriteRowRequest put_rows = 2;
repeated UpdateRowInBatchWriteRowRequest update_rows = 3;
repeated DeleteRowInBatchWriteRowRequest delete_rows = 4;
}

message BatchWriteRowRequest {
repeated TableInBatchWriteRowRequest tables = 1; // same indices w.r.t. request
}

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

message TableInBatchWriteRowResponse {
required string table_name = 1;
repeated RowInBatchWriteRowResponse put_rows = 2; // same indices w.r.t. request
repeated RowInBatchWriteRowResponse update_rows = 3; // same indices w.r.t. request
repeated RowInBatchWriteRowResponse delete_rows = 4; // same indices w.r.t. request
}

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;
```

```
optional int32 limit = 4;
repeated Column inclusive_start_primary_key = 5; // required all PKs, possibly filled with INF_MIN/INF_MAX
repeated Column exclusive_end_primary_key = 6; // required all PKs, possibly filled with INF_MIN/INF_MAX
}

message GetRangeResponse {
required ConsumedCapacity consumed = 1;
repeated Column next_start_primary_key = 2; // missing means hitting the end
repeated Row rows = 3;
}
```

Data Types

DataType Summary

CapacityUnit

Column

ColumnCondition

ColumnConditionType

ColumnSchema

ColumnType

ColumnUpdate

ColumnValue

Condition

ComparatorType

CompositeCondition

ConsumedCapacity

DeleteRowInBatchWriteRowRequest

Direction

Error

LogicalOperator

OperationType

PutRowInBatchWriteRowRequest

RelationCondition

ReservedThroughput

ReservedThroughputDetails

Row

RowExistenceExpectation

RowInBatchGetRowResponse

RowInBatchGetRowRequest

RowInBatchWriteRowResponse

TableInBatchGetRowRequest

TableInBatchGetRowResponse

TableInBatchWriteRowRequest

TableInBatchWriteRowResponse

TableMeta

UpdateRowInBatchWriteRowRequest

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

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

write

Type: int32

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

Related Operations

UpdateRow

BatchWriteRow

Column

Indicates a column.

Data Structure

```
message Column {
required string name = 1;
required ColumnValue value = 2;
}
```

name

Type: string

Description: The name of this column.

value

Type: ColumnValue

Description: The value of this column.

ColumnCondition

The column conditions in Conditional Update or Filter functions.

Data Structure

```
message ColumnCondition {
required ColumnConditionType type = 1;
required bytes condition = 2;
}
```

type：

Type: ColumnConditionType.

Description: The column condition type.

condition：

Type: bytes.

Description: The binary data that is serialized by protobuf which conditional statement contains CompositeCondition or RelationCondition.

Related Operations

ConditionUpdate

PutRow

UpdateRow

DeleteRow

BatchWriteRow

Filter

GetRow

GetRange

BatchGetRow

ColumnConditionType

Column condition,enum type.

CCT_RELATION Indicates the relation, only one condition，such as column_a > 5.

CCT_COMPOSITE Indicates combination of relation, include multiple conditions,such as column_a > 5 OR column_b < 10.

Enumeration Value List

```
enum ColumnConditionType {
```

```
CCT_RELATION = 1;
CCT_COMPOSITE = 2;
}
```

ColumnSchema

Defines a column, only used for primary key columns.

Data Structure

```
message ColumnSchema {
required string name = 1;
required ColumnType type = 2;
}
```

name

Type: string

Description: The name of this column.

type

Type: ColumnType

Description: The data type of this column.

Related Operations

CreateTable

DescribeTable

ColumnType

Indicates the data type of a column, enumeration type.

INF_MIN and INF_MAX are specialized types for the GetRange operation. The value of an INF_MIN column is smaller than all other columns and the value of a INF_MAX column is larger than all other

columns.

Enumeration Value List

```
enum ColumnType {
INF_MIN = 0;
INF_MAX = 1;
INTEGER = 2;
STRING = 3;
BOOLEAN = 4;
DOUBLE = 5;
BINARY = 6;
}
```

ColumnUpdate

In UpdateRow, this indicates updating the information of a column.

Data Structure

```
message ColumnUpdate {
required OperationType type = 1;
required string name = 2;
optional ColumnValue value = 3;
}
```

type

Type: OperationType

Description: How this column is to be modified - update or delete.

name

Type: string

Description: The name of this column.

value

Type: ColumnValue

Description: The value of this column after the update; effective when the type is PUT.

Related Operations

UpdateRow

BatchWriteRow

ColumnValue

Indicates the value of one column.

Data Structure

