# Table Store

## Quick Start

# Quick Start

# From SQL to NoSQL

# Comparison between SQL and NoSQL

NoSQL is a term used to describe the highly-scalable, high-performance non-relational databases. NoSQL databases provide schemafree data models, so that applications do not require a predefined table structure. These databases are suitable for businesses that are diversified. Also, their support for ultra-large data volumes and high concurrency makes NoSQL databases suitable for a wide range of applications.

## Comparison of SQL and NoSQL databases

| | Relational database | NoSQL database |
|---|---|---|
| Data model | Relational models normalize data, strictly defining tables, columns, indexes, the relations between tables, and other database elements. This means that all data in a data table has the same structure. | NoSQL databases generally do not strictly define the table structure. Usually, partition keys and key values are used to retrieve values, column sets, or semi-structured data. |
| ACID | Conventional relational databases support a group of attributes defined by ACID (Atomicity, Consistency, Isolation and Durability). Atomicity means that a transaction either succeeds or fails completely. Consistency means that database transactions cannot undermine the integrity of related data and the consistency of the business logic. Isolation requires that concurrent transactions be executed separately, so they | In order to achieve a more flexible horizontally-scalable data model, NoSQL databases generally forgo some of the ACID attributes of conventional relational databases. With these features, NoSQL databases can be used to overcome the problems of performance bottlenecks, scalability, operational complexity, and constantly increasing management and support costs. This makes NoSQL databases superior to |

| | | |
|---|---|---|
| | do not interfere with each other. Durability means that, once a transaction is submitted, its modifications will be permanently saved in the database. They will not be lost even if the server goes down. | conventional relational databases when facing the challenges of massive data volumes and high concurrency. |
| Performance | Database performance is generally determined by the underlying storage, dataset size, query optimization, index and table structure. | Write performance is generally limited by the underlying storage, while read performance is limited by the size of the results set. |
| Scaling | For vertical scaling, the simplest method is to deploy better hardware with increased CPU speed and greater disk space. The use of related tables across distributed systems requires an increase in usage costs and technical complexity. | Horizontal scaling can be achieved using distributed clusters of low-cost hardware. Therefore, the throughput and data scale can be increased without increasing the latency. |
| APIs | Data storage and retrieval requests are sent using queries that comply with structured query language (SQL). These queries are parsed and executed by the relational database system. | Application developers can use NoSQL database Open APIs to conveniently store and retrieve data. Using partition keys and key values, applications can query key value pairs, column sets, and semi-structured data. |

# Why should I use Table Store?

Table Store is a type of NoSQL database. It provides massive NoSQL data storage capabilities, supports schemafree data models, and provides single-row transactions. The server automatically performs partitioning and load balancing for data. It can be easily scaled up when individual tables are increased from **1 gigabyte** to **1 terabyte**, and even to **1 petabyte**, or when concurrent access requests surge from **0** to **1 million**. The server writes **terabytes** and **petabytes** of data within one millisecond. The read performance depends on the size of the results set, rather than the overall table size.

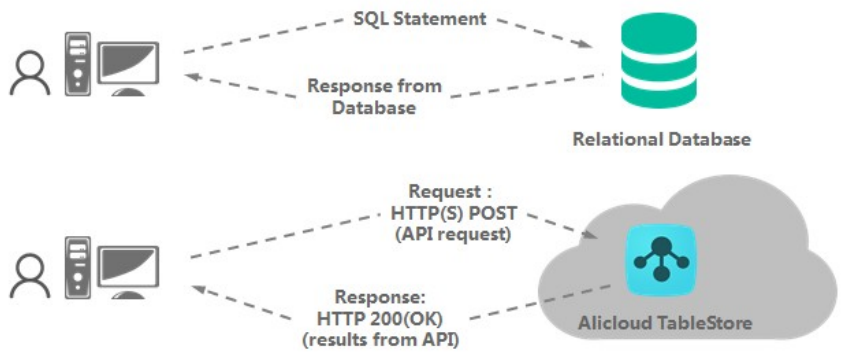
Therefore, in OLTP (OnLine Transaction Processing) scenarios, Table Store is better suited for web applications, for example, social networks, games, media sharing, IoT, and log monitoring.

