

# Alibaba Cloud

ApsaraDB for HBase  
HBase Enhanced  
Edition(Lindorm)









Document Version: 20201127

# Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).
5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please directly contact Alibaba Cloud for any errors of this document.

# Document conventions

| Style  | Description   | Example   |
|--|---|---|
|  <b>Danger</b>  | A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results. |  <b>Danger:</b><br>Resetting will result in the loss of user configuration data.                                       |
|  <b>Warning</b> | A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results. |  <b>Warning:</b><br>Restarting will cause business interruption. About 10 minutes are required to restart an instance. |
|  <b>Notice</b>  | A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.      |  <b>Notice:</b><br>If the weight is set to 0, the server no longer receives new requests.                              |
|  <b>Note</b>  | A note indicates supplemental instructions, best practices, tips, and other content.  |  <b>Note:</b><br>You can use Ctrl + A to select all files.  |
| >  | Closing angle brackets are used to indicate a multi-level menu cascade.   | Click <b>Settings&gt; Network&gt; Set network type</b> .  |
| <b>Bold</b>  | Bold formatting is used for buttons , menus, page names, and other UI elements.   | Click <b>OK</b> .   |
| <code>Courier font</code>  | Courier font is used for commands   | Run the <code>cd /d C:/window</code> command to enter the Windows system folder.  |
| <i>Italic</i>  | Italic formatting is used for parameters and variables.   | <code>bae log list --instanceid</code><br><i>Instance_ID</i>  |
| [ ] or [a b]   | This format is used for an optional value, where only one item can be selected.   | <code>ipconfig [-all -t]</code>   |
| { } or {a b}   | This format is used for a required value, where only one item can be selected.  | <code>switch {active stand}</code>  |

# Table of Contents

|  |    |
|--|----|
| 1.Product Introduction                             | 06 |
| 1.1. Overview                                      | 06 |
| 1.2. Benefits                                      | 06 |
| 2.Quick start                                      | 09 |
| 2.1. Connect to a cluster                          | 09 |
| 2.2. Install the SDK for Java                      | 10 |
| 2.3. Use the Java API to access ApsaraDB for HBase | 11 |
| 2.4. Use HBase Shell to access ApsaraDB for HBase  | 15 |
| 2.5. Non-java APIs                                 | 16 |
| 2.6. Limits  | 19 |
| 2.7. Upgrade minor version                         | 19 |
| 3.Data channel                                     | 21 |
| 3.1. Use DataWorks or DataX to import data         | 21 |
| 3.2. Use Spark to access HBase                     | 25 |
| 3.3. Use Hive to access HBase                      | 28 |
| 3.4. Use Flink to access HBase                     | 33 |
| 4.Enterprise Features                              | 35 |
| 4.1. Cold storage                                  | 35 |
| 4.2. Cold and hot data separation                  | 39 |
| 4.3. High-performance native secondary indexes     | 47 |
| 5.Performance White paper                          | 55 |
| 5.1. Test environment                              | 55 |
| 5.2. Test tool                                     | 55 |
| 5.3. Benchmark methods                             | 58 |
| 5.4. Benchmark results                             | 66 |
| 6.Cluster management                               | 68 |

---

|                                      |    |
|--------------------------------------|----|
| 6.1. Cluster management system ..... | 68 |
| 6.2. Manage groups .....             | 69 |
| 6.3. Manage namespaces .....         | 73 |
| 6.4. Manage users and ACL .....      | 74 |
| 6.5. Data query .....                | 78 |
| 7.SQL manual .....                   | 81 |
| 7.1. Manage connect strings .....    | 81 |
| 7.2. Data types .....                | 82 |
| 7.3. DDL syntax .....                | 83 |
| 7.4. DML syntax .....                | 88 |
| 7.5. Manage indexes .....            | 90 |
| 7.6. Advanced features .....         | 94 |
| 7.7. Considerations and limits ..... | 97 |

# 1.Product Introduction

## 1.1. Overview



### Cloud HBase Enhanced Edition

7<sub>x</sub>

Performance

1/2

Cost



1/10

P99 Latency

10<sub>x</sub>

MTTR

ApsaraDB for HBase Enhanced Edition(Lindorm) is a distributed database engine developed based on Alibaba HBase. It has been tested for many years and is completely compatible with native HBase databases. Compared with HBase Community Edition, Performance-enhanced Edition is much easier to use, and more secure and stable. It provides higher throughput and lower latency, and is more cost-effective. Performance-enhanced Edition supports enterprise-class features, such as multi-replica, hot and cold data separation, and multi-tenancy.

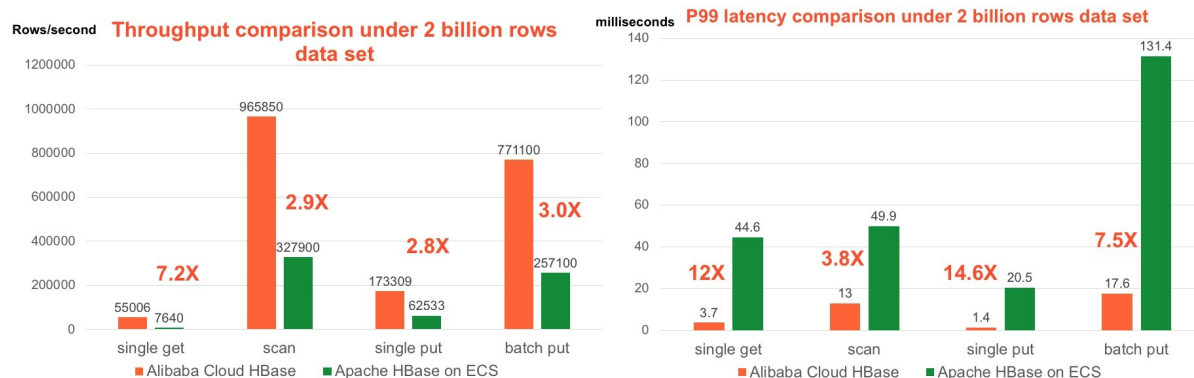
ApsaraDB for HBase Enhanced Edition is a hybrid real-time data storage service. It offers seamless scalability, high throughput, high availability, milliseconds of latency, tunable consistency levels, cost-effectiveness, and various types of indexes. It is suitable for businesses that have high requirements on scale, throughput, performance, and availability. For example, it can be used for big data services (unlimited scalability and high throughput), online services (low latency and high availability), and multi-functional queries.

In addition to the standard HBase API, ApsaraDB for HBase Enhanced Edition will soon support multiple consistency levels, multi-module API (not released yet), and other new features. For more information, see [Benefits](#) and [Performance white paper](#).

Note: ApsaraDB for HBase Enhanced Edition currently does not support Phoenix SQL.

## 1.2. Benefits

Compared with the standard edition, ApsaraDB for HBase Performance-enhanced Edition has the following improvements:



Note :

1. P99 Latency means 99% of requests have a response time below this value
2. The values in the figure are for reference only

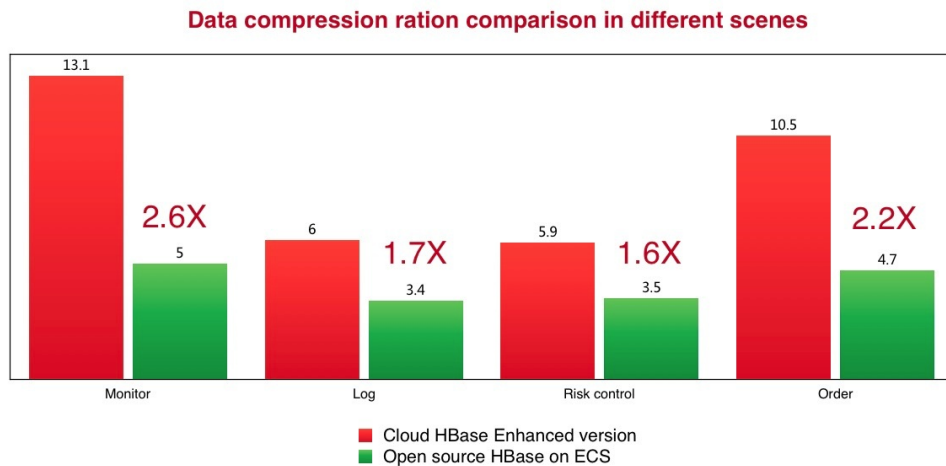
ApsaraDB for HBase Performance-enhanced Edition is deeply optimized for RPCs, memory management, caching, and logging. It uses core technologies such as high-performance data schema, coroutines, merging and submission, and traceable coding to greatly improve the performance of reads and writes. The throughput of ApsaraDB for HBase Performance-enhanced Edition is over seven times higher when compared with HBase Community Edition using the same amount of resources. It can help you reduce 90% of the noise in HBase Community Edition. For more information, see [Performance white paper](#).

## High availability

ApsaraDB for HBase Performance-enhanced Edition has improved the mean time to repair (MTTR). The fault recovery is **over 10 times faster than HBase Community Edition**. Based on the "Logging is Storing" and "if Partition then Availability or Consistency, Else Latency or Consistency (PACELC)" theories, a multi-replica architecture is adopted to guarantee the consistency of data at different levels. This allows applications to choose between consistency, availability, latency, and programmability. It provides more disaster recovery solutions for your applications to ensure high availability, such as active zone-redundancy, active geo-redundancy, and three data center consistency. Currently, only single-zone deployment is available. More deployment solutions will be available soon.

## High-level compression

Developed with a deeply optimized compression algorithm: Zstandard. It supports automatic compression level tuning and compression dictionaries provided based on sampling data. The highest compression ratio can reach 13, which is **over 50% higher** when compared with SNAPPY.



## Transparent hot and cold data separation

In most scenarios, hot data is closer to users than cold data. ApsaraDB for HBase Performance-enhanced Edition supports automatic hot and cold data separation to help you accelerate hot data consumption without the need to modify your applications. You can store cold data in cost-effective medium such as Object Storage Service (OSS).

## Multi-tenancy

Built-in data security and resource isolation features for multi-tenancy. It supports username-password authentication, ACLs, quotas, and resource groups, allowing you to build an all-in-one enterprise-class HBase platform, and improve development efficiency and optimize resource utilization. For more information, see [Cluster management](#).



## 2. Quick start

### 2.1. Connect to a cluster

ApsaraDB for HBase Performance-enhanced Edition allows you to connect to the database service by Java, C++, Python, or Go APIs. To connect to a cluster, you need the information such as the endpoint, username, and password, which can be obtained in the console.

You can connect to ApsaraDB for HBase Performance-enhanced Edition over the Virtual Private Cloud (VPC) network and the public network. To connect the database over these two networks, you must use different endpoints, different methods (Java, C++, Python, or Go APIs), and different ports. You can retrieve the endpoints and ports on the Database Connection page in the console.

ApsaraDB for HBase Performance-enhanced Edition cluster provides two types of endpoints. The endpoints include the same address but different ports.

#### The endpoint used by Java API

Different from ApsaraDB for HBase Standard Edition that provides the Zookeeper, ApsaraDB for HBase Performance-enhanced Edition uses a domain name as the unified endpoint. The endpoint is mapped on a group of high availability servers, which can establish the connection between HBase clients and the RegionServer for high-performance access. For more information, see [Install the SDK for Java](#) and [Use the Java API to access ApsaraDB for HBase](#).

#### The endpoint used by APIs for multiple languages (C++/Python/Go)

ApsaraDB for HBase Performance-enhanced Edition allows you connect to the database by using Thrift that supports multiple languages, such as C++, Python, and Go. For more information, see [APIs for multiple languages](#).

#### Connect to a cluster over a public network

These connection methods allow you to connect to the cluster over a public network. Click Apply for Public Endpoint and the public endpoint will appear in the address bar. You must replace the address with the public endpoint in the application. To stop the connection over a public network, click Release Public Endpoint.

Note 1: The connection over a public network is only used for development, debugging, or testing, but not for production. If you connect to a cluster over a public network, Alibaba Cloud cannot promise to follow the Service Level Agreement. Service Level Agreement (SLA). Due to the bandwidth limit and network latency of the public network, the service performance may be downgraded. Note 2: Do not use a public network domain name in a VPC network. Otherwise, you will connect to the cluster over the public network. In this case, service performance is downgraded.

#### Whitelist

To ensure the security, you must **add the endpoint that is used to connect to the database to the whitelist**, whether you connect to the cluster over the public network or the VPC network. For more information, see [Configure a whitelist](#).

#### Username and password

ApsaraDB for HBase Performance-enhanced Edition provides the security features such as the built-in user authentication and Access Control List (ACL) features. For more information, see [Users and ACL management](#). After you create an instance, a super account is created, which has all the permissions of the cluster. By default, both the username and the password are set to root. You can use this account to connect to the cluster. You can also create an account, delete an account, or modify the password of the root account on the [Access Control](#) page. If you do not need to provide the username and password when you connect to the cluster, disable the ACL feature on the [Access Control](#) page.

## 2.2. Install the SDK for Java

ApsaraDB for HBase Performance-enhanced Edition supports HBase-1.x and HBase-2.x clients. This topic describes how to install the SDK for Java.

You can replace the HBase dependencies with the HBase clients released by Alibaba Cloud. Alibaba Cloud provides HBase client version 1.x and version 2.x. Choose a version based on your HBase dependencies. We recommend that you use the HBase client version 2.x.

### ApsaraDB for HBase version 2.x

```
<dependency>
  <groupId>com.aliyun.hbase</groupId>
  <artifactId>alihbase-client</artifactId>
  <version>2.0.6</version>
</dependency>
```

### ApsaraDB for HBase version 1.x

```
<dependency>
  <groupId>com.aliyun.hbase</groupId>
  <artifactId>alihbase-client</artifactId>
  <version>1.1.1</version>
</dependency>
```

## Plug-ins used to connect to ApsaraDB for HBase Performance-enhanced Edition

We recommend that you use the HBase clients provided by Alibaba Cloud to connect to ApsaraDB for HBase Performance-enhanced Edition. If you are unable to replace the existing HBase dependency, you can use a plug-in provided by Alibaba Cloud. For example, the HBase client already exists in the third-party database that you use and cannot be changed. There are two versions of the alihbase-connector plug-ins. The alihbase-connector version 1.x corresponds to the ApsaraDB for HBase client version 1.x. The alihbase-connector version 2.x corresponds to the ApsaraDB for HBase client version 2.x.

### The plug-in for the ApsaraDB for HBase client version 1.x

```
<dependency>
  <groupId>com.alibaba.hbase</groupId>
  <artifactId>alihbase-connector</artifactId>
  <version>1.0.11</version>
</dependency>
```

## The plug-in for the ApsaraDB for HBase client version 2.x

```
<dependency>
  <groupId>com.alibaba.hbase</groupId>
  <artifactId>alihbase-connector</artifactId>
  <version>2.0.11</version>
</dependency>
```

If your application does not support Maven dependencies, you can download the following alihbase-connector JAR files to the libs directory.

- Click [here](#) to download the alihbase-connector JAR file of version 1.x.
- Click [here](#) to download the alihbase-connector JAR file of version 2.x.

Afterward, you can use the Java API to connect to ApsaraDB for HBase. For more information, see [Use the Java API to access ApsaraDB for HBase](#).

## 2.3. Use the Java API to access ApsaraDB for HBase

You can configure the client parameters to access ApsaraDB for HBase Performance-enhanced Edition in the following ways.

1. Install the SDK for Java. For more information, see [Install the SDK for Java](#).
2. Retrieve the endpoint of the cluster. For more information, see [Connect to a cluster](#).

### Specify client parameters

#### Method 1: Specify parameters by modifying the configuration file.

Add the following configurations to hbase-site.xml:

```
<configuration>
  <!--
    The public endpoint or Virtual Private Cloud (VPC) internal endpoint used to connect to the cluster. You can
    retrieve the endpoint on the Database Connection page in the console.
  -->
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
  </property>
  <!--
    By default, both the username and password values are set to root. You can change the username and the
    password as needed.
  -->
  <property>
    <name>hbase.client.username</name>
    <value>root</value>
  </property>
  <property>
    <name>hbase.client.password</name>
    <value>root</value>
  </property>
  <!--
    If you use the ApsaraDB for HBase client, you do not need to specify the connection.impl parameter. If you
    use alihbase-connector, you must specify this parameter.
  -->
  <!--property>
    <name>hbase.client.connection.impl</name>
    <value>org.apache.hadoop.hbase.client.AliHBaseUEClusterConnection</value>
  </property-->
</configuration>
```

## Method 2: Specify parameters by running a program.

Create a Configuration object and specify the related parameters by running the following program.

```
// Create a Configuration object.
Configuration conf = HBaseConfiguration.create();
// The public endpoint or VPC internal endpoint used to connect to the cluster. You can retrieve the endpoint on the Database Connection page in the console.
conf.set("hbase.zookeeper.quorum", "Id-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020");
// By default, both the username and password values are set to root. You can change the username and the password as needed.
conf.set("hbase.client.username", "root")
conf.set("hbase.client.password", "root")
// If you use the Alibaba Cloud HBase client, you do not need to specify the connection.impl parameter. If you use alihbase-connector, you must specify this parameter.
//conf.set("hbase.client.connection.impl", AliHBaseUEClusterConnection.class.getName());
```

Note: If you add the alihbase-connector earlier than version 1.0.9 or 2.0.9 as a dependency, the configurations are different. For more information, see [Use the Java API to access ApsaraDB for HBase \(former version\)](#).

## Create a connection

Use the Configuration object `conf` to create a Connection object. For more information about how to create the `conf`, see the sample code in the previous step.

```
// Create a connection to ApsaraDB for HBase. You only need to perform this task once while the program is running. This thread is secure and can be shared with all other threads.
// Close the Connection object after the program stops. Otherwise, more connections will be created, resulting in a waste of resources.
// You can also use try finally to avoid creating new connections.
Connection connection = ConnectionFactory.createConnection(conf);
```

## Use the Java API

After you create the connection, you can use the Java API to connect to the ApsaraDB for HBase Performance-enhanced Edition cluster. The following sample code shows how to call the API operations.

## DDL statements

```
try (Admin admin = connection.getAdmin()){
    // Create a table.
    HTableDescriptor htd = new HTableDescriptor(TableName.valueOf("tablename"));
    htd.addFamily(new HColumnDescriptor(Bytes.toBytes("family")));
    // Create a table that has only one partition.
    // We recommend that you pre-split a table based on the data.
    admin.createTable(htd);
    // Disable a table.
    admin.disableTable(TableName.valueOf("tablename"));
    // Truncate a table.
    admin.truncateTable(TableName.valueOf("tablename"), true);
    // Delete a table.
    admin.deleteTable(TableName.valueOf("tablename"));
}
```

## Data Manipulation Language (DML) statements

// A Table is a non-threaded secure object. A thread must retrieve a Table object from Connection before it can manage the table.

```
try (Table table = connection.getTable(TableName.valueOf("tablename"))) {
    // Insert data.
    Put put = new Put(Bytes.toBytes("row"));
    put.addColumn(Bytes.toBytes("family"), Bytes.toBytes("qualifier"), Bytes.toBytes("value"));
    table.put(put);
    // Read a single row.
    Get get = new Get(Bytes.toBytes("row"));
    Result res = table.get(get);
    // Delete a row.
    Delete delete = new Delete(Bytes.toBytes("row"));
    table.delete(delete);
    // Scan the range of the data.
    Scan scan = new Scan(Bytes.toBytes("startRow"), Bytes.toBytes("endRow"));
    ResultScanner scanner = table.getScanner(scan);
    for (Result result : scanner) {
        // Handle the query result.
        // ...
    }
    scanner.close();
}
```

## 2.4. Use HBase Shell to access ApsaraDB for HBase

1. Download the HBase TAR file. You can use one of the following methods to retrieve a TAR file:

- Method 1: Download the TAR file. This file includes all required enhanced dependencies. Click [here](#) to download and decompress the file. **We recommend that you use this TAR file to retrieve all features of ApsaraDB for HBase Performance-enhanced Edition.** Download and use the latest version of the TAR file `alibase-2.0.9-bin.tar.gz`.
- Method 2: Download the TAR file from the HBase [official website](#) and decompress the TAR file to the `lib` directory. Then copy the JAR file of `alibase-connector` to the same directory. For more information, see [Install the SDK for Java](#).

2. Retrieve the Virtual Private Cloud (VPC) endpoint or the public endpoint for HBase Shell. For more information, see [Connect to a cluster](#). You can refer to 'The endpoint used by Java API' section of the topic.

### Configuration

Add the following configuration to the `hbase-site.xml` file in the `conf /` directory of the decompressed TAR file:

```
<configuration>
  <!--
    The public endpoint or VPC internal endpoint used to connect the cluster. You can retrieve the endpoint on the Database Connection page in the console.
  -->
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
  </property>
  <!--
    The default values for both the username and password parameters are root. You can change the username and password as needed.
  -->
  <property>
    <name>hbase.client.username</name>
    <value>root</value>
  </property>
  <property>
    <name>hbase.client.password</name>
    <value>root</value>
  </property>
</configuration>
```

## Use HBase Shell

Run the following command in the `bin/` directory.

```
./hbase shell
```

Now, you can run the native HBase Shell commands to connect to ApsaraDB for HBase Performance-enhanced Edition. By default, the logs are stored in the `logs` directory of the decompressed TAR file. For more information about how to use HBase Shell, see [HBase Shell](#). If you use HBase Shell to connect to ApsaraDB for HBase Performance-enhanced Edition, you can manage tables and data by using some simple Data Definition Language (DDL) and Data Manipulation Language (DML) statements. The cluster management commands such as `BALANCE` and `MOVE` are disabled. For more information, see [Limits](#).

## 2.5. Non-java APIs

ApsaraDB for HBase Performance-enhanced Edition allows you to connect the database by using Thrift that supports multiple languages. ApsaraDB for HBase Performance-enhanced Edition supports all the languages that are supported by Thrift. The ApsaraDB for HBase Performance-enhanced Edition server supports Thrift version 0.12.0. Although Thrift is backward-compatible, we recommend that you download the version 0.12.0. Click [here](#) to download this version. Some languages provide certain methods to manage dependencies. You can install Thrift based on relevant syntax. For example, you can run the `pip install thrift` command if you use Python and run the `import {"github.com/apache/thrift/lib/go/thrift"}` command if you use Go.