```
message ColumnValue {
required ColumnType type = 1;
optional int64 v_int = 2;
optional string v_string = 3;
optional bool v_bool = 4;
optional double v_double = 5;
optional bytes v_binary = 6;
}
```

type

Type: ColumnType

Description: The data type of this column.

v_int

Type: int64

Description: The data of this column; only valid when the type is INTEGER.

v_string

Type: string

Description: The data of this column; only valid when the type is STRING. Must use UTF-8 encoding.

v_bool

Type: bool

Description: The data of this column; only valid when the type is BOOLEAN.

v_double

Type: double

Description: The data of this column; only valid when the type is DOUBLE.

v_binary

Type: bytes

Description: The data of this column; only valid when the type is BINARY.

Condition

The row conditions in PutRow, UpdateRow, and DeleteRow. It currently only contains the row_existence item.

Data Structure

```
message Condition {
required RowExistenceExpectation row_existence = 1;
}
```

row_existence

Type: RowExistenceExpectation

Description: The row existence check setting for this row.

column_condition：

Type: ColumnCondition.

Description: Set up column condition.

Related Operations

PutRow

UpdateRow

DeleteRow

BatchWriteRow

ComparatorType

Combination type, enum type.

CT_EQUAL Indicates equal.

CT_NOT_EQUAL Indicates not equal.

CT_GREATER_THAN Indicates greater than.

CT_GREATER_EQUAL Indicates not greater than.

CT_LESS_THAN Indicates less than.

CT_LESS_EQUAL Indicates not less 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;
}
```

CompositeCondition

Multiple combinations of conditions, such as column_a > 5 AND column_b = 10 etc. Applies to the conditional update and filter functions.

Data Structure

```
message CompositeCondition {
required LogicalOperator combinator = 1;
repeated ColumnCondition sub_conditions = 2;
}
```

combinator :

Type: LogicalOperator。

Description: Logical operator.

sub_conditions :

Type: ColumnCondition。

Description: Sub-conditional expression.

Related Operations

ConditionUpdate

PutRow

UpdateRow

DeleteRow

BatchWriteRow

Filter

GetRow

GetRange

BatchGetRow

ConsumedCapacity

Indicates capacity units consumed by an operation.

Data Structure

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

capacity_unit

Type: CapacityUnit

Description: The value of capacity units consumed by this operation.

DeleteRowInBatchWriteRowRequest

Indicates the information of the row to delete in the BatchWriteRow operation.

Data Structure

```
message DeleteRowInBatchWriteRowRequest {
required Condition condition = 1;
repeated Column primary_key = 2;
```

```
}
```

condition

Type: Condition

Description: Whether or not to perform existence check before data deletion.

primary_key

Type: repeated Column

Description: All **PrimaryKeyColumns** for the row to be deleted.

Related Operations

BatchWriteRow

Direction

The data query order in the GetRange operation.

FORWARD indicates that the query proceeds from the smallest to the largest primary key.

BACKWARD 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

Used to indicate an error message in the response message when an operation fails and to 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

Description: The error code for the current individual row operation. For details, refer to OTS Storage Exceptions.

message

Type: string

Description: The error message for the current individual row operation. For details, refer to OTS Storage Exceptions.

Related Operations

BatchGetRow

BatchWriteRow

LogicalOperator

Logical operator, enum type.

LO_NOT Indicates or.

LO_AND Indicates and.

LO_OR Indicates or.

Enumeration Value List

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

OperationType

In UpdateRow, this indicates the modification method for one column.

PUT indicates a column is inserted or this column’ s data are overwritten.

DELETE indicates this column is deleted.

Enumeration Value List

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

PutRowInBatchWriteRowRequest

Indicates the information of the row to insert in the BatchWriteRow operation.

Data Structure

```
message PutRowInBatchWriteRowRequest {
required Condition condition = 1;
repeated Column primary_key = 2;
repeated Column attribute_columns = 3;
```

```
}
```

condition

Type: Condition

Description: Whether or not to perform row existence check before data insertion.

primary_key

Type: repeated Column

Description: All primary key columns for the row to be inserted.

attribute_columns

Type: repeated Column

Description: The attributes of the row to be inserted.

Related Operations

BatchWriteRow

RelationCondition

Single condition, such as column_a > 5 etc. In ConditionUpdate 和 Filter 功能。

Data Structure

```
message RelationCondition {
required ComparatorType comparator = 1;
required string column_name = 2;
required ColumnValue column_value = 3;
required bool pass_if_missing = 4;
}
```

comparator：

Type: ComparatorType.

Description: Comparator type.

column_name：

Type: string.

Description: The column name.

column_value：

Type: ColumnValue.

Description: The column value.

pass_if_missing

Type: bool.

Description: The condition is passed or not when the column does not exist in this row. For example, the relation condition column_a>0 will be passed when column “column_a” does not exist in this row only if pass_if_missing is true.