To learn more about the Table Store data model, see Data Model Concepts.

# Database access

Different from conventional relational databases, the client accesses Table Store over HTTP.

The figure below shows the interaction between a client and a relational database and that between a client and Table Store.




The client accesses Table Store through a Restful API in an HTTP packet. The Table Store server verifies the signature in the packet. For more details, refer to Using Table Store APIs. If you use the SDK provided by Alibaba Cloud, you can call the SDK APIs to perform data operations as long as you have the provided Table Store endpoint, instance and access key.

Table Store accesses data from a client. The client must be initialized as follows:

```
    final String endPoint = "";
final String accessKeyId = "";
final String accessKeySecret = "";
final String instanceName = "";
SyncClient client = new SyncClient(endPoint, accessKeyId, accessKeySecret, instanceName);
```

To initialize the client, you must enter the following parameters:

endPoint: The URL used to access the Table Store instance.

accessKeyId: The AccessKeyId used to access Table Store.

accessKeySecret: The AccessKeySecret used to access Table Store.

instanceName: The instance name used to access Table Store.

For more information, refer to Access control.

# Create a table

A table is the basic data structure in both RDS and Table Store. When creating an RDS table, you must define the entire data structure. In contrast, you only need to define the primary keys when creating a Table Store table.

## SQL

In the example below, we will use the CREATE TABLE statement to create a table.

```
CREATE TABLE UserHistory (
user_id VARCHAR(20) NOT NULL,
time_stamp INT NOT NULL,
item_id VARCHAR(50),
behavior_type VARCHAR(10),
behavior_amount DOUBLE,
behavior_count INT,
content VARCHAR(100),
PRIMARY KEY(user_id, time_stamp)
);
```

This table’ s primary keys include **user_id** and **time_stamp**. When creating a data table, you must strictly define all primary key and attribute columns. If needed, you can use the ALTER TABLE statement to modify these definitions.

## Table Store

Here, we will use Table Store to create a data table and set the following parameters:

```
    public static final String TABLE_NAME = "UserHistory";

TableMeta tableMeta = new TableMeta(TABLE_NAME);
tableMeta.addPrimaryKeyColumn(new PrimaryKeySchema("user_id", PrimaryKeyType.STRING));
tableMeta.addPrimaryKeyColumn(new PrimaryKeySchema("time_stamp", PrimaryKeyType.INTEGER));

// The data expiration time in seconds; -1 indicates data do not expire. To set the expiration time to 1 year, use 365
* 24 * 3600
int timeToLive = -1;

// The max number of versions to save. If it is set to 3, each column will only save the latest three versions
int maxVersions = 3;

TableOptions tableOptions = new TableOptions(timeToLive, maxVersions);
CreateTableRequest request = new CreateTableRequest(tableMeta, tableOptions);

// Set the reserved read/write throughput. If not set, the reserved throughput is set to 0 by default
request.setReservedThroughput(new ReservedThroughput(new CapacityUnit(1, 1)));
```

```
client.createTable(request);
```

This table’ s primary keys include **user_id** and **time_stamp**. You must provide the following parameters:

TABLE_NAME: The name of the table.

PrimaryKeySchema: The names and types of primary keys.

timeToLive: The expiration time of data in the data table.

maxVersion: The maximum number of versions retained by each attribute column.

ReservedThroughtput: The data table’ s reserved read/write throughput. Reserved throughput cannot be set for container-type instances.