ApsaraDB for HBase Performance-enhanced Edition uses Thrift 2 as interface definition language (IDL) file. You must download the Thrift 2 to generate the API for the specified language. Compared with Thrift 1, Thrift 2 provides clearer definitions for API operations. You can perform the operations in a similar way as you call Java API operations. The earlier versions of Thrift 2 are not widely used because they do not support some DDL operations such as creating and deleting tables. Now, the comprehensive Thrift 2 APIs definitions of have been provided by Alibaba Cloud and released to the community. For more information, see [HBASE-21649](#). Thrift 2 provides more comprehensive and easy-to-use features than Thrift 1.

### Prerequisites

1. Click [here](#) to download the Thrift installation package.
2. Download [The Thrift 2 definition file of ApsaraDB for HBase](#).
3. Retrieve the public endpoint or VPC internal endpoint used to connect to the cluster. For more information, see [Connect to a cluster](#). You can refer to 'The endpoint used by APIs for multiple languages' section of the topic.

### Access

The following sections describe how to get started with Thrift. For more information about how to use Thrift, see [Apache Thrift Tutorial](#).

#### 1. Generate the API definition file for the specified language.

Download the API definition file and use the following syntax to generate the corresponding API definition files.



```
thrift --gen <language> Hbase.thrift
```

Example:

```
thrift --gen php Hbase.thrift  
thrift --gen cpp Hbase.thrift  
thrift --gen py Hbase.thrift
```

## 2. Initialize a client to access ApsaraDB for HBase Performance-enhanced Edition.

The Thrift server of ApsaraDB for HBase Performance-enhanced Edition uses **HTTP in the transport layer**. Therefore, the `THttpClient` of Thrift is required when you initialize a client. The client initialization method varies based on the languages. When the Access Control List (ACL) is enabled, you must specify the username and password headers in the `THttpClient` for authentication. The username and password are not required if ACL is disabled. Thrift allows you to call a function for a certain language to customize a header in `THttpClient`. The following example shows how to initialize the client and set the connect string, the username, and password based on Python. For more information, see the Demos for multiple languages section in this topic.

```
# -*- coding: utf-8 -*-
# You can run the pip install thrift command to retrieve the following modules.
from thrift.protocol import TBinaryProtocol
from thrift.transport import THttpClient
# The following module is generated by the thrift --gen py hbase.thrift command.
from hbase import THBaseService
from hbase.ttypes import TColumnValue, TColumn, TTableName, TTableDescriptor, TColumnFamilyDescriptor, TNamespaceDescriptor, TGet, TPut, TScan
# Endpoint
url = "http://host:9190"
transport = THttpClient.THttpClient(url)
headers = {}
# Username
headers["ACCESSKEYID"]="root";
# Password
headers["ACCESSSIGNATURE"]="root"
transport.setCustomHeaders(headers)
protocol = TBinaryProtocol.TBinaryProtocolAccelerated(transport)
client = THBaseService.Client(protocol)
transport.open()
# The operations, and finally closes the connection.
transport.close()
```

## Demos for multiple languages

All demo code has been uploaded to GitHub. The code includes the Thrift definition files and dependencies that are supported by certain languages. You can download the code for a certain language from GitHub.

### Python

<https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/python>

### Go

<https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/go>

### C++

<https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/cpp>

### Node.js

<https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/nodejs>

### PHP

<https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/php>

## More languages

For more information, see [Apache Thrift Tutorial](#).

## 2.6. Limits

ApsaraDB for HBase Performance-enhanced Edition allows you to use the native HBase API to connect to the database service. For more information, see [Use the Java API to access ApsaraDB for HBase](#). This topic describes the limits on the connection to ApsaraDB for HBase Performance-enhanced Edition.

- The API operations for cluster management such as assign region and stopRegionServer are not supported. You can log on to the Cluster Management System to manage clusters. **You can use HBase Shell or call the Java API to run the FLUSH and COMPACTION commands.**
- Coprocessor is not supported and Alibaba Cloud plans to support it in the future. If you have any questions, contact the [ApsaraDB for HBase Q&A DingTalk group](#) for troubleshooting or submit a ticket.
- Limited Hadoop Distributed File System (HDFS) permissions are provided for users who want to import or export a large amount of data. For more information, see [Data import and migration](#). If you want to use Spark analysis, you can use spark-connector, call the HBase API in Spark, or use the TableInputFormat operation of MapReduce (MR) to access ApsaraDB for HBase. For more information, see [Use Spark to access ApsaraDB for HBase](#). **However, ApsaraDB for HBase Performance-enhanced Edition allows you to use the HDFS for bulkload operations.**
- User-defined Filters are not supported. To define a Filter, you need to encapsulate the code of Filter class into a JAR file and upload it to an HBase folder or HDFS. Then you must restart HBase or dynamically loads the JAR file to make the Filter take effect. This method is not supported in ApsaraDB for HBase Performance-enhanced Edition.
- Phoenix SQL is not supported in ApsaraDB for HBase Performance-enhanced Edition.

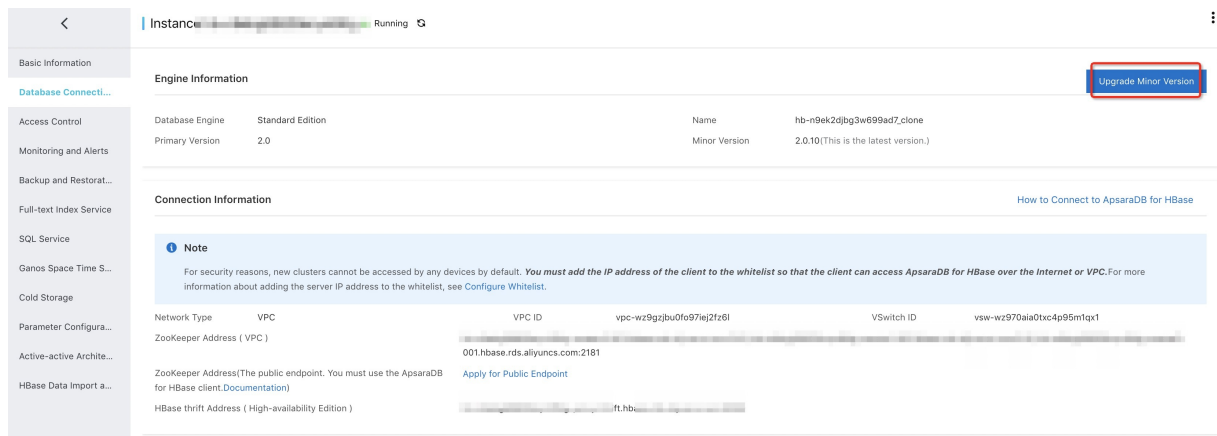
If you have any questions about the limits, submit a ticket or consult the ApsaraDB for HBase Q&A DingTalk group.

## 2.7. Upgrade minor version

The ApsaraDB for HBase team regularly releases minor versions for hot fixes and service improvements. Alibaba Cloud guarantees the compatibility of these minor versions. If critical bugs are found in any of the versions, the team will send you a notification email. We recommend that you manually upgrade minor versions during off-peak hours to ensure the continuity of your business. Alibaba Cloud will not automatically upgrade your clusters.

### Upgrade procedure

Click **Minor Version Upgrade** in the console during off-peak hours.



The HBase service is not interrupted during the upgrade. The system performs a rolling upgrade on the cluster. To minimize the impact on your workloads, all regions will be moved to other servers before the RegionServer restarts. There may be service interruptions during the upgrade process, but all HBase clusters will be still available. You can safely perform minor version upgrades.

The time it takes to upgrade a cluster depends on the size of the cluster and the number of regions. Typically, it takes less than 10 minutes to upgrade a cluster that contains two core nodes and 100 regions. For large clusters that contain thousands of regions, it may take more than an hour to upgrade the cluster.

If you have any questions during the upgrade, submit a ticket or contact the **ApsaraDB for HBase Q&A** DingTalk group for troubleshooting.

## Release notes

### 2.1.8

Cold storage and Hot and cold data separation are supported.

### 2.1.9

Full-text index service is supported and some issues are fixed.

### 2.1.10

High-performance native secondary indexes and Hot and cold data separation are supported.

### 2.1.12

High-performance native secondary indexes are optimized and issues that may be encountered during the bulkload process are fixed.

## 3.Data channel

### 3.1. Use DataWorks or DataX to import data

ApsaraDB for HBase provides Big Datahub Service (BDS) for scenarios such as data migration and real-time data synchronization between various HBase versions. BDS also allows you to synchronize the real-time data from Relational Database Service (RDS) and LogHub to ApsaraDB for HBase. For more information, see BDS introduction. BDS does not support importing data from heterogeneous data sources such as **MaxCompute (formerly ODPS)**. To import data from these data sources, you must use DataX. DataX is an offline data synchronization tool that is widely used within Alibaba Group. DataX synchronizes data between various heterogeneous data sources such as MySQL, Oracle, SQL Server, Postgre, HDFS, Hive, ADS, HBase, TableStore (OTS), MaxCompute (ODPS), and DRDS.

#### Use DataX for data synchronization

You can use one of the following methods to configure DataX synchronization tasks: 1. Use the Data Integration service provided by Alibaba Cloud DataWorks to configure synchronization tasks in DataX. 2. Use the open source edition of DataX to configure synchronization tasks.

#### Method 1: Use DataWorks to specify the parameters of DataX.

##### Create a workspace

For more information, see [Create a workspace](#).

##### Create a resource group

| Type                     | Configuration guide                     | Feature  | Remarks  |
|--------------------------|---|--|--|
| Exclusive resource group | <a href="#">Exclusive resource mode</a> | DataWorks automatically subscribes to and manages exclusive resources. This guarantees service performance and availability. | Exclusive resources cannot be shared among regions. For example, the exclusive resources in the China (Shanghai) region can only be used by the workspace in the China (Shanghai) region. The resources cannot be bound to the Virtual Private Cloud (VPC) network in other regions. You can use exclusive resources to connect only to ApsaraDB for HBase clusters attached to the same Virtual Switch (VSwitch). |

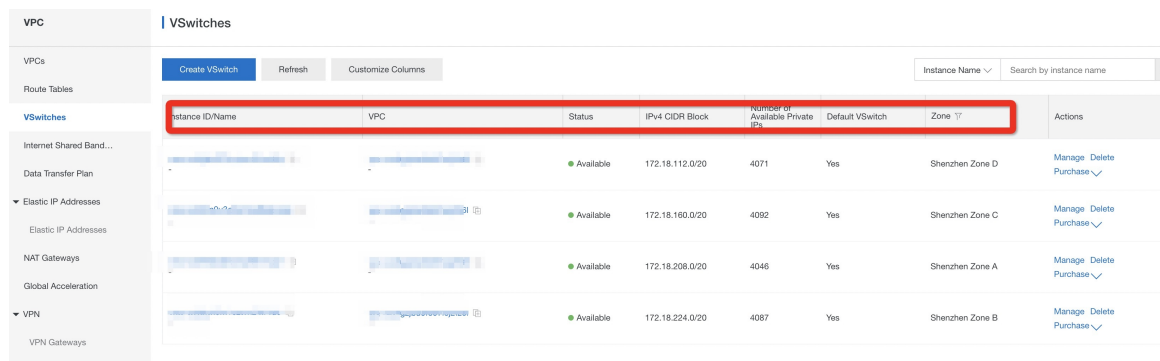
| Type                   | Configuration guide   | Feature   | Remarks  |
|------------------------|-----------------------|---|--|
| Custom resource group  | Custom resource group | <b>Only DataWorks Enterprise Edition and later support custom resource groups.</b> Elastic Compute Service (ECS) instances that belong to a custom resource group can be purchased based on your needs. You can deploy ECS instances in the VPC network to access HBase over the internal network. Otherwise, you can only access ApsaraDB for HBase over a public network. | You have all permissions on instances that belong to the custom resource group. If you want to log on or manage the instances, you must install, manage, maintain, and upgrade DataX as needed. For more information, see Custom resource group. |
| Default resource group | N/A                   | You can use instances that belong to the default resource group to access ApsaraDB for HBase in the VPC network <b>over a public network</b> instead of the internal network.   | If you access ApsaraDB for HBase over a public network, additional costs are occurred for DataWorks. For more information, see <a href="#">Pay-as-you-go</a> .   |

We recommend that you use the resources in the **exclusive resource group** or the **custom resource group** to connect to ApsaraDB for HBase for data synchronization. For more information, see [Custom resource group](#).

## Configure network parameters

### Configure the network for the exclusive resource group

1. Bind the exclusive resource group to the VPC network of ApsaraDB for HBase. For more information, see [Bind a VPC network](#).
2. Check the CIDR block of VPC and VSwitch exclusive resource in the [VPC console](#). The following figure shows the sample CIDR block 192.168.0.0/24. If you do not know the exact IP address of the instance in the exclusive resource group, add the CIDR block to the ApsaraDB for HBase whitelist before you connect to ApsaraDB for HBase.



| Instance ID  | Name         | VPC          | Status    | IPv4 CIDR Block | Number of Available Private IPs | Default VSwitch | Zone            | Actions       |
|--------------|--------------|--------------|-----------|-----------------|---------------------------------|-----------------|-----------------|---------------|
| vsw-12345678 | vsw-12345678 | vpc-12345678 | Available | 172.18.112.0/20 | 4071                            | Yes             | Shenzhen Zone D | Manage Delete |
| vsw-12345679 | vsw-12345679 | vpc-12345678 | Available | 172.18.160.0/20 | 4092                            | Yes             | Shenzhen Zone C | Manage Delete |
| vsw-12345680 | vsw-12345680 | vpc-12345678 | Available | 172.18.208.0/20 | 4046                            | Yes             | Shenzhen Zone A | Manage Delete |
| vsw-12345681 | vsw-12345681 | vpc-12345678 | Available | 172.18.224.0/20 | 4087                            | Yes             | Shenzhen Zone B | Manage Delete |

3. Add the CIDR block to the ApsaraDB for HBase whitelist. For more information, see [Configure a whitelist](#).

## Configure the network for the custom resource group

If you know the exact IP address of each ECS instance in the custom resource group, add all the IP addresses to the ApsaraDB for HBase whitelist. For more information, see [Configure a whitelist](#).

## Configure the network for the default resource group

Retrieve the CIDR block of the instance in the default resource group. For more information, see [Add whitelist](#). Add the CIDR block of the region to the ApsaraDB for HBase whitelist. For more information, see [Configure a whitelist](#).

## Create a synchronization task and bind a resource group

1. Create a synchronization task. For more information, see [Configure a data synchronization node by using the codeless UI](#).
2. Modify the plug-in configurations. Use the hbase11xwriter plug-in for writes and use the hbase11xreader plug-in for reads.

For more information, see the help documentation. You must specify the endpoint parameter instead of the Zookeeper.quorum parameter when you configure the hbaseConfig parameter in ApsaraDB for HBase Performance-enhanced Edition. The following sample code shows how to configure the connection.

```
"hbaseConfig": {  
  "hbase.client.connection.impl": "com.alibaba.hbase.client.AliHBaseUEConnection",  
  "hbase.client.endpoint": "host:30020",  
  "hbase.client.username": "root",  
  "hbase.client.password": "root"  
}
```

### Notes:

- The hbase.client.connection.impl parameter is a fixed configuration. You do not need to change it.
- The parameter hbase.client.endpoint specifies the endpoint used for the Java API. For more information, see [Connect to a cluster](#).
- The hbase.client.username and hbase.client.password parameters specify the user-defined username

and password. Make sure that the account has the read and write permissions on ApsaraDB for HBase Performance-enhanced Edition tables. The default username and password are root. This account has the read and write permissions on all tables. For more information about users and ACLs, see [Connect to a cluster](#).

3. Choose the resource group that you have created as the task resource.

## Method 1: Use the open-source DataX to specify the parameters.

### Download the DataX installation package

Click [here](#) to download and decompress the TAR file of DataX. This TAR file includes the JAR file that is required to access ApsaraDB for HBase Performance-enhanced Edition.

If you have installed DataX or downloaded the latest DataX version from [GitHub](#), you must add the required JAR file. Follow these steps:

Download only the latest JAR file of alihbase-connector version 1.x from the [Plug-ins used to connect to ApsaraDB for HBase Performance-enhanced Edition](#) section in [Install the SDK for Java](#). You do not need to download the entire compressed file. Save the JAR file in the `datax/plugin/writer/hbase11xwriter/libs` directory. If you want to use DataX to read data in ApsaraDB for HBase Performance-enhanced Edition, save the JAR file in `datax/plugin/reader/hbase11xreader/libs` directory.

### Modify the configuration file

In DataX, the `hbase11xreader` plug-in is used to read the data in ApsaraDB for HBase Performance-enhanced Edition. For more information, see [Documentation](#). The `hbase11xwriter` plug-in is used to write data in ApsaraDB for HBase Performance-enhanced Edition. For more information, see [Documentation](#). The configurations of reads and writes in ApsaraDB for HBase Performance-enhanced Edition are the same as the official configuration, except for `hbaseConfig`. You must specify the endpoint parameter instead of the `Zookeeper.quorum` parameter when you configure the `hbaseConfig` in ApsaraDB for HBase Performance-enhanced Edition. The following sample code shows how to configure the connection.

```
...
"hbaseConfig": {
    "hbase.client.connection.impl": "com.alibaba.hbase.client.AliHBaseUEConnection",
    "hbase.client.endpoint": "host:30020",
    "hbase.client.username": "root",
    "hbase.client.password": "root"
}
...
```



You can set this parameter to `com.alibaba.hbase.client.AliHBaseUEConnection` to use the `Connection` object of ApsaraDB for HBase Performance-enhanced Edition. The parameter `hbase.client.endpoint` specifies the endpoint used for the Java API. For more information, see [Connect to a cluster](#). The `hbase.client.username` and `hbase.client.password` parameters specify the user-defined username and password. Make sure that the account has the read and write permissions on ApsaraDB for HBase Performance-enhanced Edition tables. The default username and password are `root`. This account has the read and write permissions on all tables. For more information about users and ACLs, see [Connect to a cluster](#).

## Launch DataX to migrate data

For more information, see the official documentation [DataX](#).

### Note:

Before you migrate the data, you must add IP addresses of ECS instances in the whitelist. For more information, see [Connect to a cluster](#). If ECS instances and ApsaraDB for HBase Performance-enhanced Edition instances are not in the same VPC, you must use the public endpoint.

## 3.2. Use Spark to access HBase

ApsaraDB for HBase Performance-enhanced Edition allows you to use Spark to connect to the database. You can use the ApsaraDB for HBase client or add the `alihbase-connector` as the dependency. For more information, see [Install the SDK for Java](#).

### Retrieve an endpoint

For more information, see [Connect to a cluster](#). If you want to connect to the service over a public network, use the public endpoint in the API operation. By default, the port is 30020.

### Retrieve the username and password

For more information, see [Connect to a cluster](#). The default values for both the username and password parameters are `root`. If you disable the Access Control List (ACL) feature on the cluster management page, the username and password are not required.