Related Operations

ConditionUpdate

PutRow

UpdateRow

DeleteRow

BatchWriteRow

Filter

GetRow

GetRange

BatchGetRow

Provisioned Throughput

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

Description: Indicates the current reserved read/write throughput value.

Related Operations

CreateTable

UpdateRow

DescribeTable

ProvisionedThroughputDetails

Data Structure

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

Description: The provisioned throughput capacity reserved for read/write value for this table.

last_increase_time

Type: int64

Description: The last time the provisioned throughput capacity reserved for read/write settings were raised for this table, expressed in UTC time to the second.

last_decrease_time

Type: int64

Description: The last time the provisioned throughput capacity reserved for read/write settings were lowered for this table, expressed in UTC time to the second.

number_of_decreases_today

Type: int32

Description: The number of times this table’ s provisioned throughput capacity reserved for read/write settings were lowered on the current calendar day.

Related Operations

UpdateTable

DescribeTable

Row

Indicates a row of data in GetRow and GetRange response messages.

Data Structure

```
message Row {
repeated Column primary_key_columns = 1;
repeated Column attribute_columns = 2;
}
```

primary_key_columns

Type: repeated Column

Description: Indicates all primary key columns to be returned for this row.

attribute_columns

Type: repeated Column

Description: Indicates all attribute columns to be returned for this row.

Related Operations

GetRow

GetRange

BatchGetRow

RowExistenceExpectation

Conditions for row existence, enumeration type.

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 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

Indicates a row of data in the messages returned by the BatchGetRow operation.

Data Structure

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

is_ok

Type: bool

Description: Whether or not the operation for this row was successful. If true, the row was read successfully and error is invalid. If false, this row failed to be read and row is invalid.

error

Type: Error

Description: The error message for this row operation.

consumed

Type: ConsumedCapacity

Description: The capacity units consumed by this row operation.

row

Type: Row

Description: The collection of column data to be returned for this row.

Related Operations

BatchGetRow

RowInBatchGetRowRequest

Indicates the request information of the row to be read in the BatchGetRow operation.

Data Structure

```
message RowInBatchGetRowRequest {
repeated Column primary_key = 1;
}
```

primary_key

Type: repeated Column

Description: All primary key columns of the row to be read.

Related Operations

BatchGetRow

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 RowInBatchGetRowRequest rows = 2;
repeated string columns_to_get = 3;
optional ColumnCondition filter = 4;
}
```

table_name

Type: string

Description: The name of this table.

rows

Type: repeated RowInBatchGetRowRequest

Description: The information of all rows to be read in this table.

columns_to_get

Type: repeated string

The names of all columns to be returned from this table.

filter :

Type: ColumnCondition.

Optional or required: optional.

Description: The filter expression.

Related Operations

BatchGetRow.

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

Description: The name of this table.

rows

Type: repeated RowInBatchGetRowResponse

Description: All row data read in this table.

Related Operations

BatchGetRow

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 PutRowInBatchWriteRowRequest put_rows = 2;
repeated UpdateRowInBatchWriteRowRequest update_rows = 3;
repeated DeleteRowInBatchWriteRowRequest delete_rows = 4;
}
```

table_name

Type: string

Description: The name of this table.

put_rows

Type: repeated PutRowInBatchWriteRowRequest

Description: The information of the row to be inserted in this table.

update_rows

Type: repeated UpdateRowInBatchWriteRowRequest

Description: The information of the row to be updated in this table.

delete_rows

Type: repeated DeleteRowInBatchWriteRowRequest

Description: The information of the row to be deleted in this table.

Related Operations

BatchWriteRow

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

Description: The name of this table.

put_rows

Type: RowInBatchWriteRowResponse

Description: The results of the PutRow operation for this table.

update_rows

Type: RowInBatchWriteRowResponse

Description: The results of the UpdateRow operation for this table.

delete_rows

Type: RowInBatchWriteRowResponse

Description: The results of the DeleteRow operation for this table.

Related Operations

BatchWriteRow

UpdateRowInBatchWriteRowRequest

Indicates the information of the row to be updated in the BatchWriteRow operation.

Data Structure

```
message UpdateRowInBatchWriteRowRequest {
required Condition condition = 1;
repeated Column primary_key = 2;
repeated ColumnUpdate attribute_columns = 3;
}
```

condition

Type: Condition

Description: Whether or not to perform row existence check before data update.

primary_key

Type: repeated Column

Description: All primary key columns of the row to be updated.

attribute_columns

Type: repeated ColumnUpdate

Description: All attribute columns to be updated.

Related Operations

BatchWriteRow

TableMeta

Indicates the structure information of a table.

Data Structure

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

table_name

Type: string

Description: The name of this table.

primary_key

Type: repeated ColumnSchema

Description: All primary key columns for this table.

Related Operations

CreateTable

DescribeTable