# Write data

## SQL

In RDS, a table is a two-dimensional data structure composed of rows and columns. You can use the INSERT statement to add rows to a table, as shown below:

```
INSERT INTO UserHistory (
user_id, time_stamp, item_id, behavior_type,
behavior_amount, behavior_count, content)
VALUES(
'10100', 1479265526, 'cell_phone', 'share', 4.9, 78,
'The phone is quit good!'
);
```

## Table Store

In Table Store, you can use the PutRow API to insert a row of data, as shown below:

```
    // Set primary keys
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.fromLong(1479265526));
```

```
PrimaryKey primaryKey = primaryKeyBuilder.build();

// Set the attribute column value
RowPutChange rowPutChange = new RowPutChange(TABLE_NAME, primaryKey);
rowPutChange.addColumn(new Column("item_id", ColumnValue.fromString("cell_phone")));
rowPutChange.addColumn(new Column("behavior_type", ColumnValue.fromString("share")));
rowPutChange.addColumn(new Column("behavior_amount", ColumnValue.fromDouble(4.9)));
rowPutChange.addColumn(new Column("behavior_count", ColumnValue.fromLong(78)));
rowPutChange.addColumn(new Column("content", ColumnValue.fromString("The phone is quit good!")));

// Insert this row of data
client.putRow(new PutRowRequest(rowPutChange));
```

**Notes in using the PutRow API:**

Except for TABLE_NAME and primaryKey, attribute columns and types can be defined during data writing.

Attribute columns of the same names in multiple rows of data do not have to be of the same types.

Big data SQL databases are transaction-oriented. When an INSERT statement is issued, the data modifications will be made permanently only after the COMMIT statement is received. In Table Store, when the HTTP 200 (OK) status code is returned, this means the data written by PutRow have been made permanently in all copies.

You can use the BatchWriteRow API to insert multiple records, significantly improving the data writing speed.

# Query data

## SQL

You can use the SQL SELECT statement to query primary key columns, non-primary key columns, or any specified columns. The WHERE clause confirms the rows to return, as shown in the example below:

```
// Query one row based on a primary key
SELECT * FROM UserHistory
WHERE user_id = '10100' AND time_stamp = 1479265526;

// Query all data for a specified user_id
```

```
SELECT * FROM UserHistory
WHERE user_id = '10100';

// Query all records of a specified user_id in a specified time period
SELECT * FROM UserHistory
WHERE user_id = '10100' AND time_stamp > 1478660726 AND time_stamp < 1479265526;

// Query all favorite records for a specified user_id
SELECT * FROM UserHistory
WHERE user_id = '10100' AND behavior_type = 'collect';
```

# Table Store

In Table Store, the data query API can be used to retrieve data in a similar way. For single row queries, the GetRow and GetRange APIs provide a fast and efficient way to access the physical location of stored data. In this case the query performance is only affected by the size of the resulting dataset, rather than the total volume of data in the table.

By providing complete primary key information, you can use the GetRow API to quickly query the specified row of data, as shown below:

```
  // SELECT * FROM UserHistory WHERE user_id = '10100' AND time_stamp = 1479265526;
// Set primary key information
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn('user_id', PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn('time_stamp', PrimaryKeyValue.fromLong(1479265526));
PrimaryKey primaryKey = primaryKeyBuilder.build();

// Read a row
SingleRowQueryCriteria criteria = new SingleRowQueryCriteria(TABLE_NAME, primaryKey);
// Set the latest version to be read
criteria.setMaxVersions(1);
GetRowResponse getRowResponse = client.getRow(new GetRowRequest(criteria));
```

You can use the GetRange API to query all data of a specified user_id, as shown below:

```
  // Equivalent to SELECT * FROM UserHistory WHERE user_id = '10100'
RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(TABLE_NAME);
// Set StartPrimaryKey
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.INF_MIN);
rangeRowQueryCriteria.setInclusiveStartPrimaryKey(primaryKeyBuilder.build());

// Set EndPrimaryKey
primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.INF_MAX);
rangeRowQueryCriteria.setExclusiveEndPrimaryKey(primaryKeyBuilder.build());
```

```
// Set the latest version to be read
rangeRowQueryCriteria.setMaxVersions(1);

// Read all attribute columns by default
GetRangeResponse getRangeResponse = client.getRange(new
GetRangeRequest(rangeRowQueryCriteria));
```

**Note:**

When using the GetRange API, you must specify the start point for all primary keys, **but the ranges of each primary key are not connected by the AND operator**. Also, when sorting rows from the first to the last primary key, priority is given to the primary key added first. When this primary key is in the GetRange operation’s start-end primary key range, data from the corresponding row will be read. For example, if the range for two primary keys is (‘a’,5)-(‘c’,10), data with the primary key (‘b’,4) will be read because 'a' < 'b' < 'c'.