### Specify connection parameters

#### Method 1: Specify parameters by modifying the configuration file.

Add the following configurations to `hbase-site.xml`:

```

<configuration>
  <!--
    The public endpoint or Virtual Private Cloud (VPC) internal endpoint used to connect to the cluster. You can obtain this endpoint on the Database Connection page in the console.
  -->
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
  </property>
  <!--
    The default values for both the username and password parameters are root. You can change the username and password based on your needs.
  -->
  <property>
    <name>hbase.client.username</name>
    <value>root</value>
  </property>
  <property>
    <name>hbase.client.password</name>
    <value>root</value>
  </property>
  <!--
    If you use the ApsaraDB for HBase client, you do not need to specify the connection.impl parameter. If your code depends on alihbase-connector, you must specify this parameter.
  -->
  <!--property>
    <name>hbase.client.connection.impl</name>
    <value>org.apache.hadoop.hbase.client.AliHBaseUEClusterConnection</value>
  </property-->
</configuration>

```

## Method 2: Specify parameters by running a program.

Specify the related parameters of a Configuration object by running the following program.

```
// Create a Configuration object.
Configuration conf = HBaseConfiguration.create();
// The public endpoint or VPC internal endpoint used to connect the cluster. You can obtain this endpoint on
the Database Connection page in the console.
conf.set("hbase.zookeeper.quorum", "ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020");
// The default values for both the username and password parameters are root. You can change the userna
me and the password based on your needs.
conf.set("hbase.client.username", "root")
conf.set("hbase.client.password", "root")
// If you use the ApsaraDB for HBase client, you do not need to specify the connection.impl parameter. If you
r code depends on alihbase-connector, you must specify this parameter.
//conf.set("hbase.client.connection.impl", AliHBaseUEClusterConnection.class.getName());
```

## Spark sample code

```
test(" test the spark sql count result") {
  //1. Configure the connection parameters
  var conf = HBaseConfiguration.create
  conf.set("hbase.zookeeper.quorum", "ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020")
  conf.set("hbase.client.username", "test_user")
  conf.set("hbase.client.password", "password")
  //2. Create a table.
  val hbaseTableName = "testTable"
  val cf = "f"
  val column1 = cf + ":a"
  val column2 = cf + ":b"
  var rowCount: Int = -1
  var namespace = "spark_test"
  val admin = ConnectionFactory.createConnection(conf).getAdmin()
  val tableName = TableName.valueOf(namespace, hbaseTableName)
  val htd = new HTableDescriptor(tableName)
  htd.addFamily(new HColumnDescriptor(cf))
  admin.createTable(htd)
  //3. Insert data into the table.
  val rng = new Random()
  val k: Array[Byte] = new Array[Byte](3)
  val famAndQf = KeyValue.parseColumn(Bytes.toBytes(column))
  val puts = new util.ArrayList[Put]()
  var i = 0
  for (k1 <- (1 to 1000)) {
```

```

for (b1 <- ('a' to 'z')) {
  for (b2 <- ('a' to 'z')) {
    for (b3 <- ('a' to 'z')) {
      if(i < 10) {
        k(0) = b1.toByte
        k(1) = b2.toByte
        k(2) = b3.toByte
        val put = new Put(k)
        put.addColumn(famAndQf(0), famAndQf(1), ("value_" + b1 + b2 + b3).getBytes())
        puts.add(put)
        i = i + 1
      }
    }
  }
}

val conn = ConnectionFactory.createConnection(conf)
val table = conn.getTable(tableName)
table.put(puts)
//4. Create a Spark table.
val sparkTableName = "spark_hbase"
val createCmd = s"""CREATE TABLE ${sparkTableName} USING org.apache.hadoop.hbase.spark
    | OPTIONS ('catalog'=
    | '{"table":{"namespace":"${hbaseTableName}", "name":"${hbaseTableName}"}, "rowkey": "rowkey",
    | "columns":{
    | "col0":{"cf":"rowkey", "col":"rowkey", "type":"string"},
    | "col1":{"cf":"cf1", "col":"a", "type":"string"},
    | "col2":{"cf":"cf1", "col":"b", "type":"String"}}}'
    | )"""
println(" createCmd: \n" + createCmd + " rows : " + rowsCount)
sparkSession.sql(createCmd)
//5. Execute the SQL statement to query the number of records in the table.
val result = sparkSession.sql("select count(*) from " + sparkTableName)
val sparkCounts = result.collect().apply(0).getLong(0)
println(" sparkCounts : " + sparkCounts)

```

### 3.3. Use Hive to access HBase

ApsaraDB for HBase Performance-enhanced Edition allows you to connect to the database service by using Hive. However, Hive does not use the standard operations such as GET or PUT to call ApsaraDB for HBase, but calls the internal classes in ApsaraDB for HBase. Therefore, you must replace the existing JAR files in the `hive/lib` directory, rather than adding the alihbase-connector JAR file in the `hive/lib` directory.

```
[hadoop@hadoop16 lib]$ ll |grep hbase
-rw-rw-r-- 1 hadoop hadoop 26687536 Oct 19 15:42 hbase-annotations-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 546195 Oct 19 15:42 hbase-annotations-1.0.2-tests.jar
-rw-rw-r-- 1 hadoop hadoop 20781 May 23 2017 hbase-checkstyle-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 8870 May 23 2017 hbase-client-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 9362 May 23 2017 hbase-common-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 1118767 May 23 2017 hbase-common-1.0.2-tests.jar
-rw-rw-r-- 1 hadoop hadoop 512484 May 23 2017 hbase-examples-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 191960 May 23 2017 hbase-hadoop2-compat-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 122950 May 23 2017 hbase-hadoop-compat-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 86957 May 23 2017 hbase-it-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 35461 May 23 2017 hbase-it-1.0.2-tests.jar
-rw-rw-r-- 1 hadoop hadoop 12585 May 23 2017 hbase-prefix-tree-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 422209 May 23 2017 hbase-protocol-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 102077 May 23 2017 hbase-resource-bundle-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 3690744 May 23 2017 hbase-rest-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 56001 May 23 2017 hbase-server-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 384728 May 23 2017 hbase-server-1.0.2-tests.jar
-rw-rw-r-- 1 hadoop hadoop 3631080 May 23 2017 hbase-shell-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 4712641 May 23 2017 hbase-testing-util-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 12547 May 23 2017 hbase-thrift-1.0.2.jar
-rw-rw-r-- 1 hadoop hadoop 10973 May 23 2017 hive-hbase-handler-1.2.1.jar
-rw-rw-r-- 1 hadoop hadoop 2477029 May 23 2015
-rw-rw-r-- 1 hadoop hadoop 115935 Jun 19 2015
```

1. Delete JAR files whose names start with hbase in the `hive/lib` directory. The following figure shows the JAR files to be deleted, which are highlighted in red. We recommend that you do not delete `hive-hbase-handler-{version}.jar`. This JAR file includes the logic code used to connect to ApsaraDB for HBase in Hive. 2. Click [here](#) to download all JAR files whose names start with alihbase- and put the JAR files in the `hive/lib` directory. If there are other dependencies, it depends on the actual situation. You do not need to replace the JAR files if they already exist. 3. If you have specified the dependency by setting the parameter `--auxpath` in `hive/.hiverc` or by setting this parameter when you start Hive, replace the loaded JAR file with a new JAR file starting with alihbase.

## Prerequisites

### Retrieve an endpoint

For more information, see [Connect to a cluster](#). If you want to connect to the service over a public network, use the public endpoint in the API operation.

### Retrieve the username and password

For more information, see [Connect to a cluster](#). By default, both the username and the password are root. If you disable the Access Control List (ACL) feature on the cluster management page, the username and password are not required.

### Add the IP address of the server where the Hive is deployed to the ApsaraDB for HBase whitelist.

All IP addresses of Hive servers used to connect to ApsaraDB for HBase must be added to the whitelist of the ApsaraDB for HBase cluster. Otherwise, Hive clients cannot connect to ApsaraDB for HBase. For more information, see [Configure the whitelist](#).

## Configure connection parameters in Hive

There are two methods to configure the parameters for connecting to ApsaraDB for HBase in Hive. You can specify Hive connection parameters in the `hive-site.xml` file. Add the following configurations to this file:

```
<configuration>
  <!--
    The public endpoint or Virtual Private Cloud (VPC) internal endpoint used to connect the cluster. You can retrieve the endpoint on the Database Connection page in the console.
  -->
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
  </property>
  <!--
    By default, both the username and the password values are set to root. You can change the username and the password.
  -->
  <property>
    <name>hbase.client.username</name>
    <value>root</value>
  </property>
  <property>
    <name>hbase.client.password</name>
    <value>root</value>
  </property>
</configuration>
```

You can also specify the parameters by running the following commands on the Hive client:

```
set hbase.zookeeper.quorum=ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020
set hbase.client.username=root
set hbase.client.password=root
```

After you specify the parameters, you can create Hive external tables for ApsaraDB for HBase.

## How to use Hive

If the ApsaraDB for HBase table that you want to manage does not exist, you can run the command for creating a table in Hive. A Hive table and ApsaraDB for HBase table are created and automatically associated with each other.

- Launch the Hive CLI.

```
[root@emr-header-2 hive-conf]# hive
Logging initialized using configuration in file:/etc/emr/hive-conf-2.3.3-1.0.1/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
hive>
```

- Create an ApsaraDB for HBase table.

```
CREATE TABLE hive_hbase_table(key int, value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
TBLPROPERTIES ("hbase.table.name" = "hive_hbase_table", "hbase.mapred.output.outputtable" = "hive_hbase_table");
```

- Insert data into the ApsaraDB for HBase table by Hive CLI.

```
insert into hive_hbase_table values(212,'bab');
```

```
hive> insert into hive_hbase_table values(212,'bab');
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20181014173030_a0e99198-9aa5-4d29-b011-dc7b36365a20
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1536221485395_0084, Tracking URL = http://emr-header-1.cluster-74778:20888/proxy/application_1536221485395_0084/
Kill Command = /usr/lib/hadoop-current/bin/hadoop job -kill job_1536221485395_0084
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2018-10-14 17:30:40,833 Stage-3 map = 0%, reduce = 0%
2018-10-14 17:30:47,252 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 3.66 sec
MapReduce Total cumulative CPU time: 3 seconds 660 msec
Ended Job = job_1536221485395_0084
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1 Cumulative CPU: 3.66 sec HDFS Read: 11867 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 660 msec
OK
Time taken: 17.385 seconds
```

- View the ApsaraDB for HBase table. You can see the table has been created and the data has been inserted.

```
[root@i1zbp16ku9i9clejitib6dz ~]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/t-apsara-hbase-1.4.6.3/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/t-emr-hadoop-2.7.2.2/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.4.6.3, r89ac288a5add370c07548ec3ce25f6e1f3210d23, Fri Jul 6 14:13:46 CST 2018

hbase(main):001:0> list
TABLE
A_TABLE
BAKE
BASE_TABLE
B_IDX
B_IDX1
B_IDX2
DEFAULT_TEST_IDX
MY_TABLE
PROD_METRICS
SYSTEM_MUTEX
SYSTEM: CATALOG
SYSTEM: FUNCTION
SYSTEM: SEQUENCE
SYSTEM: STATS
SYSTEM: TEST
hive_hbase_table
tbodytag
tv
18 row(s) in 0.2110 seconds
```

```
hbase(main):004:0* scan 'hive_hbase_table'
ROW                                COLUMN+CELL
 212                                column=cf1:val, timestamp=1539509446271, value=bab
1 row(s) in 0.0950 seconds
```

- Write data to the ApsaraDB for HBase table and check the data in Hive.

```
hbase(main):005:0> put 'hive_hbase_table','132','cf1:val','acb'
0 row(s) in 0.0430 seconds
```

- Query the data in Hive:

```
hive> select * from hive_hbase_table;
OK
132    acb
212    bab
Time taken: 0.273 seconds
```

- After you delete the table in Hive, the associated table in ApsaraDB for HBase is also deleted.

```
hive>
>
>
> drop table hive_hbase_table;
OK
Time taken: 6.307 seconds
```

- Query the table in ApsaraDB for HBase. An error message is returned indicating that the table does not exist.

```
hbase(main):008:0* scan 'hive_hbase_table'
ROW                                COLUMN+CELL

ERROR: Unknown table hive_hbase_table!
```

- If the ApsaraDB for HBase table already exists, you can associate it with an external table in Hive. If you delete the external table, the ApsaraDB for HBase table will not be deleted.
- Create an ApsaraDB for HBase table and use PUT to insert data into the table.

```
hbase(main):020:0* create 'hbase_table','f'
0 row(s) in 1.3010 seconds

=> Hbase::Table - hbase_table
hbase(main):021:0> put 'hbase_table','1122','f:col1','hello'
0 row(s) in 0.0190 seconds

hbase(main):022:0> put 'hbase_table','1122','f:col2','hbase'
0 row(s) in 0.0110 seconds
```

- Create an external table that is associated with the ApsaraDB for HBase table in Hive and query data in Hive.

```
hive> create external table hbase_table(key int, col1 string, col2 string)
> STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
> WITH SERDEPROPERTIES ("hbase.columns.mapping" = "f:col1,f:col2")
> TBLPROPERTIES("hbase.table.name" = "hbase_table", "hbase.mapred.output.outputtable" = "hbase_table");
OK
Time taken: 0.129 seconds
hive> select * from hbase_table;
OK
1122    hello    hbase
Time taken: 0.181 seconds, Fetched: 1 row(s)
hive>
```

- Delete the external table in Hive. The associated ApsaraDB for HBase table still exists.



```
hive> drop table hbase_table;
OK
Time taken: 0.102 seconds
hive> 
```

```
hbase(main):023:0> scan 'hbase_table'
ROW                                COLUMN+CELL
1122                                column=f:col1, timestamp=1539510170256, value=hello
1122                                column=f:col2, timestamp=1539510181752, value=hbase
1 row(s) in 0.0160 seconds
```

For more information, visit <https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration>.

Note: If you associate Hive with snapshots of ApsaraDB for HBase Performance-enhanced Edition, you are unable to read HFiles by Hive. You can create a table in Hive and associate this Hive table with the ApsaraDB for HBase table or create an external table in Hive to associate with the existing ApsaraDB for HBase table. No matter which method you use, you can query the data in ApsaraDB for HBase by Hive.

## 3.4. Use Flink to access HBase

ApsaraDB for HBase Performance-enhanced Edition allows you to use Apache Flink to connect to the database service. You can use HBase tables as dimension tables or result tables in Flink. For more information, see [Create an HBase result table](#) and [Create an HBase dimension table](#).

When you use Data Definition Language (DDL) statements to create an HBase table as the Flink dimension or result table, you must use the endpoint of ApsaraDB for HBase Performance-enhanced Edition. For more information, see [Connect to a cluster](#). When you use Flink to connect to ApsaraDB for HBase Performance-enhanced Edition, you can use `the endpoint used by Java API`. Both the default values of the username and password parameters are root. If you use a new account, make sure that the account has read and write permissions on the tables associated with Flink. For more information, see [Manage users and ACLs](#). The following examples show how to use DDL statements to create HBase tables:

```
CREATE TABLE hbase (
  `key` varchar,
  `name` varchar,
  PRIMARY KEY (`key`), -- The rowkey of the hbase table.
  PERIOD FOR SYSTEM_TIME // Specify that this is a dimension table.
) with (
  TYPE = 'cloudhbase',
  endpoint = 'host:port', -- The endpoint used by Java API to connect to ApsaraDB for HBase Performance-enhanced Edition.
  columnFamily = 'xxxxxx',
  userName = 'root', -- The username.
  password = 'root', -- The password.
  tableName = 'xxxxxx'
);
```

### Create an HBase result table

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  endpoint = 'host:port', -- The endpoint used by Java API to connect to ApsaraDB for HBase Performance-enhanced Edition.
  userName = 'root', -- The username.
  password = 'root', -- The password.
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
)
```

## Network connection

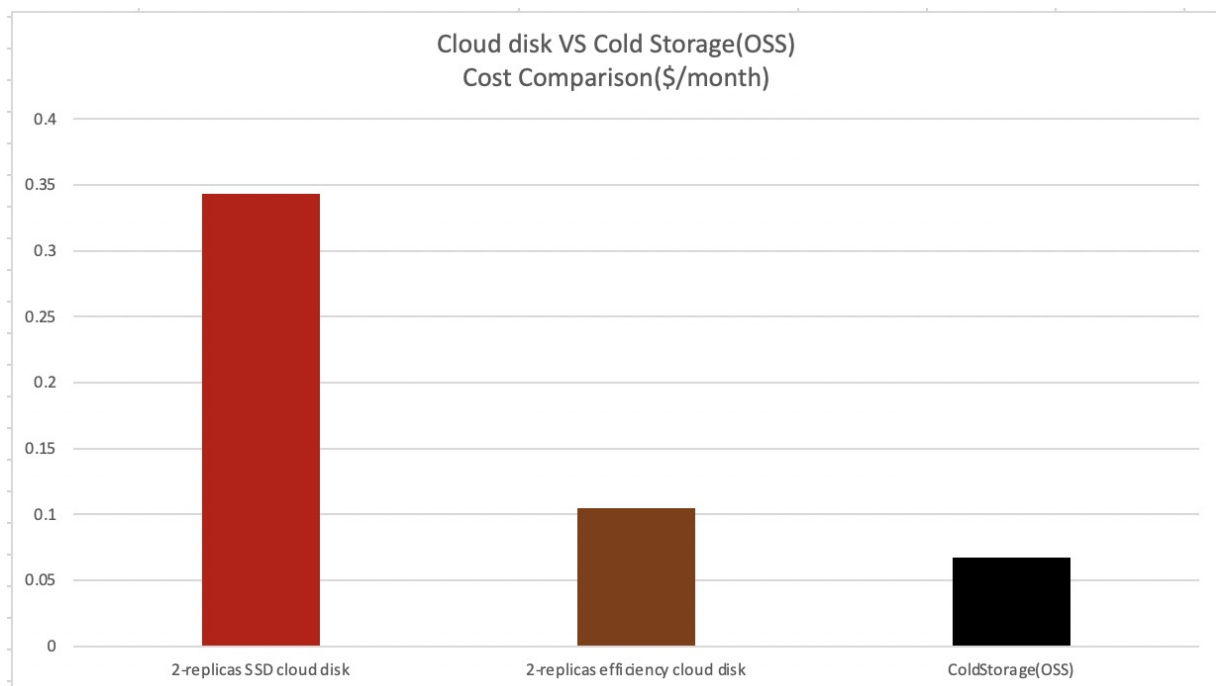
If you use a Flink cluster in shared mode, you must connect to ApsaraDB for HBase over a Virtual Private Cloud (VPC) network. For more information, see [Authorize Flink to access the VPC network](#). If you use a Flink cluster in exclusive mode, make sure that the Flink cluster and the HBase cluster are in the same VPC network. Otherwise, you can only use the public endpoint to connect to ApsaraDB for HBase. For more information, see [Connect to a cluster](#). Before you connect to ApsaraDB for HBase, you must add the IP address of the Flink cluster to the HBase whitelist. For more information, see [Configure a whitelist](#).

## 4. Enterprise Features

### 4.1. Cold storage

In big data scenarios, tiered storage is frequently used to store cold and hot data separately. ApsaraDB for HBase provides a new cold storage medium to store cold data. It provides equivalent write performance at one third the storage cost of ultra disks. You can query cold data in the cold storage at any time. Cold storage is applicable to various cold data scenarios such as data archiving and infrequently accessed data consumption. Cold storage is easy to use and can reduce storage costs. When you purchase an ApsaraDB for HBase instance, you can choose cold storage as an additional storage medium. Then, you can run table creation statements to store cold data in cold storage.

ApsaraDB for HBase Performance-enhanced Edition uses the cold storage technology to separate and store hot and cold data on the same table. You can use cold and hot data separation feature to store hot data in hot storage for efficient reads and writes, and store cold data in cold storage to minimize storage costs. For more information, see [Cold and hot data separation](#).



#### Activate cold storage

You can buy cold storage independently and use it as an additional storage.

When you create a new ApsaraDB for HBase Performance-enhanced Edition instance, select cold storage and specify the cold storage capacity.

#### Configuration Upgrade

If you did not activate cold storage when you create an instance, click Cold Storage in the left-side navigation pane in the cluster console and click Enable Now to activate cold storage.

**Note:** Only versions of ApsaraDB for HBase Performance-enhanced Edition later than 2.1.8 support cold storage. When you activate cold storage, the version of your instance is automatically updated to the latest version.

## Use cold storage

**Note:** To use cold storage, you must upgrade ApsaraDB for HBase Performance-enhanced Edition to a version later than 2.1.8. The version of the client dependency alihbase-connector must be later than 1.0.7 or 2.0.7. The version of HBase Shell must be later than alihbase-2.0.7-bin.tar.gz.

ApsaraDB for HBase Performance-enhanced Edition allows you to set storage properties based on column families. You can set the Storage parameter of a column family or all column families of a table to COLD. Then all data of this column family or all column families in the table is stored in cold storage and does not occupy the Hadoop Distributed File System (HDFS) space of the cluster. You can specify the property when you create a table or modify the property of the column family after you create a table.

You can use Java API or HBase Shell to create a table and modify the table properties. If you use the Java API, you must install the SDK for Java and configure the parameters first. For more information, see [Use the Java API to access ApsaraDB for HBase](#). If you use HBase Shell, you must download and configure HBase Shell first. For more information, see [Use HBase Shell to access ApsaraDB for HBase](#).

## Create a table that uses cold storage

### HBase Shell

```
hbase(main):001:0> create 'coldTable', {NAME => 'f', STORAGE_POLICY => 'COLD'}
```

### Java API

```
Admin admin = connection.getAdmin();
HTableDescriptor descriptor = new HTableDescriptor(TableName.valueOf("coldTable"));
HColumnDescriptor cf = new HColumnDescriptor("f");
cf.setValue("STORAGE_POLICY", AliHBaseConstants.STORAGETYPE_COLD);
descriptor.addFamily(cf);
admin.createTable(descriptor);
```

## Modify the table property to use cold storage

If you have created a table, you can modify the property of a column family in the table to use cold storage. If the column family contains data, the data is archived to cold storage after a major compaction.

### HBase Shell

```
hbase(main):011:0> alter 'coldTable', {NAME=>'f', STORAGE_POLICY => 'COLD'}
```

### Java API

```
Admin admin = connection.getAdmin();
TableName tableName = TableName.valueOf("coldTable");
HTableDescriptor descriptor = admin.getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
// Set the storage type of the table to cold storage.
cf.setValue("STORAGE_POLICY", AliHBaseConstants.STORAGETYPE_COLD);
admin.modifyTable(tableName, descriptor);
```

Set the property of the table to hot storage. If you want to change the storage type from cold storage to hot storage, you can modify the table property. If the column family contains data, the data is archived to hot storage after a major compaction.

### HBase Shell

```
java hbase(main):014:0> alter 'coldTable', {NAME=>'f', STORAGE_POLICY => 'DEFAULT'}
```

### Java API