The INF_MIN and INF_MAX are specialized types for the GetRange operation, which specify the minimum and maximum values respectively.

GetRange supports the limit and direction parameters, which control the number of rows in the results set and the order in which rows are read.

To prevent network delays, the GetRange API implements a limit on the returned results set. It also judges the next_start_primary_key returned in the response. If this is blank, it indicates all results have been returned; otherwise, the API continues reading results.

GetRange supports the Filter function.

Table Store supports the Multiple Data Versions function. When using the GetRow and GetRange APIs, you can specify the range of historical attribute column data versions to read.

You can use the GetRange API to query all data from a specified time period, as shown below:

```
 // SELECT * FROM UserHistory WHERE user_id = '10100' AND time_stamp >= 1478660726 AND
time_stamp < 1479265526;
RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(TABLE_NAME);
// Set StartPrimaryKey
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
```

```
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.fromLong(1478660726));
rangeRowQueryCriteria.setInclusiveStartPrimaryKey(primaryKeyBuilder.build());

// Set EndPrimaryKey
primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.fromLong(1479265526));
rangeRowQueryCriteria.setExclusiveEndPrimaryKey(primaryKeyBuilder.build());

// Set the latest version to be read
rangeRowQueryCriteria.setMaxVersions(1);

// Read all attribute columns by default
GetRangeResponse getRangeResponse = client.getRange(new
GetRangeRequest(rangeRowQueryCriteria));
```

This query is equivalent to:

```
SELECT * FROM UserHistory
WHERE user_id = '10100' AND time_stamp > 1478660726 AND time_stamp < 1479265526;
```

To perform condition checks on attribute columns, you can use the Filter Function. For example, to query all favorite records for a specified user_id:

```
  // SELECT * FROM UserHistory WHERE user_id = '10100' AND behavior_type = 'collect';
RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(TABLE_NAME);
// Set StartPrimaryKey
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.INF_MIN);
rangeRowQueryCriteria.setInclusiveStartPrimaryKey(primaryKeyBuilder.build());

// Set EndPrimaryKey
primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.INF_MAX);
rangeRowQueryCriteria.setExclusiveEndPrimaryKey(primaryKeyBuilder.build());

// Set the attribute column filter condition: behavior_type = 'collect'
SingleColumnValueFilter filter = new SingleColumnValueFilter("behavior_type",
SingleColumnValueFilter.CompareOperator.EQUAL, ColumnValue.fromString("collect"));
// Table Store is a schemafree model, so some rows do not have the attribute column behavior_type
// Set to false, indicating that, if this row does not have the attribute column behavior_type, it does not
satisfy the conditions
filter.setPassIfMissing(false);
rangeRowQueryCriteria.setFilter(filter);

// Set the latest version to be read
rangeRowQueryCriteria.setMaxVersions(1);

// Read all attribute columns by default
```

```
GetRangeResponse getRangeResponse = client.getRange(new
GetRangeRequest(rangeRowQueryCriteria));
```

This query is equivalent to:

```
SELECT * FROM UserHistory
WHERE user_id = '10100' AND behavior_type = 'collect';
```

You can also achieve this using the following method:

```
 // SELECT * FROM UserHistory WHERE user_id = '10100' AND behavior_type = 'collect';
RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(TABLE_NAME);
// Set StartPrimaryKey
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.INF_MIN);
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.INF_MIN);
rangeRowQueryCriteria.setInclusiveStartPrimaryKey(primaryKeyBuilder.build());

// Set EndPrimaryKey
primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.INF_MAX);
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.INF_MAX);
rangeRowQueryCriteria.setExclusiveEndPrimaryKey(primaryKeyBuilder.build());

// Set the data filer conditions: user_id='10100' and behavior_type = 'collect'
SingleColumnValueFilter filter1 = new SingleColumnValueFilter("user_id",
SingleColumnValueFilter.CompareOperator.EQUAL, ColumnValue.fromString("10100"));
SingleColumnValueFilter filter2 = new SingleColumnValueFilter("behavior_type",
SingleColumnValueFilter.CompareOperator.EQUAL, ColumnValue.fromString("collect"));
CompositeColumnValueFilter filter = new
CompositeColumnValueFilter(CompositeColumnValueFilter.LogicOperator.AND);
filter.addFilter(filter1);
filter.addFilter(filter2);
rangeRowQueryCriteria.setFilter(filter);
```

This operation scans the entire table and finds records that meet the conditions user_id='10100' AND behavior_type='collect'. However, as this is a full table scan, it is much less efficient than a query based on a specified primary key range.

**Note:**

Filters support combinations of up to 10 conditions and can be used in the GetRow, BatchGetRow, and GetRange APIs.

Filters are applied to GetRange data at the server side, so they do not reduce the number of disk I/O operations. However, filters effectively reduce traffic transmitted over the network.

A good primary key design significantly improves the efficiency of range queries.

# Update data

## SQL

In RDS, you can use the UPDATE statement to modify one or more rows in a table, as shown below:

```
UPDATE UserHistory
SET behavior_type = 'collect'
WHERE user_id = '10100' AND time_stamp = 1479265526 AND behavior_count > 4.0;
```

## Table Store

In Table Store, you can use the UpdateRow API to update one row of data, as shown below:

```
    // Set primary keys
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.fromLong(1479265526));
PrimaryKey primaryKey = primaryKeyBuilder.build();

// Set the update conditions. Here, we expect the row to already exist and will only update the row if the
behavior_count value is greater than 4.0
Condition condition = new Condition(RowExistenceExpectation.EXPECT_EXIST);
condition.setColumnCondition(new SingleColumnValueCondition("behavior_count",
SingleColumnValueCondition.CompareOperator.GREATER_THAN, ColumnValue.fromDouble(4.0)));
rowUpdateChange.setCondition(condition);

// Set the attribute column value
RowUpdateChange rowUpdateChange = new RowUpdateChange(TABLE_NAME, primaryKey);
rowUpdateChange.put(new Column("behavior_type", ColumnValue.fromSting("collect")));

// Insert this row of data
client.updateRow(new UpdateRowRequest(rowUpdateChange));
```

**Note:**

Using UpdateRow, you must specify the table name TABLE_NAME and all primary keys primaryKey. Also, the column to be updated may or may not already exist.

UpdateRow will only modify the columns to be modified in a single row. PutRow can be used

to overwrite all data in an existing row.

For the update operation, you can set two types of check conditions: Row Existence Check and Condition Checks.

In SQL, the UPDATE statement will update all records that satisfy the WHERE condition. Table Store’ s UpdateRow API will only update the one row of data with the specified primary key. The condition check function will only check the primary key and attribute columns of this row.

You can use the BatchWriteRow API to update multiple records, significantly improving the data writing speed.

# Delete data

## SQL

In RDS, you can use the DELETE statement to delete one or more rows of data from a table, as shown below:

```
DELETE FROM UserHistory
WHERE user_id = '10100' and time_stamp = 1479265526;
```

## Table Store

In Table Store, you can use the DeleteRow API to delete one row of data at a time, as shown below:

```
    // Set primary keys
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("user_id", PrimaryKeyValue.fromString("10100"));
primaryKeyBuilder.addPrimaryKeyColumn("time_stamp", PrimaryKeyValue.fromLong(1479265526));
PrimaryKey primaryKey = primaryKeyBuilder.build();

RowDeleteChange rowDeleteChange = new RowDeleteChange(TABLE_NAME, primaryKey);

// Delete this row of data
client.deleteRow(new DeleteRowRequest(rowDeleteChange));
```

**Note:**

Using DeleteRow, you must specify the table name TABLE_NAME and all primary keys primaryKey. Also, you can only delete one row of data.