```
Admin admin = connection.getAdmin();
TableName tableName = TableName.valueOf("coldTable");
HTableDescriptor descriptor = admin.getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
// Set the storage type of the table to the default type. By default, the storage type is hot storage.
cf.setValue("STORAGE_POLICY", AliHBaseConstants.STORAGETYPE_DEFAULT);
admin.modifyTable(tableName, descriptor);
```

## View the cold storage status

You can view the cold storage status and expand the capacity of the cold storage on the Cold Storage page in the console.

**Cold Storage Description**  
 HBase allows you to store cold data in cold storage. The cost of cold storage is 1/3 of the cost of an ultra disk but offers the same write performance. Cold storage also ensures that data can be read at any time. It is suitable for scenarios such as data archive and storing historical data that is not frequently accessed. For more information about the storage, see [Documentation](#).

**Cold Storage Information**

Cold Storage Scaling

Total Capacity 810 GB
 Used of Capacity 0.00 %

To check the cold storage usage and hot storage usage for a table, go to the [Cluster Management System](#) and click the User tables tab.

User tables
 

group name:  namespace:  table type:  enabled:

table name:

|                          | NameSpace | User Table | Description           | TableSize | ColdStorageSize | HotStorageSize | FileNum | RegionNum | enabled | Group   |
|--------------------------|-----------|------------|-----------------------|-----------|-----------------|----------------|---------|-----------|---------|---------|
| <input type="checkbox"/> |           |            | {NAME = 'HBASE', ...} | OMB       | 0               | 0              | 1       | 256       | true    | default |

## Performance testing

## Environment requirements

Master: ECS. c5.xlarge, 4-core 8 GB memory, 20 GB ultra disk. 4 RegionServer: ECS. c5.xlarge, 4-core 8 GB memory, 20 GB ultra disk. 4 Test Machine: ECS. c5.xlarge, 4-core 8 G memory.

## Write performance

| Table type | avg rt  | p99 rt  |
|------------|---------|---------|
| Hot tables | 1736 us | 4811 us |

| Table type  | avg rt  | p99 rt  |
|-------------|---------|---------|
| Cold tables | 1748 us | 5243 us |

Note: Each data record includes 10 columns and has 100 bytes data stored in each column. This means that there is 1 KB data in each row. The system writes data in 16 parallel threads.

## Random GET performance

| Table type  | avg rt   | p99 rt   |
|-------------|----------|----------|
| Hot tables  | 1704 us  | 5923 us  |
| Cold tables | 14738 us | 31519 us |

Note: If you disable BlockCache, the system reads the data from the disk every time. Each data record includes 10 columns and has 100 bytes data stored in each column. This means that there is 1 KB data in each row. The system reads 1 KB data for each request by using eight parallel threads.

## Scan performance within a specified range

| Table type  | avg rt   | p99 rt    |
|-------------|----------|-----------|
| Hot tables  | 6222 us  | 20975 us  |
| Cold tables | 51134 us | 115967 us |

Note: Disable the BlockCache of the table. Each data record includes 10 columns and has 100 bytes data stored in each column. This means that there is 1 KB data in each row. The system reads 1 KB data for each request by using eight parallel threads. Set the Caching parameter to 30.

## Notes

1. The read Input/Output Operations Per Second (IOPS) of cold storage is low (up to 25 times/s per node), so cold storage is applicable to infrequent queries.
2. The write throughput of cold storage equals to the throughput of the ultra disks that are used for hot storage.
3. Cold storage cannot process a large number of concurrent read requests. An error may occur if cold storage is used to process a large number of concurrent read requests.
4. If you have purchased an extremely large cold storage space, you can adjust the read IOPS as needed. You can submit a ticket to request technical support.
5. We recommend that you store no more than 30 TB of cold data in each core node. To expand the storage capacity of each core node, you can submit a ticket.

## 4.2. Cold and hot data separation

In a big data scenario, business data, such as order or monitoring data, may grow over time. As your business develops, the rarely used data is archived. Enterprises may want to use a cost-effective storage to store this type of data to reduce costs. ApsaraDB for HBase Performance-enhanced Edition supports cold and hot data separation to help enterprises save data storage costs. ApsaraDB for HBase Performance-enhanced Edition uses a new medium (cold storage) to store cold data, which allows you to save two thirds of the costs compared with using ultra disks.

ApsaraDB for HBase Performance-enhanced Edition can automatically separate the cold and hot data stored in a table based on user settings. The cold data is automatically archived in the cold storage. When you query a table that has cold and hot data stored separately, you only need to specify query hints or time ranges for the system to determine whether to scan the cold or hot data. This is an automated process which is transparent to users.

## How it works

ApsaraDB for HBase Performance-enhanced Edition determines whether the data written into a table is cold data based on the timestamp of the data and the time boundary (in milliseconds) set by users. New data is stored in the hot storage. It ages with time and is finally moved to the cold storage. You can change the time boundary for separating cold and hot data as needed. Data can be moved from the cold storage to hot storage or from the hot storage to cold storage.

### Scenario

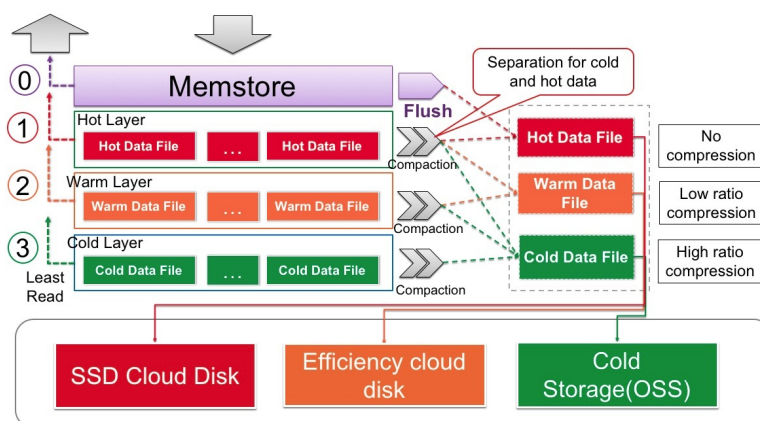
- ✓ Only recently written data is frequently accessed

### Values

- ✓ Reducing the storage cost of cold data
- ✓ Improving access performance of hot data

### Advantages

- ✓ **Easy-using**, enable it through adding the attribute to table
- ✓ **No change required** for application
- ✓ Cold and hot separation line, **flexible adjustment**
- ✓ **Free setting** of storage media, compression algorithms for hot and cold data



## Use cold and hot data separation

Note: To use cold storage, you must upgrade ApsaraDB for HBase Performance-enhanced Edition to a version later than 2.1.8. The version of the client dependency AliHBase-Connector must be later than 1.0.7/2.0.7. The version of HBase Shell must be later than alihbase-2.0.7-bin.tar.gz.

Before you use the Java API to access ApsaraDB for HBase, follow the steps described in [Use the Java API to access ApsaraDB for HBase](#) to install the Java SDK and configure parameters.

Before you use HBase Shell to access ApsaraDB for HBase, follow the steps described in [Use HBase Shell to access ApsaraDB for HBase](#) to download and configure HBase Shell.

## Activate cold storage

Follow the steps described in [Cold storage](#) to activate cold storage for your ApsaraDB for HBase cluster.

## Set a time boundary for a table



You can modify the COLD\_BOUNDARY parameter to change the time boundary for separating cold and hot data. The time boundary is measured in seconds. For example, if COLD\_BOUNDARY => 86400, newly inserted data ages out after 86,400 seconds (1 day) and is then archived as cold data.

You do not need to set the property of the column family to COLD for cold and hot data separation. If you have already set it to COLD, follow the instructions in [Cold storage](#) to remove the property.

#### HBase Shell

```
// Create a table that has cold and hot data stored separately.  
hbase(main):002:0> create 'chsTable', {NAME=>'f', COLD_BOUNDARY=>'86400'}  
  
// Disable cold and hot data separation.  
hbase(main):004:0> alter 'chsTable', {NAME=>'f', COLD_BOUNDARY=>""}  
  
// Enable cold and hot data separation for a table or change the time boundary.  
hbase(main):005:0> alter 'chsTable', {NAME=>'f', COLD_BOUNDARY=>'86400'}
```

#### Java API

```
// Create a table that has cold and hot data stored separately.
Admin admin = connection.getAdmin();
TableName tableName = TableName.valueOf("chsTable");
HTableDescriptor descriptor = new HTableDescriptor(tableName);
HColumnDescriptor cf = new HColumnDescriptor("f");
// The COLD_BOUNDARY parameter specifies the time boundary in seconds for separating cold and hot data.
// In this example, new data ages out after 86,400 seconds (1 day) and is then archived as cold data.
cf.setValue(AliHBaseConstants.COLD_BOUNDARY, "86400");
descriptor.addFamily(cf);
admin.createTable(descriptor);
// Disable cold and hot data separation.
// Note: You must run a major compaction before you can move the data from the cold storage to hot storage.
HTableDescriptor descriptor = admin
    .getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
// Disable cold and hot data separation.
cf.setValue(AliHBaseConstants.COLD_BOUNDARY, null);
admin.modifyTable(tableName, descriptor);
// Enable cold and hot data separation for a table or change the time boundary.
HTableDescriptor descriptor = admin
    .getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
// The COLD_BOUNDARY parameter specifies the time boundary in seconds for separating cold and hot data.
// In this example, new data ages out after 86,400 seconds (1 day) and is then archived as cold data.
cf.setValue(AliHBaseConstants.COLD_BOUNDARY, "86400");
admin.modifyTable(tableName, descriptor);
```

## Insert data

You can write a table that has cold and hot data stored separately in the same way as writing a regular table. For more information about how to insert data to a table, see [Use the Java API to access ApsaraDB for HBase](#) or [Use the API of a non-Java language to access ApsaraDB for HBase](#). The timestamp of data is the `time` when the data is written into a table. New data is stored in the hot storage (standard disks). When the age of the data exceeds the value specified in the `COLD_BOUNDARY` parameter, the system automatically moves the data to the cold storage during the major compaction process. The process is completely transparent to users.

## Query data

ApsaraDB for HBase Performance-enhanced Edition allows you to use a table to store both cold and hot data. This saves you the effort of dealing with more than one table when you query data. You can set the `HOT_ONLY` hint in a GET or SCAN statement to query only hot data if you can confirm that the age of the target data is less than the value specified in the `COLD_BOUNDARY` parameter. You can also set the `TimeRange` parameter in a GET or SCAN statement to specify the time range of the data to be queried. The system automatically determines whether the target data is hot or cold based on the specified time range. It takes more time to query cold data than querying hot data. The throughput of reading cold data is lower than that of reading hot data. For more information, see [Cold storage](#).

## Examples

### Get

HBase Shell

```
// Query data without the HOT_ONLY hint. The query may hit cold data.
hbase(main):013:0> get 'chsTable', 'row1'

// Query data with the HOT_ONLY hint. The query only hits the hot data. If row1 is stored in the cold storage,
no query result is returned.
hbase(main):015:0> get 'chsTable', 'row1', {HOT_ONLY=>true}

// Query data within a specified time range. The system determines whether the query hits the cold or hot d
ata based on the TIMERANGE and COLD_BOUNDARY settings. The value of the TIMERANGE parameter is meas
ured in milliseconds.
hbase(main):016:0> get 'chsTable', 'row1', {TIMERANGE => [0,1568203111265]}
```

Java API

```
Table table = connection.getTable("chsTable");

// Query data without the HOT_ONLY hint. The query may hit cold data.
Get get = new Get("row1".getBytes());
System.out.println("Get operation: " + cache.get(key));

// Query data with the HOT_ONLY hint. The query only hits the hot data. If row1 is stored in the cold storage,
no query result is returned.
get = new Get("row1".getBytes());
get.setAttribute(AliHBaseConstants.HOT_ONLY, Bytes.toBytes(true));

// Query data within a specified time range. The system determines whether the query hits the cold or hot d
ata based on the TIMERANGE and COLD_BOUNDARY settings. The value of the TIMERANGE parameter is meas
ured in milliseconds.
get = new Get("row1".getBytes());
get.setTimeRange(0, 1568203111265)
```

### Scan

Note: If you do not set the HOT\_ONLY hint or specify a time range for the SCAN statement, both the cold and hot data is queried. The query results are merged and returned to you. This is determined by the principle how the HBase SCAN operation works.

#### HBase Shell

```
// Query data without the HOT_ONLY hint. The query hits both the cold and hot data.
hbase(main):017:0> scan 'chsTable', {STARTROW =>'row1', STOPROW=>'row9'}

// Query data with the HOT_ONLY hint. The query only hits the hot data.
hbase(main):018:0> scan 'chsTable', {STARTROW =>'row1', STOPROW=>'row9', HOT_ONLY=>true}

// Query data within a specified time range. The system determines whether the query hits the cold or hot data based on the TIMERANGE and COLD_BOUNDARY settings. The value of the TIMERANGE parameter is measured in milliseconds.
hbase(main):019:0> scan 'chsTable', {STARTROW =>'row1', STOPROW=>'row9', TIMERANGE => [0, 1568203111265]}
```

#### Java API

```
TableName tableName = TableName.valueOf("chsTable");
Table table = connection.getTable(tableName);

// Query data without the HOT_ONLY hint. The query hits both the cold and hot data.
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
for (Result result : scanner) {
    System.out.println("scan result:" result);
}

// Query data with the HOT_ONLY hint. The query only hits the hot data.
scan = new Scan();
scan.setAttribute(AliHBaseConstants.HOT_ONLY, Bytes.toBytes(true));

// Query data within a specified time range. The system determines whether the query hits the cold or hot data based on the TIMERANGE and COLD_BOUNDARY settings. The value of the TIMERANGE parameter is measured in milliseconds.
scan = new Scan();
scan.setTimeRange(0, 1568203111265);
```

#### Notes:

1. The cold storage is only used to archive data that is rarely accessed. Only a few of queries can hit the cold data. Most of the queries carry the `HOT_ONLY` hint or have a time range specified to hit the hot data only. If your cluster receives a large number of queries hitting the cold data, you need to check whether the time boundary is set appropriately.

2. If you update a field in a row stored in the cold storage, the field is moved to the hot storage after it is updated. When this row is hit by a query that carries the HOT\_ONLY hint or has a time range targeting the hot data, only the updated field in the hot storage is returned. If you want the system to return the entire row, you must delete the HOT\_ONLY hint from the query or make sure that the specified time range covers the time period from when this row was inserted to when this row was last updated. We recommend that you do not update the data stored in the cold storage. If you need to update the cold data frequently, we recommend that you adjust the time boundary to move the data to the hot storage.

## Query the sizes of cold and hot data

You can check the sizes of cold and hot data in a table on the User tables tab of the [cluster management system](#). If the ColdStorageSize field displays 0, this indicates that the cold data is still stored in RAM. You can run the flush command to flush the data to disks, and then run a major compaction to check the size of the cold data.

User tables

group name:  namespace:  table type:  enabled:

table name:

|                          | NameSpace | User Table | Description         | TableSize | ColdStorageSize | HotStorageSize | FileNum | RegionNum | enabled | Group   |
|--------------------------|-----------|------------|---------------------|-----------|-----------------|----------------|---------|-----------|---------|---------|
| <input type="checkbox"/> |           |            | {NAME = '...', ...} | OMB       | 0               | 0              | 1       | 256       | true    | default |

## Advanced features

### Prioritize hot data selection

The system may lookup both the cold and hot data upon SCAN queries such as queries submitted to retrieve all the order or chat records. The query results are paginated based on the timestamps of the data in descending order. In most cases, the hot data is displayed ahead of the cold data. If the SCAN queries do not carry the HOT\_ONLY hint, the system must scan both the cold and hot data, resulting in performance degradation. If you have prioritized hot data selection, cold data is queried only when you want to view more query results. For example, you want the system to return more results by clicking Next on the page. This minimizes the cold data access frequency and reduces the response time.

To prioritize hot data selection, you only need to set the `COLD_HOT_MERGE` property to true in your scan query. This feature allows the system to scan hot data first. If you want to view more query results, the system then starts scanning cold data.

HBase Shell

```
hbase(main):002:0> scan 'chsTable', {COLD_HOT_MERGE=>true}
```

Java API

```
java
scan = new Scan();
scan.setAttribute(AliHBaseConstants.COLD_HOT_MERGE, Bytes.toBytes(true));
scanner = table.getScanner(scan);
```

**Notes:** 1. When hot data prioritization is enabled, if a row stores both cold and hot data is queried, the query results are returned in two batches. This means that you will find two results for the same rowkey in the result set. 2. The system cannot ensure that the rowkey of the returned cold data is greater than that of the hot data because it returns the hot data first. This means that the results returned upon SCAN queries are not sequentially sorted. However, the records in the returned cold or hot data are still sorted in the rowkey order, as shown in the following demo. In some scenarios, you can sort the SCAN results by designing the rowkeys appropriately. For example, you can create a rowkey that consists of user IDs and order creation times. Records can be retrieved based on user IDs and sorted based on creation times.

```
// Assume that the rowkey "coldRow" stores cold data and the rowkey "hotRow" stores hot data.
// In most cases, the rowkey "coldRow" is returned ahead of the rowkey "hotRow" because rows in HBase are sorted in lexicographical order.
hbase(main):001:0> scan 'chsTable'
ROW                                COLUMN CELL
coldRow                            column=f:value, timestamp=1560578400000, value=cold_value
hotRow                             column=f:value, timestamp=1565848800000, value=hot_value
2 row(s)

// When COLD_HOT_MERGE is set to true, the system scans the rowkey "hotRow" first. As a result, the rowkey "hotRow" is returned ahead of the rowkey "coldRow".
hbase(main):002:0> scan 'chsTable', {COLD_HOT_MERGE=>true}
ROW                                COLUMN CELL
hotRow                             column=f:value, timestamp=1565848800000, value=hot_value
coldRow                            column=f:value, timestamp=1560578400000, value=cold_value
2 row(s)
```

**Separate cold and hot data based on the fields in a rowkey.** In addition to the timestamps of KV pairs, ApsaraDB for HBase Performance-enhanced Edition can also separate cold and hot data based on specific fields in a rowkey. For example, it can parse the timestamp field in a rowkey to separate cold and hot data. If separating cold and hot data based on the timestamps of KV pairs cannot meet your requirements, submit a ticket or consult the ApsaraDB for HBase Q&A DingTalk group.

**Considerations**  
For considerations about using cold and hot data separation, see [Cold storage](TODO).

## 4.3. High-performance native secondary indexes

HBase supports rowkey (primary key) indexing, allowing you to sort rows based on the binary order of rowkeys. Based on rowkey indexing, row scans, prefix scans, and range scans can be performed efficiently. However, if you want to query a table based on a column other than rowkeys, you must use filters to narrow down the range of rowkeys. Otherwise, the entire table is scanned, resulting in a waste of resources and increased response time.

Many solutions are provided for users to query HBase data in multiple dimensions. For example, you can create a secondary index table. In this case, you need to manually maintain the secondary indexes. You can also export the data to be queried to an external system, such as Solr or Elasticsearch, and then index the data. Search engines such as Solr and Elasticsearch provide powerful Ad Hoc Query capabilities. ApsaraDB for HBase is integrated with the full-text index service. This service saves you the effort of deploying and maintaining an external search engine.

However, it is recommended that you do not use these power search engines in scenarios where regular query patterns are used to query tables containing only a few columns. ApsaraDB for HBase Performance-enhanced Edition offers a solution that uses native secondary indexes to query HBase data with lower costs. Secondary indexes are built in ApsaraDB for HBase to support high throughput and performance. This solution has been tested on businesses, such as Double 11 Shopping Festivals, in Alibaba for many years. It is suitable for indexing large amounts of data. The following figure compares the read/write performance of data indexed using ApsaraDB for HBase with that using Phoenix.



The following sections introduce two important terms in indexes, and then describe how to use Data Definition Language (DDL) and Data Manipulation Language (DML) statements to manage secondary indexes. Advanced examples are provided to show how to sort rows based on the binary order of rowkeys and how to optimize the performance of queries. The last section lists the constraints and limits about using secondary indexes.

### Terms

Create a primary table and four index tables as follows:

```
create table 'dt' (rowkey varchar, c1 varchar, c2 varchar, c3 varchar, c4 varchar, c5 varchar constraint pk primary key(rowkey));
create index 'idx1' on 'dt' (c1);
create index 'idx2' on 'dt' (c2, c3, c4);
create index 'idx3' on 'dt' (c3) include (c1, c2, c4);
create index 'idx4' on 'dt' (c5) include (ALL);
```

## Index key columns

Index key columns refer to primary key columns in an index table. For example, column c1 in table idx1 and columns c2, c3, and c4 in table idx2 are index key columns. Index key columns and their order determine what kind of query patterns that an index table supports. A single-column index is an index table that contains only one index key column. A multi-column index (also known as a composite index) is an index table that contains more than one index key column. For more information about how to make queries hit a specified index table, see [Query optimization](#).

## Index included columns

If the system cannot find the required index key column in the index table, it scans the primary table. In a distributed search scenario, primary table scanning causes unnecessary RPC calls, which significantly increases the response time. To reduce the query time at the cost of storage space, you can configure an index table to cover the required columns in the primary table to avoid primary table scanning when a query hits the index table. Index included columns refer to columns of the primary table covered by an index table. For example, indexes idx3 and idx4 contain index included columns.

As your business develops, new columns may be added to the primary table. Therefore, ApsaraDB for HBase provides the syntax for you to treat all columns in a primary table as index included columns, as shown in idx4. The index table automatically covers all the columns of the primary table and treats them as index included columns.

## Preparation

- Cluster version: The version of your ApsaraDB for HBase cluster must be 2.1.10 or later. You can click the minor version of your cluster in the console to upgrade the cluster version.
- Client version: The client version must be later than alihbase-client 1.1.9/2.0.4, or later than alihbase-connect or 1.0.9/2.0.9. For more information, see [Install the SDK for Java](#).
- HBase Shell version: The HBase Shell version must be later than alihbase-2.0.9-bin.tar.gz. You can visit [HBase Shell](#) to download the latest version of HBase Shell for ApsaraDB for HBase Performance-enhanced Edition.

## Use DDL statements to manage indexes

You can use DDL statements to manage indexes in [HBase Shell](#) or Java API. This section describes how to run DDL statements in HBase Shell to manage indexes. For more information about how to use the Java API, see annotations about the AliHBaseUEAdmin operation.

The following lists some common DDL statements used to manage indexes:



```
# Create an index
# Create the index idx1 for the primary table dt: create index idx1 on dt ('f1:c2', 'f1:c3');
# Include all the columns of the primary table. You must not use COVERED_ALL_COLUMNS as a column name
because it is used as a keyword in this command.
hbase(main):002:0> create_index 'idx1', 'dt', {INDEXED_COLUMNS => ['f1:c2', 'f1:c3']}, {COVERED_COLUMNS =
> ['COVERED_ALL_COLUMNS']}
# View the index schema.
hbase(main):002:0> describe_index 'dt'
# Disable the index table idx1 for the primary table dt. Consequently, when you update the primary table dt,
idx1 is not updated.
hbase(main):002:0> offline_index 'idx1', 'dt'
# Delete the index table idx1.
hbase(main):002:0> remove_index 'idx1', 'dt'
```

The commands are described in details as follows.

## Create an index

```
hbase(main):002:0>create_index 'idx1', 'dt', {INDEXED_COLUMNS => ['f1:c2', 'f2:c3']}
```

Create the index table idx1 for the primary table dt. The index table contains two index key columns: column c2 in column family f1 and column c3 in column family f2. No other columns are specified as index included columns.

```
hbase(main):002:0>create_index 'idx2', 'dt', {INDEXED_COLUMNS => ['f1:c1']}, {COVERED_COLUMNS => ['f2:c2'
]}
```

Create the index table idx1 for the primary table dt. The index table contains one index key column and one index included column: column c2 in column family f1 and column c3 in column family f2.

```
hbase(main):002:0>create_index 'idx3', 'dt', {INDEXED_COLUMNS => ['f1:c3']}, {COVERED_COLUMNS => ['COVE
RED_ALL_COLUMNS']}
```

The index table idx3 covers all the columns of the primary table dt. Therefore, the system does not need to scan the primary table for a given query that hits idx3. The `COVERED_ALL_COLUMNS` keyword is specified to cover all the columns of the primary table. Therefore, you must not use it as a column name.

After you create the schema (index key columns and index included columns) of an index table, you must also configure storage settings for the index table. Example:

```
hbase(main):002:0>create_index 'idx1', 'dt', {INDEXED_COLUMNS => ['f1:c1', 'f2:c2']}, {DATA_BLOCK_ENCODING => 'DIFF', BLOOMFILTER => 'ROW', COMPRESSION => 'LZO'}
```

## View the index schema

Run the list command to view a primary table and the associated index tables. Example:

```
hbase(main):023:0> list
TABLE
dt
dt.idx1
dt.idx2
yh1
4 row(s)
Took 0.0129 seconds
=> ["dt", "dt.idx1", "dt.idx2", "yh1"]
```

Run the describe\_index command to view the schemas of all index tables associated with a primary table.

```
hbase(main):024:0> describe_index 'dt'
Index [idx2] on [dt] (f1.c3 ASC,f2.c4 ASC)
Index [idx1] on [dt] (f1.c1 ASC,f2.c2 ASC) includes (@@ALL@@)
Total 2 indexes found for table dt
```

## Delete an index

You must run the offline\_index command to disable an index before you can run the remove\_index command to delete the index. Example:

```
# Disable the index table idx1 for the primary table dt. Consequently, when you update the primary table dt,
the index idx1 is not updated.
hbase(main):002:0>offline_index 'idx1', 'dt'
# Delete the index table idx1.
hbase(main):002:0>remove_index 'idx1', 'dt'
```

You cannot run the disable command to disable an index table. If the status of an index table is offline, no query hits the table.

## Create indexes for a table that contains historical data

If you run the create\_index command to create indexes for a table that contains historical data, the command automatically synchronizes the historical data from the primary table to the index tables. This can be time-consuming when the primary table contains a large amount of data. Note: The data synchronization task triggered by the create\_index command runs at the server end. Even if you terminate the hbase.shell process, the task continues running until data synchronization is complete.

Creating indexes asynchronously will soon be supported. With this feature, historical data synchronization is no longer triggered by the create\_index command. You need to manually trigger the task by running a command. You can check the progress of the synchronization task by verifying that the status of the index is active or not.

## Manage indexes (DML)

This section describes how to run DML statements with the Java API to manage indexes. For more information about how to use the HBase Java API and connect to ApsaraDB for HBase, see [Use the Java API to connect to ApsaraDB for HBase](#).

## Insert data

You do not need to manually insert data into your index table. When you insert data to the primary table, ApsaraDB for HBase automatically synchronizes the data to all the associated index tables. ApsaraDB for HBase Performance-enhanced Edition defines the synchronization logic for you to synchronize updates to all the associated index tables when you write the primary table. The insert operation does not return a response to the client until the primary and index tables are finished updating. The logic can be interpreted as follows:

- High consistency: After you complete writing the primary table, the updates can be immediately retrieved.
- Update or time out: When ApsaraDB for HBase updates the primary and index tables, it does not know whether the primary and index tables contain the same data. For both the primary and index tables to reach data consistency at the final state, ApsaraDB for HBase chooses to update all of them or times out the write request.

## Query data

Similar to relational databases, ApsaraDB for HBase Performance-enhanced Edition allows you to query primary tables instead of querying index tables. ApsaraDB for HBase Performance-enhanced Edition automatically selects the optimal index table based on the schemas of the index tables and the query patterns. Use Filter to define query conditions based on columns other than rowkeys. Example:

```
byte[] f = Bytes.toBytes("f");
byte[] c1 = Bytes.toBytes("c1");
byte[] value = Bytes.toBytes("yourExpectedValue");
// This is equivalent to select * from dt where f.c1 == value.
Scan scan = new Scan();
SingleColumnValueFilter filter = new SingleColumnValueFilter(f, c1, EQUAL, value);
filter.setFilterIfMissing(true);
scan.setFilter(filter);
```

Notes:

- When you use conditions such as `LESS` and `GREATER`, you must pay close attention to how the numbers are sorted. For more information, see [Sort signed numbers](#).
- You must set the `setFilterIfMissing` parameter to `true` to use indexes. Otherwise, the system scans the entire primary table for a given query.

By using `FilterList`, you can combine AND and OR conditions in the statement to support more complex queries. Example:

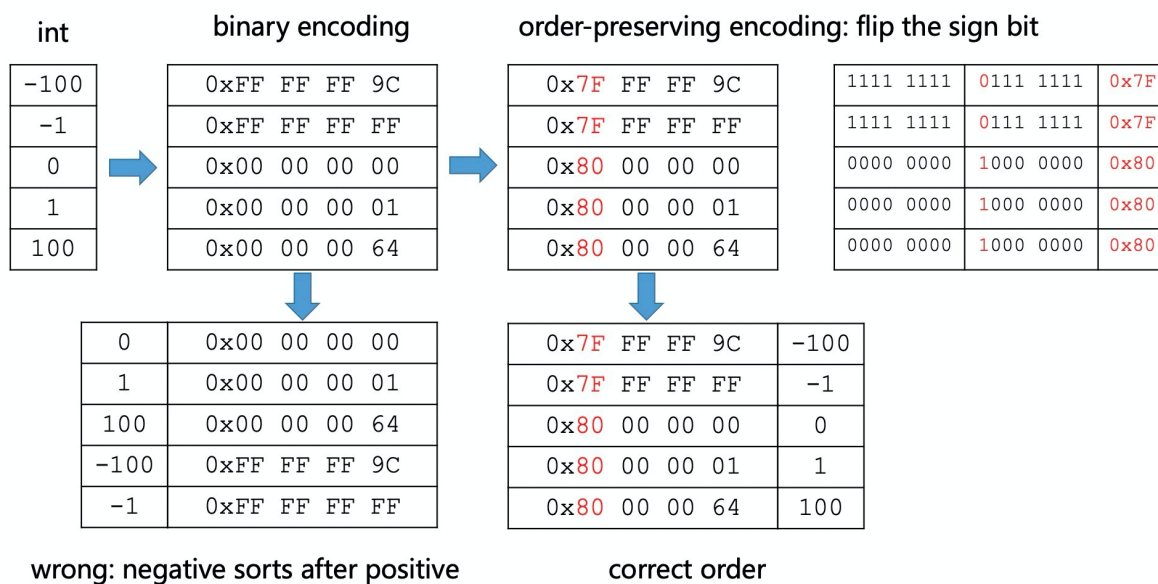
```
// This is equivalent to where f.c1 >= value1 and f.c1 < value2.
FilterList filters = new FilterList(FilterList.Operator.MUST_PASS_ALL);
filters.addFilter(new SingleColumnValueFilter(f, c1, GREATER_OR_EQUAL, value1));
filters.addFilter(new SingleColumnValueFilter(f, c1, LESS, value2));
```

ApsaraDB for HBase Performance-enhanced Edition automatically selects an index table based on filters and index schemas. For more information, see the [Query optimization](#) section.

## Advanced features

### Sort signed numbers

The HBase-native API only supports one data type: `byte[]` (byte arrays). Numbers in byte arrays are sorted in binary order. Therefore, you need to convert all the data to `byte[]`. This involves preserving the order of how the numbers are sorted after you convert them to `byte[]`. The HBase client provides the `Bytes` class for you to switch between `byte[]` and other data types. However, it is only applicable to 0 and positive numbers. It does not support negative numbers. The following figure takes the `INT` type as an example.



In this figure, `Bytes.toBytes(int)` means `binary encoding`. When the numbers contain negative numbers, the order of how the `INT` numbers are sorted cannot be preserved after they are converted to `byte[]`. We can reverse the sign bit of these numbers to resolve this problem. You can use this solution to handle the same problem in `byte`, `short`, `long`, and `float` data. ApsaraDB for HBase Performance-enhanced Edition provides a new class `org.apache.hadoop.hbase.util.OrderedBytes` for you to resolve this problem. You can find this class on any HBase clients that are reliant on `alibase-client` or `alibase-connector`. The following example shows how to use this class:

```
// Convert INT to order-preserved byte[].
int x = 5;
byte[] bytes = OrderedBytes.toBytes(x);
// Convert byte[] to INT.
int y = OrderedBytes.toInt(bytes);
```

For more information about how to use this class, see the annotations of the class.

### Query optimization

This section explains how ApsaraDB for HBase Performance-enhanced Edition selects indexes for a given query. This is equivalent to **prefix match**. It is a Rule Based Optimization (RBO) policy. When the cluster receives a query, it selects the index table with the highest matching rate based on the query conditions combined with AND and the prefixes of the matching index tables. The following examples are provided to explain this rule.

For example, you have created the following primary table and index tables:

```
create table 'dt' (rowkey varchar, c1 varchar, c2 varchar, c3 varchar, c4 varchar, c5 varchar constraint pk primary key(rowkey));
create index 'idx1' on 'dt' (c1);
create index 'idx2' on 'dt' (c2, c3, c4);
create index 'idx3' on 'dt' (c3) include (c1, c2, c4);
create index 'idx4' on 'dt' (c5) include (ALL);
```

The following queries are sent to query the primary table:

```
select rowkey from dt where c1 = 'a';
select rowkey from dt where c2 = 'b' and c4 = 'd';
select * from dt where c2 = 'b' and c3 >= 'c' and c3 < 'f';
select * from dt where c5 = 'c';
```

Now we analyze these queries in sequence:

1) `select rowkey from dt where c1 = 'a'`

This query hits the index table `idx1`.

2) `select rowkey from dt where c2 = 'b' and c4 = 'd'`

This query hits the index table `idx2`. It looks for rows that meet the `c2='b'` condition, and then matches these rows against `c4='d'`. Column C4 is an index key column. However, column C3 is not defined in the WHERE condition. Consequently, the query fails to match the prefix of `idx2`.

3) `select * from dt where c2 = 'b' and c3 >= 'c' and c3 < 'f'`

The query hits the index table `idx2` and completely matches the table. This query uses the `select *` statement, but the index table does not include all the columns from the primary table. Therefore, the system must also look up the primary table. When the system looks up the primary table, it may need to make several RPC calls because the rowkeys may be dispersed in the primary table. If the system needs to scan a large amount of data, the response time increases accordingly.

4) `select * from dt where c5 = 'c'`

This query hits the index table `idx4` and completely matches the table. The index table `idx3` contains all the columns from the primary table. The query uses the `select *` statement. Therefore, the system does not need to scan the primary table.

Based on these conclusions, you need to consider the query patterns and then design your index tables accordingly. For more information about how to use indexes in a better manner, see "Database index design and optimization".

## Constraints and limits

- Different primary tables can be associated with index tables with the same name. For example, the primary table dt can be associated with the index table idx1 and the primary table foo can also be associated with an index table using the same name. However, the names of the index tables associated with the same primary table must be unique.
- You can only create indexes for primary tables that have one version (version = 1).
- If a primary table is assigned a TTL value, its index tables are assigned the same TTL as the primary table. You cannot set a TTL for the index tables.
- An index table cannot contain more than 3 index key columns.
- The total sizes of index key columns and rowkeys in the primary table must not be greater than 30 KB. We recommend that you do not use a column larger than 100 bytes as an index key column.
- A primary table cannot be associated with more than 5 index tables.
- A query can only hit one index table. Index Merge queries are not supported.
- When you create an index table, the data in the primary table is synchronized to the index table. It will be time-consuming if you create an index table for a large data table. Creating indexes asynchronously will soon be supported.

The following features are currently not released:

- Create indexes asynchronously: This feature only creates indexes. It does not synchronize historical data from the primary table to the index tables. You must run another command to create indexes on the historical data.
- Insert data with custom timestamps.

If you have any questions about using indexes, submit a ticket or consult the [ApsaraDB for HBase Q&A](#) DingTalk group.

## 5. Performance White paper

### 5.1. Test environment

- This test compares the performance of open source HBase with that of ApsaraDB for HBase Performance-enhanced Edition in multiple scenarios.
- Use a VPC network. The client and server must be in the same zone.
- All tests are performed in Zone B of the China (Shanghai) region.
- The version of the open source HBase is 1.4.9. You can replace this version with the HBase service EMR-3.20.0 provided by E-MapReduce.
- The version of ApsaraDB for HBase Performance-enhanced Edition is 1.5.1.2.

| Configuration item          | Description                        |
|-----------------------------|------------------------------------|
| Core node configuration     | 16-core and 32 GB (ecs.c5.4xlarge) |
| Number of core nodes        | 3                                  |
| Disk capacity per core node | 3.2 TB                             |
| Disk type of core nodes     | Ultra disk                         |
| Master node configuration   | 4-core and 8 GB                    |

#### Configurations of the ApsaraDB for HBase Performance-enhanced Edition cluster

| Configuration item          | Description       |
|-----------------------------|-------------------|
| Core node configuration     | 16-core and 32 GB |
| Number of core nodes        | 3                 |
| Disk capacity per core node | 3.2 TB            |
| Disk type of core nodes     | Ultra disk        |
| Master node configuration   | 4-core and 8 GB   |

#### Client configurations

| Configuration item          | Description                        |
|-----------------------------|------------------------------------|
| Client node configuration   | 16-core and 32 GB (ecs.c5.4xlarge) |
| The number of client nodes. | 1                                  |

### 5.2. Test tool

AHBench is a benchmark test suite developed by the ApsaraDB for HBase team.

This test suite integrates the features included in the ApsaraDB for HBase Performance-enhanced Edition performance white paper, such as the Yahoo Cloud Serving Benchmark (YCSB) test set, test process control, and results aggregation. AHBench allows you to run benchmark tests without any complex programming required.

## Download AHBench

Click [here](#) to download AHBench. Upload AHBench to the stress testing client and extract the file.

## Runtime environment

The stress testing client must meet the following requirements:

- Linux
- JDK 1.8 or later
- Python 2.7
- We recommend that you use a client with at least 16 exclusive CPU cores.

## Configurations

### (Required) Configure the HBase cluster endpoint

- Configure the endpoint of the HBase cluster in the AHBench/conf/hbase-site.xml file.
- If you are using the ApsaraDB for HBase Enterprise Standard Edition, you need to retrieve the Zookeeper (ZK) address.
- If you are using the ApsaraDB for HBase Performance-enhanced Edition, you need to retrieve the ZK address. For more information, see [Use HBase Shell to access ApsaraDB for HBase](#).

### (Required) Configure runtime environment variables

Use AHBench/conf/ahbench-env.properties to configure the environment variables for the tool.

```
vi AHBench/conf/ahbench-env.properties
# Configure the installation path for Java Development Kit (JDK) If the environment variable Path has been configured in the system, you can skip this step.
# JAVA_HOME=/usr/java/jdk1.8.0/
# Configure the version of the HBase cluster. If you use version 1.x, set the value to 1. If you use version 2.x, set the value to 2.
HBASE_VERSION=2
```

### (Optional) Configure other related parameters

Use AHBench/conf/ahbench-config.properties to configure other parameters such as the compression algorithm, encoding algorithm, number of threads, data volume, and field size. **The default parameters are recommended for most cases.** You can customize the configurations to fit your needs.



Note: Some parameters are only supported in certain HBase editions. For example, ZSTD Compression and INDEX encoding are only supported in ApsaraDB for HBase Performance-enhanced Edition. You can configure the ZSTD and INDEX parameters to enhance the performance.

```
# Configure the compression algorithm of the table.
# Valid values: NONE, LZO, ZSTD, SNAPPY, GZ, LZ4, and ZSTD.
# Note that some systems may not support the specified compression algorithm.
# We recommend that you use the ZSTD compression algorithm for ApsaraDB for HBase Performance-enhanced Edition.
ahbench.table.compression=SNAPPY
# Configure the encoding algorithm of the table.
# Valid values: NONE, DIFF, and INDEX.
# Note that some systems may not support the specified encoding algorithm.
# We recommend that you use the INDEX encoding algorithm for ApsaraDB for HBase Performance-enhanced Edition.
ahbench.table.encoding=DIFF
```

## Start the test

### Fast test set

The test data includes 10 million entries, which occupies at least 20 GB storage. It takes about 40 minutes to finish testing and the actual time may vary based on different HBase editions.

```
cd AHBench
./fast_test
```

### Full test set

The test data includes 2 billion entries, which occupies at least 2 TB storage. It takes about 25 hours to finish testing and the actual time may vary based on different HBase editions.

```
cd AHBench
./full_test
```

If you want to repeat the test and you have imported data in the previous test, you do not need to import data again. This reduces the execution time. It takes about 3.5 hours to finish testing after you skip the data import. The actual time may vary based on different HBase editions.

```
cd AHBench
./full_test --skipload
```

## Test results

After all test cases are complete, a CSV file is generated in the current directory. CSV is short for Comma-Separated Values. You can import the benchmark results to the data analysis software such as Excel and Numbers for further analysis.

The following figure shows the test results in the CSV file.

```
hadoop@nake-ahbench-for-c5 ~/AHBench $ ls -ltr
total 52
drwxr-xr-x 14 hadoop hadoop 4096 Sep  5 10:23 workloads/
-rw-r--r--  1 hadoop hadoop 1421 Sep  5 10:23 README
-rw-r--r--  1 hadoop hadoop  558 Sep  5 10:23 LICENSE
-rwxr-xr-x  1 hadoop hadoop  741 Sep  5 10:23 full_test*
-rwxr-xr-x  1 hadoop hadoop  963 Sep  5 10:23 fast_test*
drwxr-xr-x  5 hadoop hadoop 4096 Sep  5 10:23 tools/
drwxr-xr-x  2 hadoop hadoop 4096 Sep  5 10:31 conf/
drwxrwxr-x  2 hadoop hadoop 4096 Sep  5 10:37 tmp/
drwxr-xr-x  2 hadoop hadoop 4096 Sep  5 10:43 bin/
drwxr-xr-x  2 hadoop hadoop 4096 Sep  5 10:43 suite/
-rw-rw-r--  1 hadoop hadoop  437 Sep  6 11:16 full_throughput.csv
-rw-rw-r--  1 hadoop hadoop  408 Sep  6 11:16 full_spike_latency.csv
drwxrwxr-x  2 hadoop hadoop 4096 Sep  6 11:43 logs/
hadoop@nake-ahbench-for-c5 ~/AHBench $ cat full_throughput.csv
TestName,OperationType,Throughput(rows/s),AverageLatency(us),P95Latency(us),P99Latency(us),P999Latency(us)
full_throughput_single_read,READ,68901.0,2898.706728054881,9255.0,46559.0,108863.0
full_throughput_scan,SCAN,1247157.0,4004.8719123340397,18639.0,47167.0,71103.0
full_throughput_single_write,INSERT,195834.0,1016.75785602,1344.0,3009.0,13295.0
full_throughput_batch_write,BATCH,782735.0,12722.031288233438,20623.0,87743.0,116095.0
hadoop@nake-ahbench-for-c5 ~/AHBench $ cat full_spike_latency.csv
TestName,OperationType,Throughput(rows/s),AverageLatency(us),P95Latency(us),P99Latency(us),P999Latency(us)
full_spike_single_read,READ,4996.0,656.0502980620339,974.0,2335.0,51775.0
full_spike_scan,SCAN,249864.0,1390.0287474438796,2061.0,8035.0,16655.0
full_spike_single_write,INSERT,49971.0,582.7455317893222,664.0,1112.0,5943.0
full_spike_batch_write,BATCH,199871.0,2798.396296975711,2945.0,14951.0,96319.0
```

## Considerations and FAQ

- The test suite uses the `ahbenchtest-read` and `ahbenchtest-write` tables for testing. AHBench may delete and create these tables during the test. If you have permissions, you definitely can delete these tables.
- We recommend that you do not run the test tool in a production environment. The system may stop responding due to the heavy workloads of the test.
- If AHBench returns an error and quits unexpectedly, check the following items:
  - Check whether `JAVA_HOME` is properly configured and the Python runtime environment is installed.
  - Check whether the endpoint of the cluster is correct.
  - Check whether the version of the HBase cluster is correct.
  - Check whether the cluster supports the specified compression algorithm.
  - Check whether the cluster is in the normal status.
  - Check whether the storage of the cluster is sufficient.
- Elastic Compute Service (ECS) is a virtual runtime environment, benchmark results of the same configurations may fluctuate between 5 and 10 percent. This fluctuation is within the acceptable range.

## 5.3. Benchmark methods

The benchmark tests compare the throughput and response latency between HBase Community Edition and ApsaraDB for HBase Performance-enhanced Edition. The throughput benchmark test uses the same number of threads to test the throughput of HBase Community Edition and the throughput of ApsaraDB for HBase Performance-enhanced Edition. The response latency benchmark test uses the same amount of workloads to test the response latency of HBase Community Edition and the response latency of ApsaraDB for HBase Performance-enhanced Edition. The compression ratio benchmark test writes the same amount of data into HBase Community Edition and ApsaraDB for HBase Performance-enhanced Edition to test their compression ratios.

Create a table in the cluster of HBase Community Edition and the cluster of ApsaraDB for HBase Performance-enhanced Edition. The tables used in all test cases use the same schema. Create 200 partitions based on the Yahoo Cloud Serving Benchmark (YCSB) data.

The table created in the cluster of ApsaraDB for HBase Performance-enhanced Edition uses the exclusive INDEX encoding and Zstandard compression algorithms. When you set the encoding algorithm to DIFF, it is automatically updated to the INDEX encoding algorithm. For more information about Zstandard, see [ApsaraDB for HBase compression algorithms](#). The statement for creating the table is as follows:

```
create 'test', {NAME=> 'f', DATA_BLOCK_ENCODING => 'DIFF', COMPRESSION => 'ZSTD'}, {SPLITS => (1..199).map{|i| "user#{(i * ((2**63-1)/199)).to_s.rjust(19, "0")}"}}
```

The table created in the cluster of HBase Community Edition uses the DIFF encoding and SNAPPY compression algorithms, which are recommended by official HBase. The statement for creating the table is as follows:

```
create 'test', {NAME=> 'f', DATA_BLOCK_ENCODING => 'DIFF', COMPRESSION => 'SNAPPY'}, {SPLITS => (1..199).map{|i| "user#{(i * ((2**63-1)/199)).to_s.rjust(19, "0")}"}}
```

## Prepare the test data

Prepare the data to be read from a single row and within a specified range.

The test dataset contains 2 billion rows. Each row contains 20 columns. The size of each column is 20 bytes.

The YCSB configuration file is configured as follows:

```
recordcount=2000000000
operationcount=150000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=false
fieldcount=20
fieldlength=20
readproportion=1.0
updateproportion=0.0
scanproportion=0
insertproportion=0
requestdistribution=uniform
```

Run the following command to launch YCSB:

```
bin/ycsb load hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -s
```

## Test scenarios

### Throughput benchmark tests

The throughput benchmark tests compare the throughput of HBase Community Edition with that of ApsaraDB for HBase Performance-enhanced Edition based on the same number of threads. The tests include four test scenarios. The test scenarios are independent of each other.

#### Read data in a single row

The test dataset contains 2 billion rows. Each row contains 20 columns. The size of each column is 20 bytes. The query range is 10 million rows. After the preceding data is prepared, run a major compaction and wait for the system to complete the major compaction. Run a warm-up test for 20 minutes, and then run a formal test for 20 minutes.

The configuration file of YCSB is as follows:

```
recordcount=10000000  
operationcount=2000000000  
workload=com.yahoo.ycsb.workloads.CoreWorkload  
readallfields=false  
fieldcount=1  
fieldlength=20  
readproportion=1.0  
updateproportion=0.0  
scanproportion=0  
insertproportion=0  
requestdistribution=uniform
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -p maxexecutiontime=1200
```

#### Read data within a specified range

The test dataset contains 2 billion rows. Each row contains 20 columns. The size of each column is 20 bytes. The query range is 10 million rows. 50 rows are read each time. After the preceding data is prepared, run a major compaction and wait for the system to complete the major compaction. Run a warm-up test for 20 minutes, and then run a formal test for 20 minutes.

The configuration file of YCSB is as follows:

```
recordcount=10000000
operationcount=2000000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=false
fieldcount=1
fieldlength=20
readproportion=0.0
updateproportion=0.0
scanproportion=1.0
insertproportion=0
requestdistribution=uniform
maxscanlength=50
hbase.usepagefilter=false
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 100 -p columnfamily=f -p maxexecutiontime=1200
```

## Write data into a single row

Insert one column into the table at a time. The size of each column is 20 bytes. Run the test for 20 minutes.

The configuration file of YCSB is as follows:

```
recordcount=2000000000
operationcount=100000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=false
fieldcount=1
fieldlength=20
readproportion=0.0
updateproportion=0.0
scanproportion=0
insertproportion=1.0
requestdistribution=uniform
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -p maxexecutiontime=1200
```

## Write data into multiple rows

Insert one column into the table at a time. The size of each column is 20 bytes. Write data into 100 rows per batch. Run the test for 20 minutes.