Just like UpdateRow, DeleteRow supports Condition Checks.

In SQL, the DELETE statement will delete all records that satisfy the WHERE condition. Table Store's DeleteRow API will only delete the one row of data with the specified primary key. The condition check function will only check the primary key and attribute columns of this row.

You can use the BatchDeleteRow API to delete multiple records, significantly improving the data deletion speed.

# Delete a table

## SQL

In RDS, you can use the DROP TABLE statement to delete data tables that are no longer needed, as shown below:

```
DROP TABLE UserHistory;
```

## Table Store

In Table Store, you can use the DeleteTable API to delete a data table at a time, as shown below:

```
    DeleteTableRequest request = new DeleteTableRequest(TABLE_NAME);
client.deleteTable(request);
```

**Note: Once a table is deleted, it cannot be recovered.**

# Operations of Table Store

Table Store supports the following three types of operations.

# Table operations

ListTable: List all tables in the current instance.

CreateTable: Create a table.

DeleteTable: Delete a table.

DescribeTable: Get the attribute information of a table.

UpdateTable: Update the configuration of the reserved read/write throughput of a table.

For the detailed information, refer to Tables of Table Store.

# Data operations

The data operations of Table Store could be classified into the following three types:

Single-row operations:

GetRow: Read the data of a single row.

PutRow: Insert a new row. If the row already exists, the existed row will be deleted first and the inserted row will be written as a new row.

UpdateRow: Update a row. The operation can add and delete the Attribute columns in the row, or update the value of the Attribute columns that already exists. If such row does not exist, the operation will add a new row.

DeleteRow: Delete a row.

Batch operations:

BatchGetRow: Batch operation of multiple GetRow operations, which will read several rows of data from one or more tables.

BatchWriteRow: Batch operation of multiple PutRow, UpdateRow and DeleteRow operations, which will insert, update and delete several rows of data from one or

more tables.

Read range:

- GetRange: Read data within a continuous range of the primary key columns in the table.

## Writing operations

The writing operations of Table Store have the following features:

Atomicity

The results of PutRow, UpdateRow and DeleteRow operations guarantee atomicity, as those operations shall either succeed or fail without any intermediate state.

Strong consistency

When a data writing operation receives the success response, all copies of the data in the distributed file system have been already updated. Any reading operation will then fetch the latest data of the written row.

## Note

Table Store provides BatchWriteRow operation to batch process the write operations of several single rows. The operation can contain all types of the single operation, including PutRow, UpdateRow and DeleteRow. BatchWriteRow operation does not guarantee atomicity as a whole, meanwhile each single writing operation in it guarantees atomicity. Therefore, the response of one batch writing operation may contain several success or failure responses for each single writing operation.

For more information, refer to Data operations of Table Store.

# Table Store activation procedure

## Preparation

Check that you have activated an Alibaba Cloud account and logged on with the account (to activate an account, click here).

Read the Table Store pricing carefully.

**Note:** Billing based on the reserved read/write throughput starts immediately after a table is created if the value of reserved read/write throughput is not 0.

## Operation procedure

Log on to the Table Store console of Alibaba Cloud or the official website of Table Store.

Click **Purchase now**.

Check **Accept Service Terms** and click **Activate now**.

# Use the Table Store console

# Table Store console overview

The Table Store (formerly called OTS) console has a graphical user interface (GUI) allowing you to perform the following operations:

Create, update and release instances.

Create, update and delete tables.

View and monitor table data.

# Instance management

## Create an instance

Log on to the Table Store console (the **Instance List** page).

Select a region.

**Note**: When you select a region, the instance specifications supported in this region are shown on the page.

Click **Create Instance** in the top-right corner.

Fill in the required information and select the instance specification.

**Note**:

Follow the instance naming conventions.

Up to 10 instances can be created under a single Alibaba Cloud account. The instance name must be globally unique.

The instance specification can not be modified after the instance is created. On how to select the instance specification, refer to Product Introduction-Region and instance.

Click **Confirm** and wait for several seconds. If the newly created instance does not appear in the instance list, click **Refresh** to refresh the **Instance List** page.

# Manage an instance

Log on to the Table Store console (the **Instance List** page).

Select the instance to be managed and click its name or the **Manage** button in the Action column to go to the **Instance Detail** page. You can do the following management:

The **Instance Detail** page shows the information about the instance, such as instance access URL, VPC list and table list.

You can change the instance network type, bind VPC and create tables.

# Release an instance

Log on to the Table Store console (the **Instance List** page).

Select the instance to be managed and click **Release** in the Action column.

**Note:** You must delete all the tables in the instance first before you release this instance.

Click **Release** in the confirmation dialog box.

# Table management

## Create a table

Log on to the Table Store console (the **Instance List** page).

Locate the target instance and click its name or **Manage** in the Action column to go to the **Instance Detail** page.

Click **Create Table**.

**Note:** Up to 64 tables can be created in an instance.

Fill in the required information and pay attention to the following issues:

The table name must be unique at the instance level.

The tables in the high-performance instance (currently does not support MaxVersions) and capacity instance have different attributes information.

If the table is created in a high-performance instance, you must specify the reserved read throughput and write throughput.

**Note:** Billing based on the reserved throughput starts immediately after the table is created. The reserved throughput can be set to 0 CU. Configuring the reserved read/write throughput does not affect the table’ s read/write performance and service capability.

If the table is created in a capacity instance, you must specify TTL

(minimum: 86400s or –1), MaxVersions and MaxTimeDeviation. For details, refer to the data model description in Product Introduction - Concepts explanation.

Set the table’ s Primary Key columns (Columns 1 – 4). The key type can be Integer or String. Once set, the configuration and order of the Primary Keys cannot be changed. Click **Add a table Primary Key** to add the Primary Keys.

Click **Confirm**.

The page returns to the **Instance Detail** page automatically. After the table is successfully created, it is displayed in the table list. You can click **Refresh** to refresh the table list.

# Update a table

The Table Store console allows you to modify the reserved read/write throughput of a table created in a high-performance instance, and to modify the TTL, MaxVersions and MaxTimeDeviation of a table created in a capacity instance.

**Modify the reserved read/write throughput of a table created in a high-performance instance**

Log on to the Table Store console (the **Instance List** page).

Locate the target instance and click its name or **Manage** in the Action column to go to the **Instance Detail** page.

Locate the table to be updated in the table list, and click **adjustTableAttr** in the Action column.

Fill in the new **Reserved read throughputs** and **Reserved write throughputs** according to relevant rules.

Click **Confirm** and return to the **Instance Detail** page. The new reserved read/write throughputs take effect immediately.

**Modify the TTL, MaxVersions and MaxTimeDeviation of a table created in a capacity instance**

Log on to the Table Store console (the **Instance List** page).

Locate the target instance and click its name or **Manage** in the Action column to go to the

**Instance Detail** page.

Locate the table to be updated in the table list and click **adjustLifeCycle** in the Action column.

Fill in the new TTL, MaxVersion and MaxTimeDeviation according to relevant rules.

Click **Confirm** and return to the **Instance Detail** page. The new TTL, MaxVersions and MaxTimeDeviation take effect immediately.

# Delete a table

Log on to the Table Store console (the **Instance List** page).

Locate the target instance and click its name or **Manage** in the Action column to go to the **Instance Detail** page.

Locate the table to be deleted in the table list and click **Delete** in the Action column.

Click **Delete** and the table will be deleted permanently.

**Note：** Data within the table can not be retrieved after the table is deleted.

# Table information and monitoring indicators

## Basic information

You can view the basic information and actual usage of your tables on the table management page (table list). For example:

The table name

The reserved read/write throughput

The last adjusting time

Primary Keys (sorted in the order specified during table creation)

## Monitoring indicators

You can view the monitoring indicators of a table, such as the source data size, QPS, the read/write throughputs and the average response latency.

**Source data size:** It indicates the size of data that you save to Table Store and is measured in MB. The size calculated by Table Store may be slightly larger because it includes incremental data. The incremental data includes all column names and values that are saved along with each row of data to Table Store.

**Note:** The shorter the name of a Primary Key column or Attribute column is, the smaller the expansion ratio is.

Monitoring latency: 3 hours.

Collection granularity at metric points: 5 minutes.

**QPS:** The server QPS (unit: PV/s) can be reflected by various operating curves.

Monitoring latency: 10 minutes.

Collection granularity at metric points: 30 seconds.

**Read/Write throughputs:** It includes the actual read throughput and the actual write throughput, which can be reflected by various operating curves. The throughput is measured in CU.

Monitoring latency: 30 seconds.

Collection granularity at metric points: 30 seconds.

**Average response latency:** It indicates the latency (unit: ms) of returning results from the server.

Monitoring latency: 30 seconds.

Collection granularity at metric points: 30 seconds.

## Operation procedure

Log on to the Table Store console (the **Instance List** page).

**Note:** Before a subaccount is used to log on to the Table Store console to view the monitoring indicators, you need to use the primary account to log on to RAM - Users management console to authorize the permission of Alibaba Cloud Monitor Read Only Access to the subaccount. If the permission is not authorized, console logon with the subaccount fails. For the detailed steps, refer to Authorization Management - Use cases.

Locate the target instance and click its name or **Manage** in the Action column to go to the **Instance Detail** page.

Locate the table to be viewed in the table list and click **Manage** in the Action column to go to the **Detail** page, where you can view the basic information of the table.

Click **Monitoring indicators** in the left navigation bar, and then click the corresponding sub-menu to view the table’ s source data size, QPS, the read/write throughputs and the average response latency.

# VPC user guide

Virtual Private Cloud (VPC) is an isolated network environment built on Alibaba Cloud, allowing you to take full control of your virtual network. For example, you can select a private IP address range, divide network segment, configure routing table and gateway, and so on. It also allows you to connect VPC to a traditional data center through the leased lines VPN to build a custom network environment, thereby achieving the smooth cloud migration.

## Preparations

Create a VPC

When you create a VPC, select a suitable region and ensure that your VPC and the Table Store instance are located in the same region. For more information, refer to Create a VPC.

Create an ECS instance in the VPC

After the VPC is successfully created, create an ECS instance in the VPC. For more information, refer to Create an ECS instance.

## VPC operation guide

After you create a VPC and a VPC ECS instance, log on to the Table Store console to create an

instance and bind the instance. The steps are as follows:

Log on to the Table Store console.

Select the region where the created VPC is located, and then click **Create Instance**.

Fill in the required information and click **Confirm**. The page returns to the **Instance List** page automatically.

Select the instance created just now and click its name or the **Manage** button to its right in the Action column to go to the **Instance Detail** page.

Click **Bind VPC** to go to the instance and VPC binding page.

Fill in required information and click **Confirm**.

When the instance and VPC are bound successfully, the page returns to the **Instance Detail** page automatically. You can find the information of the bound VPC in the **VPC list**. You can also click the link in the VPC ID column to go to the VPC instance list page, where you can view the Table Store instance bound to the VPC and the VPC information list.

Then you can use the bound instance to access Table Store from the ECS instance in the VPC. The endpoint for access can be the VPC IP address or VPC access address, but a domain name is recommended.

To delete the binding relationship between the Table Store instance and VPC, select the VPC and click **Unbind** to its right in the VPC list.

After that, the preceding address pair cannot be used to access Table Store from the ECS instance in the VPC. If you need to access Table Store again, rebind the Table Store instance to the VPC.

# Change the network type of an instance

Table Store defines three network types for instances. You can change the netework type of an instance according to the following steps.

Log on to the Table Store console.

Select the region where the created VPC is located, and then click **Create Instance**.

Fill in the required information and click **Confirm**. The page returns to the **Instance List** page automatically.

Select the instance to be managed and click its name or the **Manage** button in the Action column to go to the **Instance Detail** page.

Click the **Change network** button next to **Instance network type**.

Select a network and click **Confirm**.