```
recordcount=2000000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
fieldcount=1
fieldlength=20
cyclickey=true
readallfields=false
readproportion=0
updateproportion=0
scanproportion=0
insertproportion=0.0
batchproportion=1.0
batchsize=100
requestdistribution=uniform
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 100 -p columnfamily=f -p maxexecutiontime=1200
```

## Response latency benchmark tests

The response latency benchmark tests compare the response latency of HBase Community Edition with that of ApsaraDB for HBase Performance-enhanced Edition based on the same Operations per Second (OPS).

### Read data in a single row

The test dataset contains 2 billion rows. Each row contains 20 columns. The size of each column is 20 bytes. The query range is 10 million rows. The maximum OPS is 5000. After the preceding data is prepared, run a major compaction and wait for the system to complete the major compaction. Run a warm-up test for 20 minutes, and then run a formal test for 20 minutes.

The configuration file of YCSB is as follows:

```
recordcount=10000000
operationcount=2000000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=false
fieldcount=1
fieldlength=20
readproportion=1.0
updateproportion=0.0
scanproportion=0
insertproportion=0
requestdistribution=uniform
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -p maxexecutiontime=1200 -p target=5000
```

## Read data within a specified range

The test dataset contains 2 billion rows. Each row contains 20 columns. The size of each column is 20 bytes. The query range is 10 million rows. 50 rows are read at a time. The maximum OPS is 5000. After the preceding data is prepared, run a major compaction and wait for the system to complete the major compaction. Run a warm-up test for 20 minutes, and then run a formal test for 20 minutes.

The configuration file of YCSB is as follows:

```
recordcount=10000000
operationcount=2000000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=false
fieldcount=1
fieldlength=20
readproportion=0.0
updateproportion=0.0
scanproportion=1.0
insertproportion=0
requestdistribution=uniform
maxscanlength=50
hbase.usepagefilter=false
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 100 -p columnfamily=f -p maxexecutiontime=1200 -p target=5000
```

## Write data into a single row

Insert a column to the table at a time. The size of each column in the row is 20 bytes. Run the test for 20 minutes. The maximum OPS is 50000.

The configuration file of YCSB is as follows:

```
recordcount=2000000000
operationcount=100000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=false
fieldcount=1
fieldlength=20
readproportion=0.0
updateproportion=0.0
scanproportion=0
insertproportion=1.0
requestdistribution=uniform
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=testwrite -threads 200 -p columnfamily=f -p maxexecutiontime=1200 -p target=50000
```

## Write data into multiple rows

Insert one column into the table at a time. The size of each column is 20 bytes. Write data into 100 rows per batch. Run the test for 20 minutes. The maximum OPS is 2000.



```
recordcount=2000000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
fieldcount=1
fieldlength=20
cyclickey=true
readallfields=false
readproportion=0
updateproportion=0
scanproportion=0
insertproportion=0.0
batchproportion=1.0
batchsize=100
requestdistribution=uniform
```

Run the following command to launch YCSB:

```
bin/ycsb run hbase10 -P <workload> -p table=testwrite -threads 100 -p columnfamily=f -p maxexecutiontime=1200 -p target=2000
```

## Compression ratio benchmark tests

The following compression ratio benchmark tests all follow the same procedure. Manually trigger a flush and major compaction by inserting 5 million rows into the table through YCSB. After the data is inserted into the table, check the size of the table.

| The number of columns in each row. | The size of each column. |
|------------------------------------|--------------------------|
| 1                                  | 10                       |
| 1                                  | 100                      |
| 20                                 | 10                       |
| 20                                 | 20                       |

The configuration file of YCSB is as follows:

```

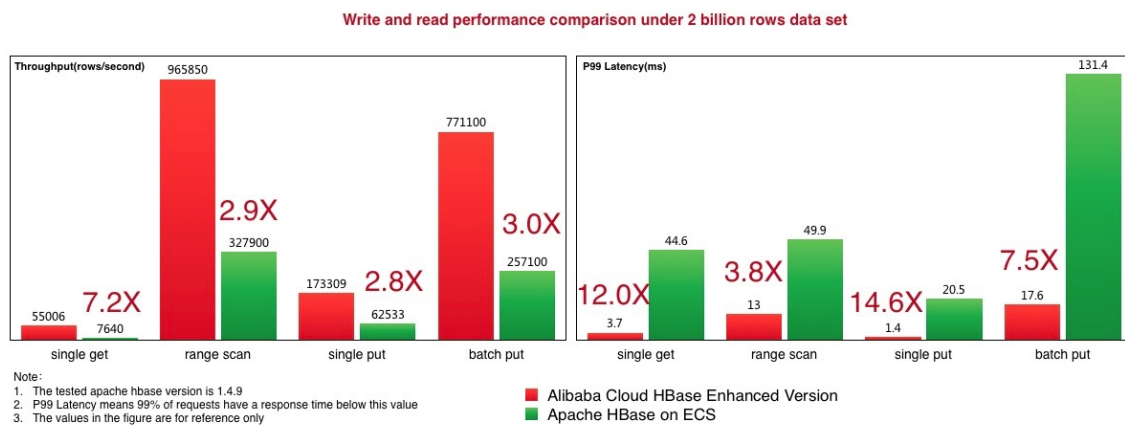
recordcount=5000000
operationcount=150000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
readallfields=false
fieldcount=<Number of columns in each row>
fieldlength=<Size of each column>
readproportion=1.0
requestdistribution=uniform

```

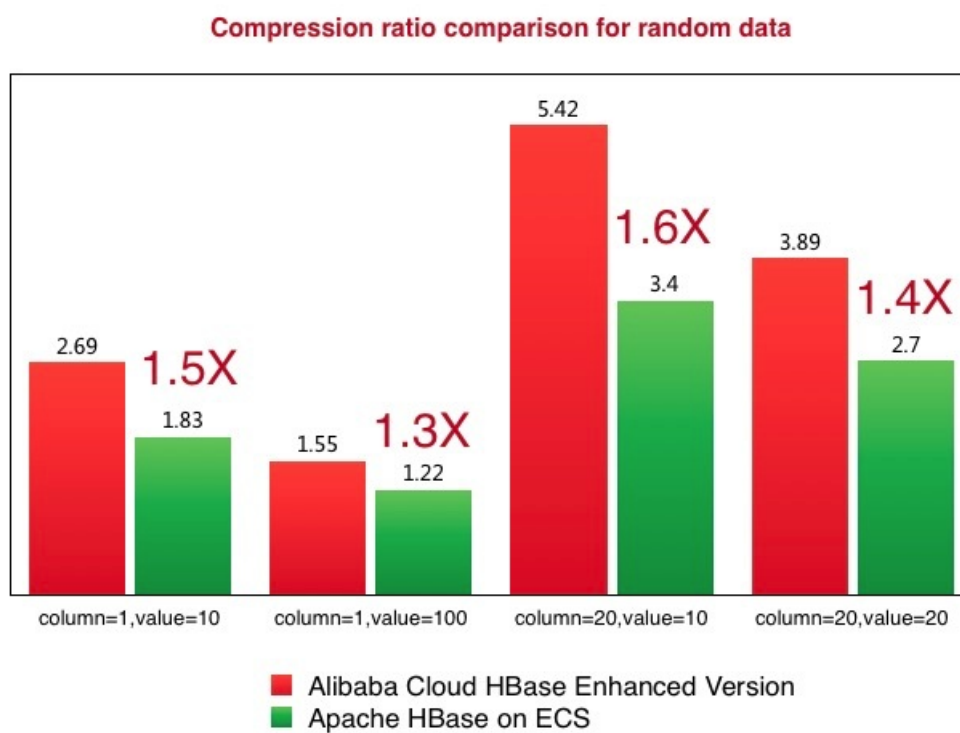
Run the following command to insert data:

```
bin/ycsb load hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -s
```

## 5.4. Benchmark results



## Compression ratio benchmark results



The benchmark results may vary depending on the scenario.

## 6.Cluster management

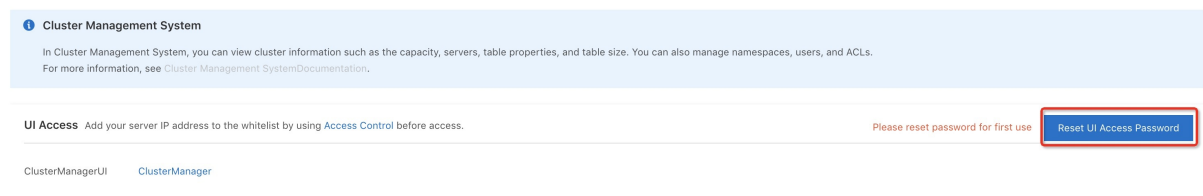
### 6.1. Cluster management system

Before you connect to the cluster management system, you must add the IP address of your client to the whitelist of ApsaraDB for HBase. For more information about how to configure the whitelist, see [Configure the whitelist](#).

#### Configure the whitelist

#### Set the password

Before you connect to the cluster management system, make sure that you have set the [password](#). You can click Reset UI Access Password on the page in the following figure to reset the password.

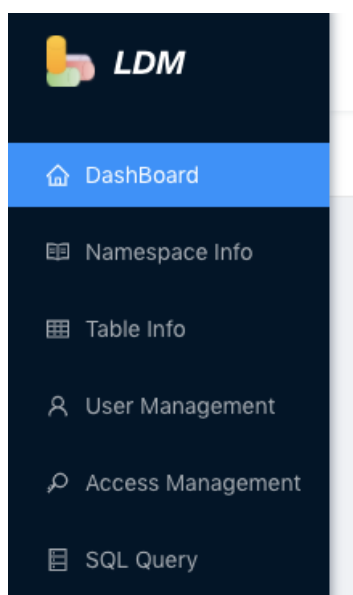


#### Introduction

ApsaraDB for HBase Performance-enhanced Edition provides a new cluster management system. You can view the cluster space usage, servers, table properties, table sizes, and other information in this system. You can also manage namespaces, users, and ACLs in the cluster management system. The portal of the cluster management system is available in the ApsaraDB for HBase console. You can click the ClusterManager hyperlink to log on to the system.

The cluster management system includes the following sections:

#### Homepage



On the homepage, you can view the following cluster information:

## Cluster space usage

In this section, you can view the space usage of your cluster. Based on this information, you can decide whether to expand the disk of your cluster.

## Group information

In this section, you can view all groups in your cluster, and manage these groups. For more information about groups, see [Manage groups](#).

## Server list

In this section, you can view all active RegionServers, and their statuses. For example, you can click a RegionServer name to view details about the RegionServer, including the region information. You can add RegionServers to different groups in order to isolate them. For more information about groups, see [Manage groups](#).

## Cluster health information

This section shows faulty RegionServers and inactive RegionServers. When ApsaraDB for HBase balances or splits regions, the regions will be temporarily inactive. However, if you find out that a region or RegionServer remains inactive for a long period of time, submit a ticket to report this problem or consult the ApsaraDB for HBase DingTalk group.

## Namespace information

On the Namespaces page, you can view all namespaces in your cluster, and manage these namespaces. For more information about how to manage namespaces, see [Manage namespaces](#).

## Tables

On the tables page, you can view all tables in your cluster and their properties. You can click a table name to view detailed information about the table, and the region information.

## Users

On the users page, you can view the users of your cluster, and their permissions. You can create users, change passwords, and delete users. For more information, see [Manage users and ACLs](#).

## Permissions

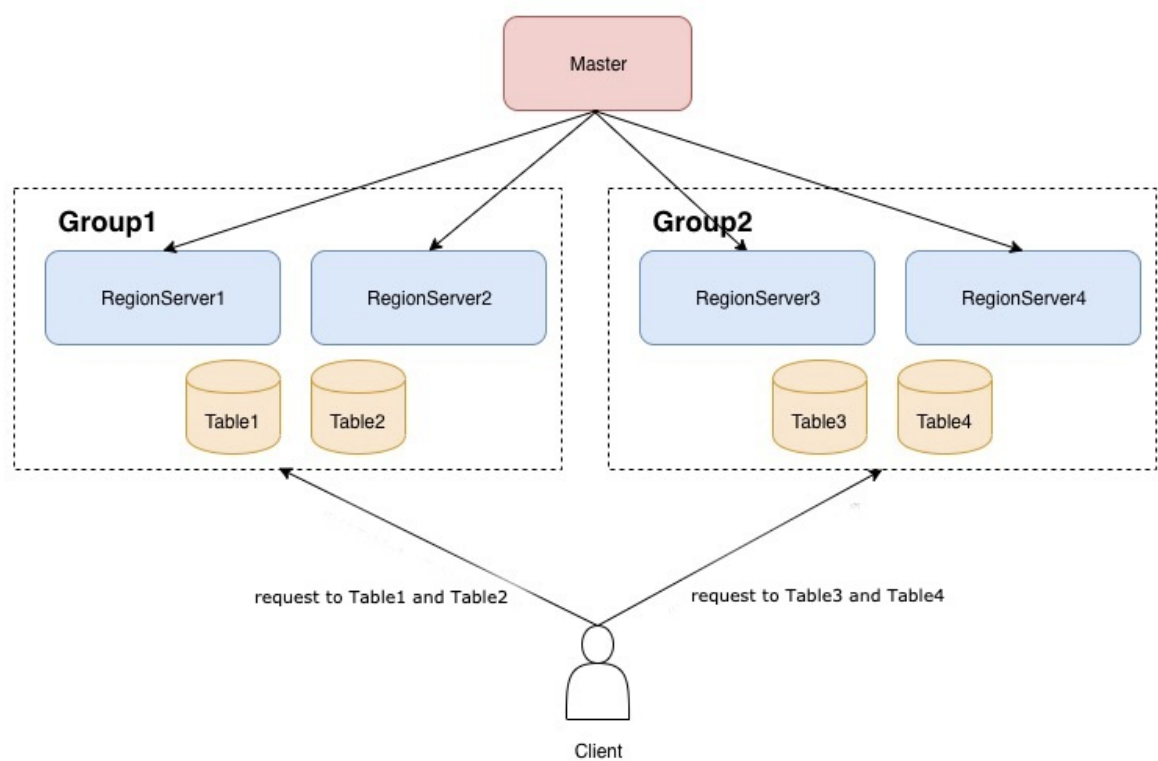
On the permissions page, you can grant permissions to a user or revoke permissions from a user. For more information, see [Manage users and ACLs](#).

## SQL queries

On the SQL queries page, you can run SELECT statements to query tables. For more information, see [SQL queries](#).

# 6.2. Manage groups

When multiple users or services use the same ApsaraDB for HBase cluster, the resource preemption occurs. The reads and writes of some important online services may be adversely affected by multiple concurrent reads and writes of offline services. ApsaraDB for HBase provides the Group feature to isolate resources for multiple tenants. To isolate the resources, you can assign RegionServers to different groups and host different tables in the group.



The preceding figure shows that RegionServer1 and RegionServer2 are assigned to Group1 and RegionServer3 and RegionServer4 are assigned to Group2. Table1 and Table2 are also moved to Group1. In this case, all regions of Table1 and Table2 are distributed to RegionServer1 and RegionServer2 in Group1. All regions of Table3 and Table4 are distributed to RegionServer3 and RegionServer4 in Group2. Therefore, the requests to Table1 and Table2 are sent to RegionServer1 and RegionServer2. The requests to Table3 and Table4 are sent to RegionServer3 and RegionServer4. This mechanism helps you to isolate resources.

ApsaraDB for HBase Performance-enhanced Edition allows you to manage groups in the [Cluster Management System](#).

You can view the information of all groups of the current cluster on the home page. If you have not created any group, the system provides a default group. All RegionServers and tables belong to the default group.

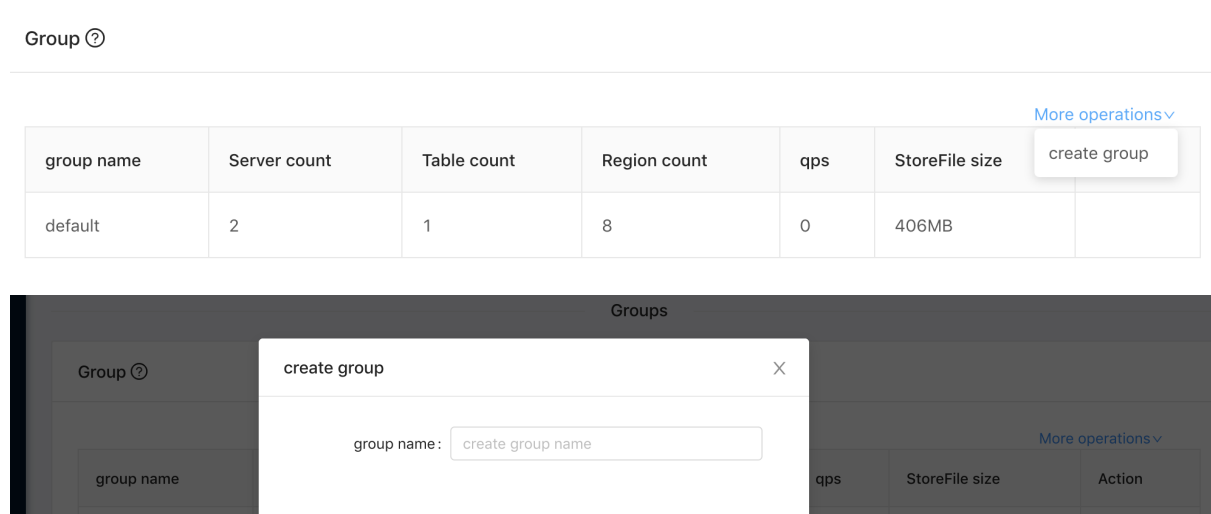
Group ⓘ

[More operations](#)▼

| group name | Server count | Table count | Region count | qps | StoreFile size | Action |
|------------|--------------|-------------|--------------|-----|----------------|--------|
| default    | 2            | 1           | 8            | 0   | 406MB          |        |

Create a group

On the Groups page, choose More > Create Group. After a new group is created, the numbers of servers and tables in this group are both 0. You must move servers and tables to this group.

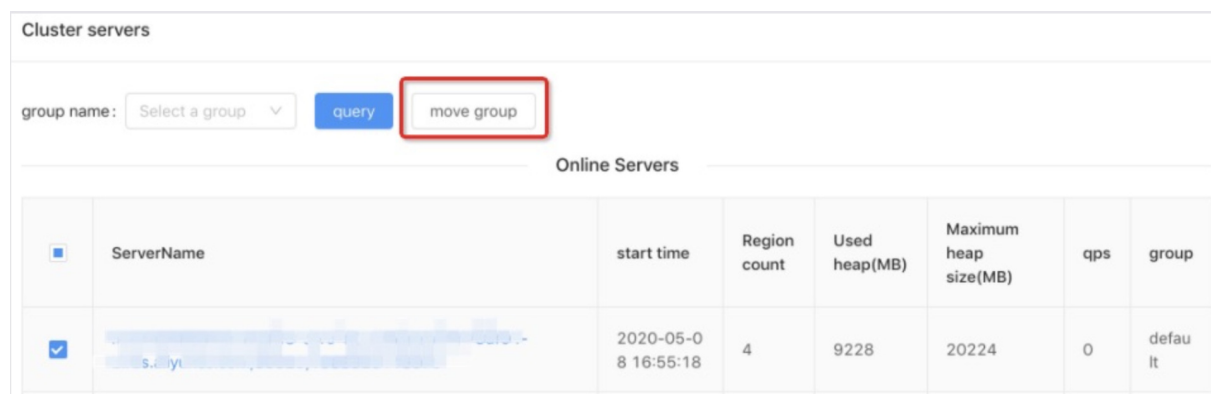


## Delete a group

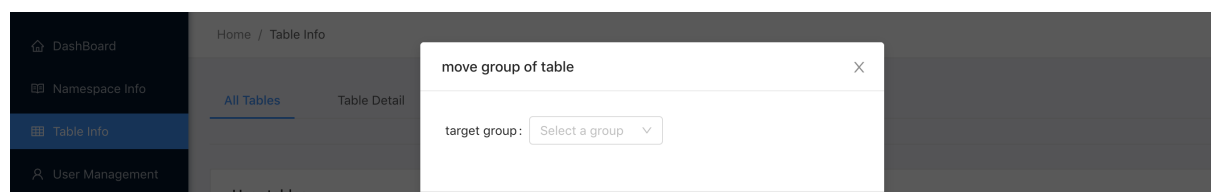
Click Delete in the Actions column on the home page of the Cluster Management System to delete the corresponding group. **Note: Before you delete a group, you must delete all tables and servers in the group. Otherwise, the deletion fails.**

## Move a RegionServer to a group

By default, all RegionServers belong to the default group. You must move the servers to the required group before you use them. Follow these steps: 1. On the home page of Cluster Management System, select the RegionServer in the Cluster servers table and click Move.



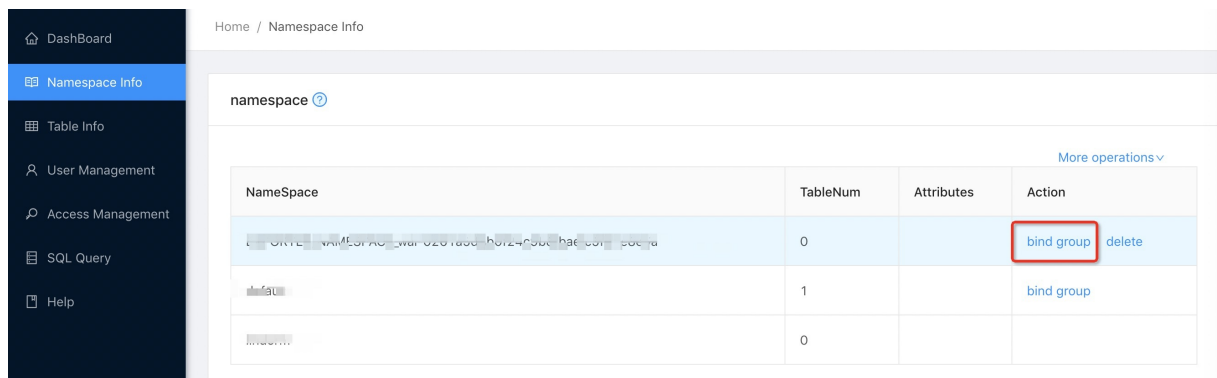
2. In the dialog box that appears, select the target group from the drop-down list and click OK.



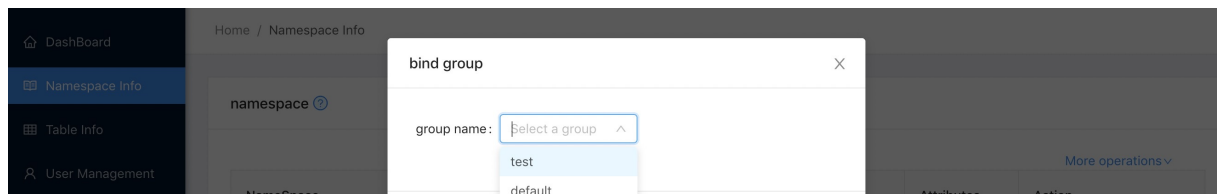
**Notes:**1. If you move a table to a group that does not contain any RegionServer, the region of the table cannot be accessed because no online server is available.2. We recommend that you distribute at least two RegionServers to a group. If one of the RegionServers stops responding, you can access another server in the same group. If the only RegionServer in a group stops responding, all tables in this group cannot be accessed.3. After you move a RegionServer to a group, the online regions on this RegionServer are shared among all other RegionServers in the group and requests are evenly distributed to these regions. This ensures the server load balancing.

## Configure the group for a namespace

If you do not configure a group for a namespace, all tables that you create belong to the default group. Regions are only available in the default group. If you do not want to group tables every time you create tables, you can specify a group for the namespace. All tables created in this namespace can be added to the same group. 1. On the Namespaces page in the Cluster Management System, click Bind Group in the Actions column of the corresponding namespace.



2. In the dialog box that appears, select the target group from the drop-down list and click OK.

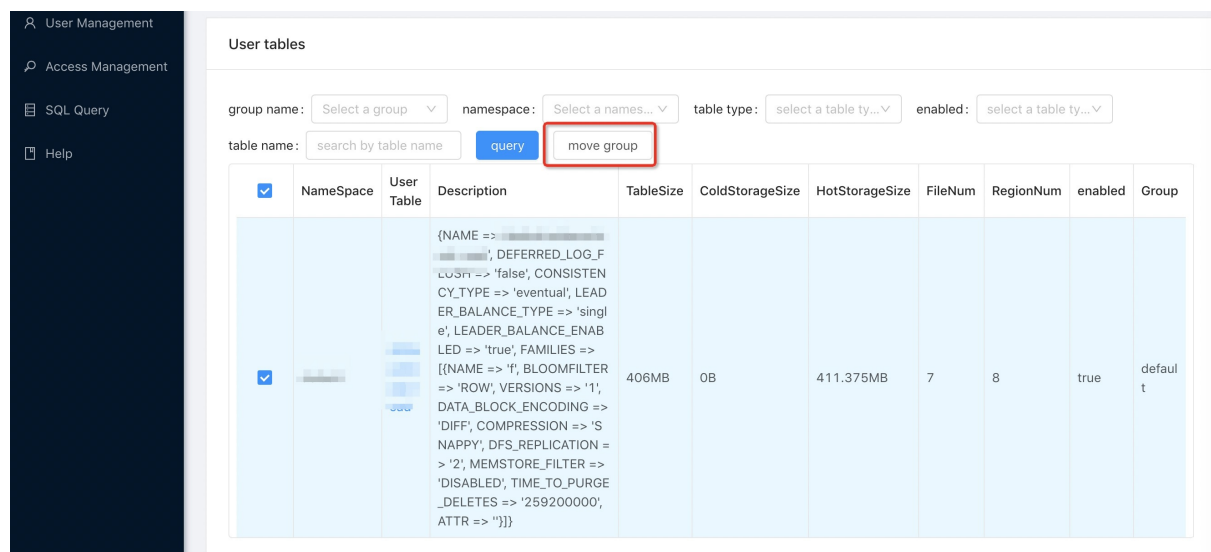


**Notes:**1. If you modify the group information of the namespace, the groups of the existing tables in this namespace are not changed.2. If you have bound a group to the namespace, all tables created in this namespace also belong to this group. You can move the tables to other groups.

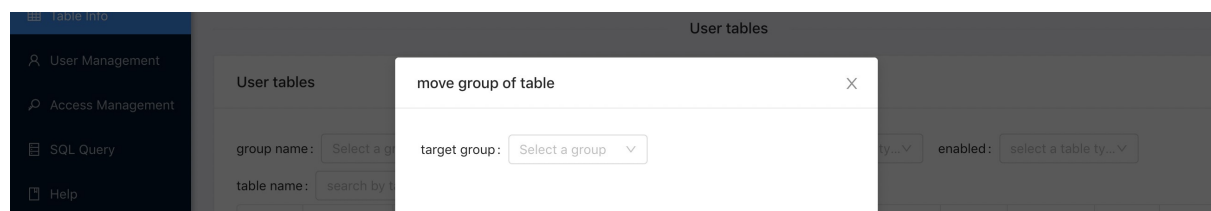
## Configure a group for tables

Follow the steps to change the group of a table: 1. On the User tables page of Cluster Management System, select the table and click Move.





2. In the dialog box that appears, select the target group from the drop-down list and click OK.



**Notes:** You must move tables to the group that contains RegionServers. Otherwise, these tables cannot be found in existing regions.

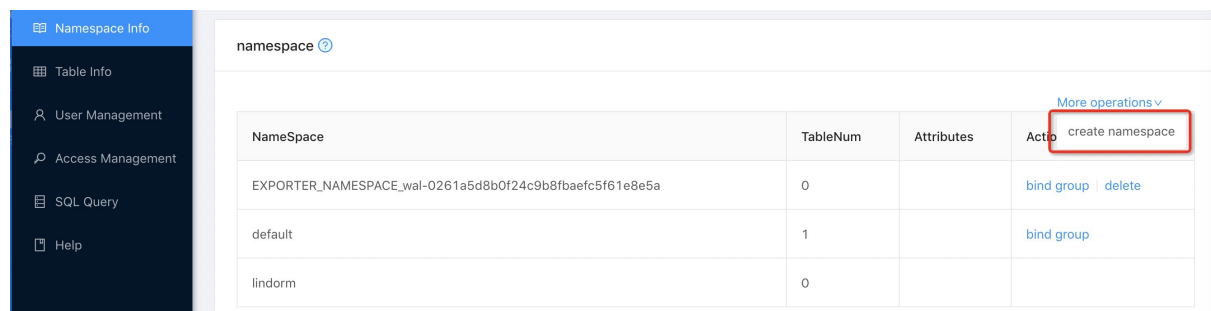
## 6.3. Manage namespaces

Namespaces in ApsaraDB for HBase are defined as logical groups of tables. Different namespaces can have tables with the same name. Different tenants have different namespaces, and each tenant can only access and manage authorized namespaces. This means that different users can only query the table sets within their own namespaces. ApsaraDB for HBase provides a default namespace. If you do not specify a namespace, the system uses the default namespace.

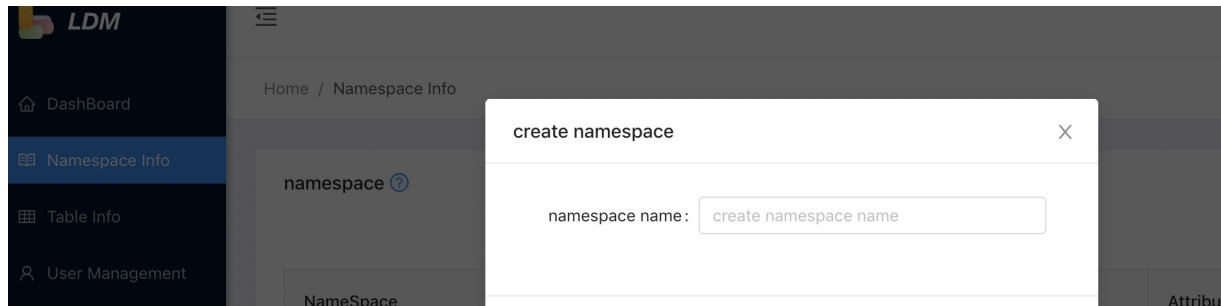
You can use the Java API or HBase Shell to create and delete a namespace. You can also log on to the [Cluster Management System](#) to manage namespaces.

### Create a namespace

1. On the Namespaces page, choose More > Create Namespace.

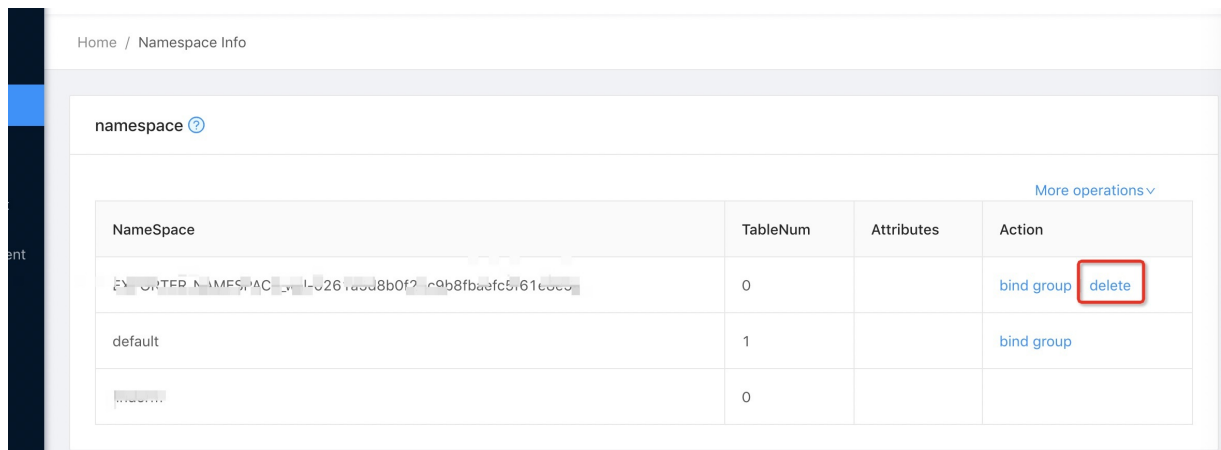


2. In the dialog box that appears, enter the namespace name and click OK.



## Delete a namespace

On the Namespaces page, click Delete in the Actions column for a namespace to delete the namespace.



**Note:** Before you delete a namespace, you must delete all tables in the namespace. Otherwise, the deletion fails.

## Bind a group to the namespace

If you bind a group to the namespace, the tables that are created in this namespace can be automatically associated with the group. For more information, see [Manage groups](#).

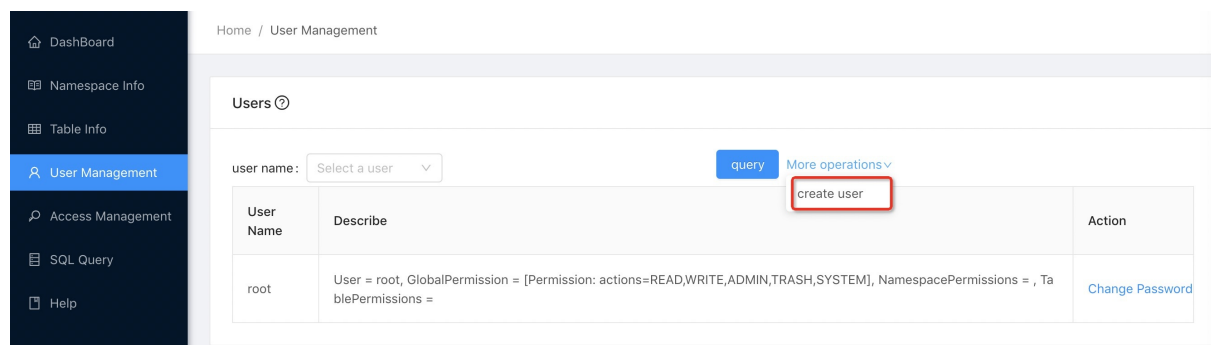
# 6.4. Manage users and ACL

ApsaraDB for HBase Performance-enhanced Edition provides simplified user authentication and Access Control List (ACL) management. You only need to configure the username and password for user authentication. The passwords are encrypted and stored in the server and are transmitted only for authentication. Even if the ciphertext is intercepted, the encrypted data cannot be decrypted or reused.

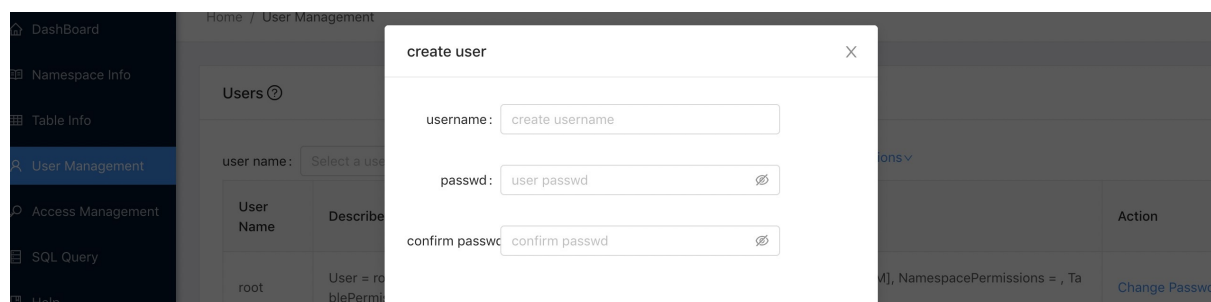
You can log on to the [Cluster Management System](#) to manage users. On the Users page, all the users of the current cluster are listed. After you purchase a cluster, the system creates an account that has all permissions on the cluster and you can manage the cluster by using this account. Both the username and password of this account are root. You can change the password of this account or delete it in the Cluster Management System.

## Create a user

1. On the Users page, choose More > Create User.



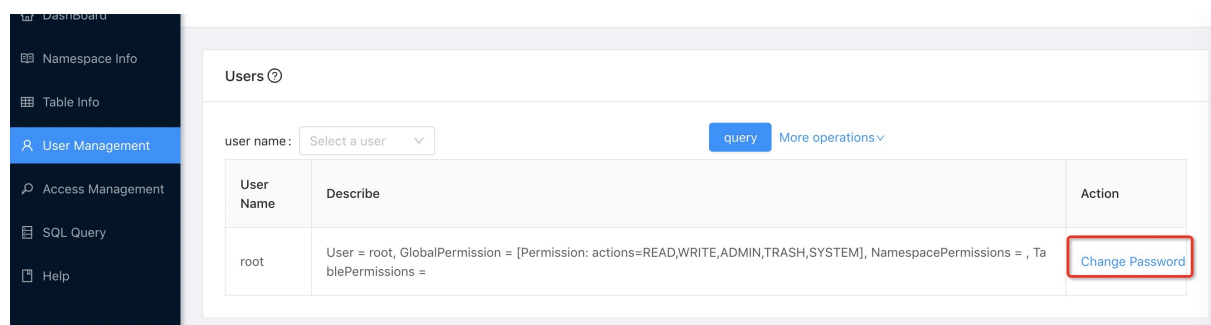
2. In the dialog box that appears, enter the username and password, and click OK.



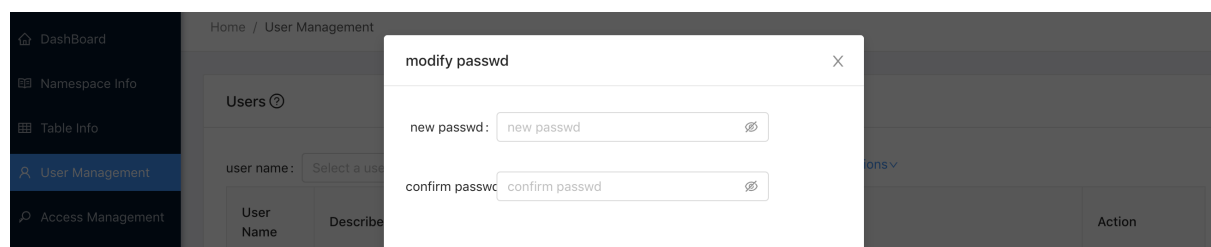
**Notes:** 1. Passwords are stored in ciphertext on the ApsaraDB for HBase Performance-enhanced Edition server. After you create a new user, the password is not displayed. You are responsible for remembering your password. If you lose your password, you must change your password. 2. A new user does not have any permissions. You must grant the required permissions to the user on the Permissions page. For more information, see Manage ACLs.

## Change the password

1. Click Change Password on the Users page in the Cluster Management System.

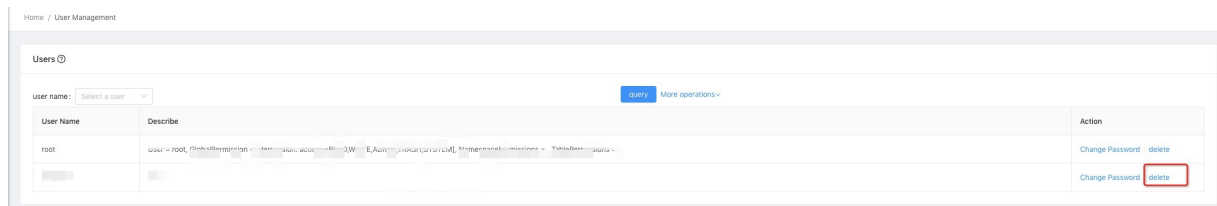


2. In the dialog box that appears, enter the new password and click OK.



## Delete users from a user group

1. Click Delete in the Actions column on the Users page in the Cluster Management System to delete the user.



## Manage ACL permissions

### Permission types

The server of ApsaraDB for HBase Performance-enhanced Edition can determine what a user can do based on the permissions of the user. For example, if User1 only has the read permission on the Table1, an error message is returned if User1 tries to write Table1 or read Table2. The following permission types are supported in ApsaraDB for HBase Performance-enhanced Edition:

### WRITE permissions

The users with WRITE permissions can run statements such as PUT, BATCH, DELETE, INCREMENT, APPEND, and CHECKANDMUTATE to write tables.

### READ permissions

The users with READ permissions can run statements such as GET, SCAN, and EXIST to read tables, or run the statements such as getTableDescriptor, listTables, and listNamespaceDescriptors to retrieve descriptors and namespaces of tables.

### ADMIN permissions

The ADMIN permissions allow users to manage tables or data by using the Data Definition Language (DDL) statements such as createTable, enableTable, and disableTable. However, these permissions do not include the delete permissions on tables or data. The ADMIN permissions also allow users to manage namespaces by using the DDL statements such as createNamespace.

### TRASH permissions

To avoid accidental operations in which tables may be deleted or cleared, only the users with the TRASH permissions can use the DDL statements such as truncateTable and deleteTable.

### SYSTEM permissions

Only the users with SYSTEM permissions can run the COMPACT and FLUSH statements. In addition, if you want to use Big Datahub Service (BDS) to migrate and synchronize data, you must use the account with SYSTEM permissions.

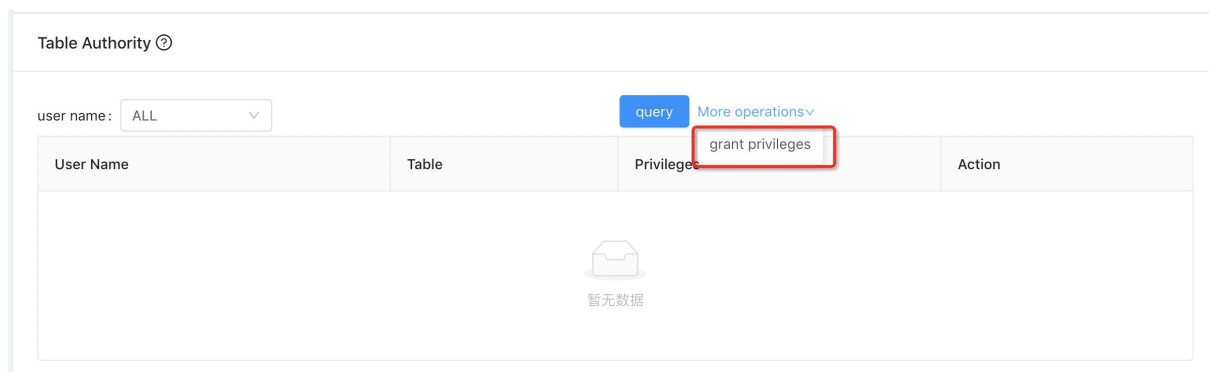
### Classified permissions

You can manage permissions at three levels: Global, Namespace, and Table. Only one of these permissions can take effect at a time. For example, if you grant the read and write permissions at Global level to User1, you can use User1 to read and write all tables of all namespaces. If you grant the read and write permissions of Namespace1 to User2, you can use User2 to read and write all tables of Namespace1. **Note:** Only the users with the ADMIN permissions at Global level can create and delete namespaces.

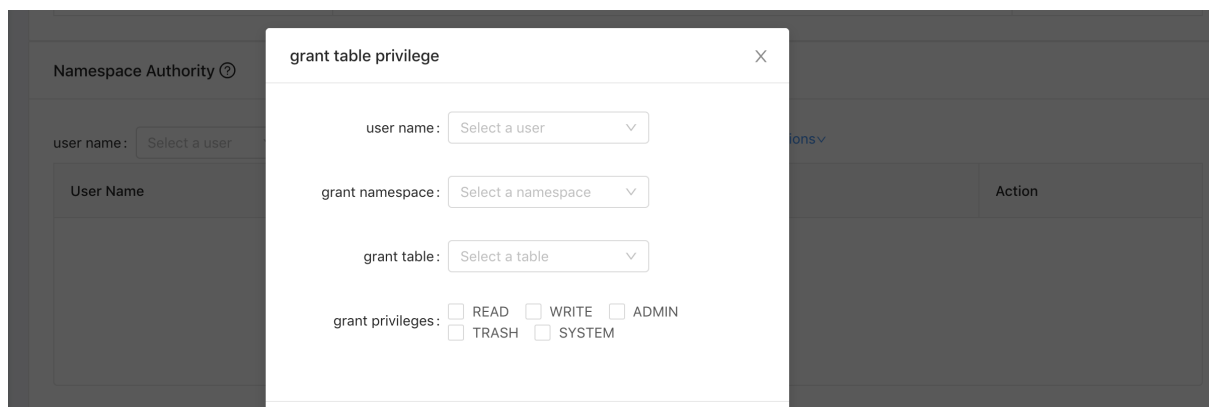
## Permissions

### Grant permissions to a user

You can grant permissions to a user on the Permissions page in the Cluster Management System. To grant the READ permission of a table to a user, follow these steps: 1. On the Permissions page, choose More > Grant Permission.



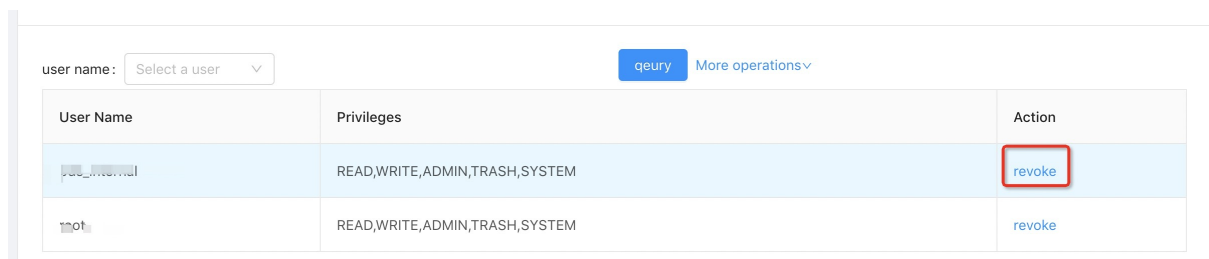
2. In the dialog box that appears, select a user, a namespace, a table, select the READ permission, and then click OK.



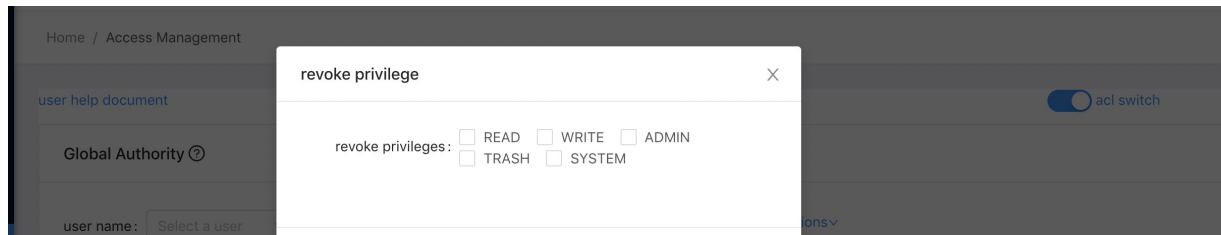
### Revoke permissions

You can revoke permissions of a user in the Cluster Management System. Each user may have permissions at multiple levels. You can revoke permissions for a user. Follow these steps:

1. Go to the Permissions page which displays a list of permissions. Find the user for whom you want revoke permissions and click Revoke in the Actions column.

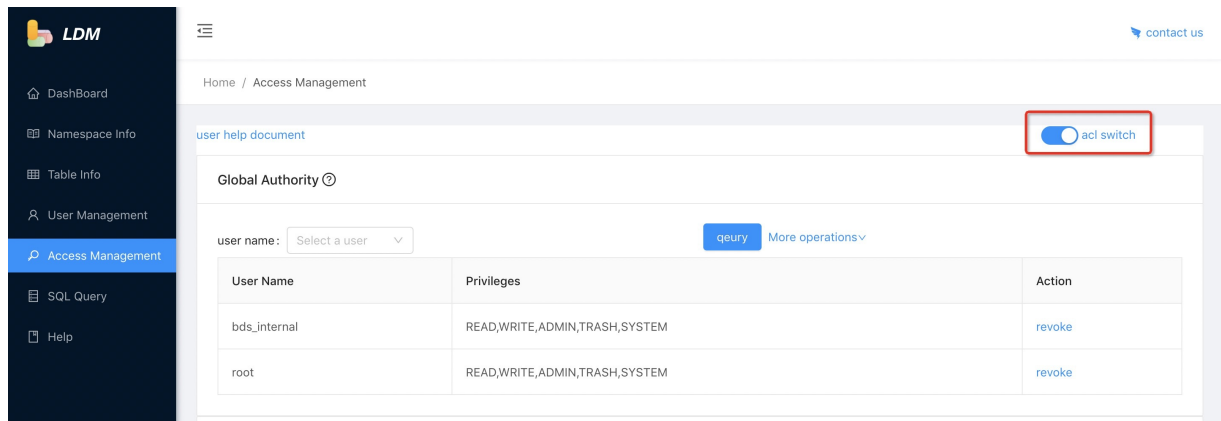


2. In the dialog box that appears, all the permissions on the current object (Global, Table, or Namespace) are listed. Select permissions to be revoked and click OK.



## Enable or disable ACL

You can disable the user authentication and ACL management features. After you disable the ACL feature, the username and password are not required for subsequent access (such as access by using APIs, SQL, and non-Java methods). There are no limits on users to manage clusters. After you enable or disable ACL management, the settings take effect immediately. You do not need to restart the cluster. After you enable ACL management, you must provide a username and password to connect to the service. Otherwise, the client cannot be authenticated and an error message is returned. If a username and password are provided, the client can be authenticated and connected to the service. However, if the user tries to perform management operations outside of their permissions, they are disconnected from the service.



## 6.5. Data query

You may need to query a data entry stored in ApsaraDB for HBase when you develop, debug, or maintain your services. You can use HBase Shell to write GET and SCAN queries. ApsaraDB for HBase Performance-enhanced Edition also provides a SQL query interface in the [cluster management system](#) for you to query data. If you are familiar with the SQL syntax, you can use this interface to query HBase tables.

To use this feature, navigate to the SQL queries page of the cluster management system. Before you query a table, you must select the namespace that stores the table. After you select a namespace, the table list on the right side displays all the tables in the namespace. You can click a table name to view the schema of the table. ROW represents a rowkey, and COL represents a predefined column name. You can write a SELECT query based on the table schema.

The following example shows how to perform a SQL query:

Select a namespace, for example, 'default'.

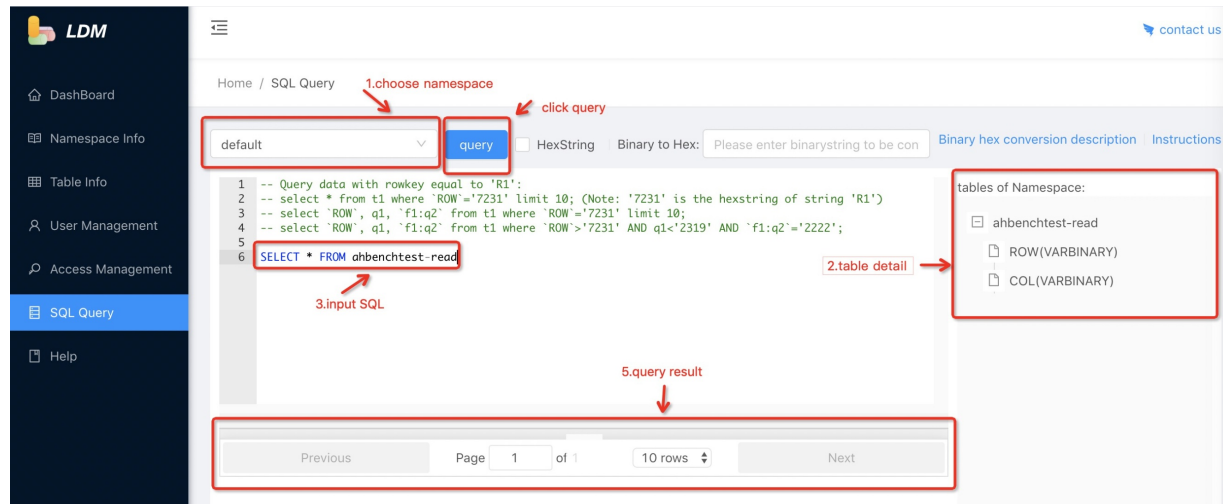
Check the schema of the table that you want to query in the right-side table list.

Enter a SQL statement into the editor.

Click Run or press 'CTRL + Enter' (Windows)/'command + return' (macOS) to run the SQL statements.

The query results are displayed below the editor. You can also check error messages in this area.

The following figure shows the steps.



**Note:** The SQL query editor in the cluster management system does not support the use namespace syntax. You must manually select a namespace from the namespace list in the upper-left corner of the editor.

## Limits

Before you make a SQL query, make sure that you have read and understand the following limits:

The SQL query editor only supports the 'SELECT' statement. If you want to modify data, use the command-line interface or develop an application that uses the ApsaraDB for HBase API.

To ensure data security, the system returns up to 100 entries upon each query.

To query varbinary data with conditions, you must use hexadecimal strings as values.

The 'ROW' field represents a rowkey in an HBase table. The 'ROW' and 'qualifier' fields only support 'varbinary' data. If the 'qualifier' field does not belong to the family 'f', you must specify a family, for example, 'select `ROW`, q1, `f1:q2` from ...'.

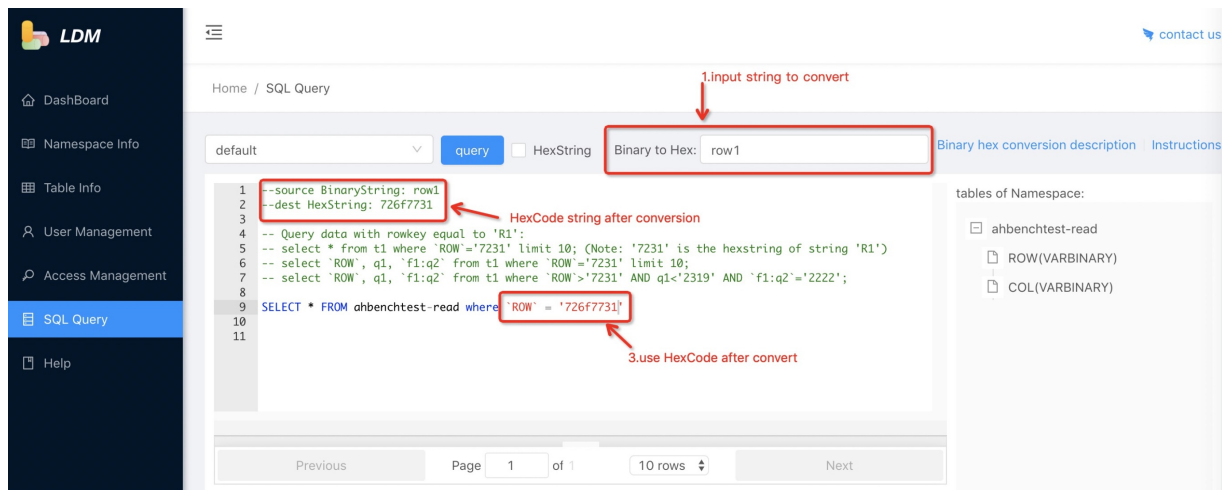
The 'ROW' and 'COL' fields are SQL reserved fields. You must enclose a 'ROW' or 'COL' in a pair of back quotes (` `). When you specify a family for a 'qualifier', you must also enclose it in a pair of back quotes.

## Convert binary strings to hexadecimal strings

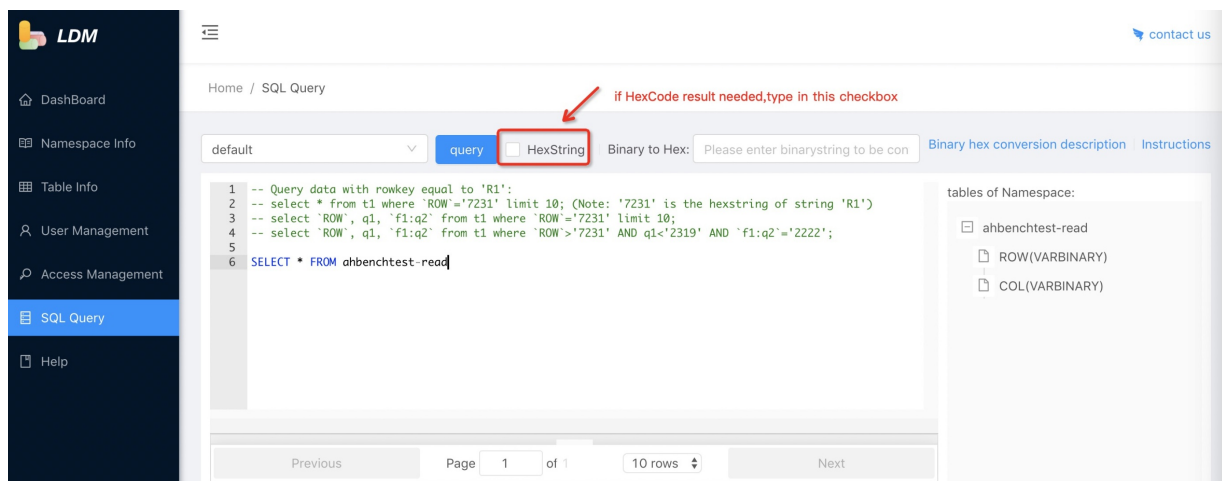
Data is stored in byte arrays ( `byte[]` ) in ApsaraDB for HBase. The varbinary values in the query results are displayed as HBase binary strings on the SQL Queries page.

If you want to query varbinary fields with conditions (using the **WHERE** clause that contains rowkey or other fields), you must specify the values of the query conditions as hexadecimal strings. Example: To query data with rowkey **r1**, the where condition in the SQL query must be **where rowkey='7321'** (The hexadecimal value of string **r1** is **7321**).

A converter is provided on the **SQL Queries** page for you to convert binary strings to hexadecimal strings. After you enter a binary string into the text box, the corresponding hexadecimal value is automatically displayed in the SQL editor. The following figure shows an example.



If you want to view query results displayed as hexadecimal strings, select the **HexString** option next to **Run**. The system then automatically converts varbinary values in the query results to hexadecimal strings. The following figure shows an example.





# 7.SQL manual

## 7.1. Manage connect strings

ApsaraDB for HBase Performance-enhanced Edition defines the following connect string format:

```
jdbc:hbase:url=http://<host>:<port>;serialization=<PROTOBUF>
```

### Establish a Java Database Connectivity (JDBC) connection

The following information is required to create a JDBC connection:

- Connect string
- Username and password

```
private static void initJDBCConnection() throws Exception {  
    String url = "jdbc:hbase:url=http://host:30060;serialization=PROTOBUF";  
    // If you use a lightweight Phoenix client, use the following connect string.  
    // String url = "jdbc:phoenix:thin:url=http://host:port;serialization=PROTOBUF";  
    String user = "root";  
    String password = "root";  
    Class.forName(HBaseDriver.class.getName()); //Register an HBase driver.  
    // If you use a lightweight Phoenix client, use the following driver.  
    // Class.forName(org.apache.phoenix.queryserver.client.Driver.class.getName());  
    Connection pconn = DriverManager.getConnection(url, user, password);  
}
```

### Multi-threading and connection pooling

It is not safe to share a JDBC connection among multiple threads in ApsaraDB for HBase Performance-enhanced Edition. Therefore, connection pooling is used to achieve high throughput and high concurrency, and manage connections at the same time. You can use the Spring or MyBatis framework to implement connection pooling. You can also manually create a connection pool, or create a connection for each thread.

The following Java code shows how to use ThreadLocal to create a private connection for threads:

```
private static ThreadLocal<Connection> resources = new ThreadLocal<Connection>();
private static List<Connection> connectionList = new ArrayList<Connection>();
public static Connection getConnection() throws Exception {
    Connection pconn = resources.get();
    if (pconn == null) {
        pconn = DriverManager.getConnection(url, userName, password);
        resources.set(pconn);
        synchronized (connectionList) {
            connectionList.add(pconn);
        }
    }
    return pconn;
}
public static void close() throws Exception {
    for (Connection pconn : connectionList) {
        if (pconn != null) {
            pconn.close();
        }
    }
}
```

## 7.2. Data types

This topic lists the standard SQL data types supported by the Java Database Connectivity (JDBC) of ApsaraDB for HBase Performance-enhanced Edition.

| SQL data type | Java data type    | Description   |
|---------------|-------------------|---|
| BOOLEAN       | java.lang.Boolean | 1 byte. Value 0 represents false and value 1 represents true.   |
| TINYINT       | java.lang.Byte    | 1 byte.   |
| SMALLINT      | java.lang.Short   | 2 bytes.  |
| INTEGER       | java.lang.Integer | 4 bytes.  |
| BIGINT        | java.lang.Long    | 8 bytes.  |
| FLOAT         | java.lang.Float   | 4 bytes. Value 0 in the sign bit indicates a negative number and value 1 indicates a positive number. |

| SQL data type            | Java data type       | Description   |
|--------------------------|----------------------|---|
| DOUBLE                   | java.lang.Double     | 8 bytes. Value 0 in the sign bit indicates a negative number and value 1 indicates a positive number. |
| DECIMAL(precision,scale) | java.lang.BigDecimal | Variable length.  |
| VARCHAR j                | ava.lang.String      | Variable length. Chinese characters are supported.  |
| BINARY(N)                | byte[]               | Fixed length. The data is padded with zeros or truncated to match the fixed length.                   |
| VARBINARY                | byte[]               | Variable length.  |

Data type descriptions:

- DECIMAL(precision,scale):
  - Expressed in decimal notation. The storage size (bytes) increases with the precision of the number. The DECIMAL type is used to store data that requires a high precision, such as payment amounts. For other data such as monitoring data, you can use the FLOAT or DOUBLE type.
  - The precision argument specifies the total number of digits in a number. Valid values: 1 to 38.
  - The scale argument specifies the number of decimal places in a number. Valid values: 0 to precision.
- binary(N): binary type data with a fixed length of N bytes. The data is padded with zeros or truncated to match the length N.
- varbinary: variable-length binary data. For primary key columns in a row, you can only set the last column to this type.

#### 1. DATE/TIME/TIMESTAMP and other date types

Currently, no date type is supported. You can use BIGINT or INT as a replacement. Typically, values of milliseconds or nanoseconds are stored as BIGINT. Values of seconds are stored as INT.

#### 2. CHAR(N)

Variable-length CHAR type. Strings must be stored as VARCHAR. You do not need to predefine the length.

#### 3. Arrays and other complex data types

Complex data types are currently unsupported.

## Limits

- The length of a primary key column cannot exceed 2 KB.
- The total length of all primary key columns cannot exceed 30 KB.
- The length of a non-primary key column cannot exceed 2 MB.

## 7.3. DDL syntax

This topic describes how to use Data Definition Language (DDL) to manage schemas and tables. For more information about how to use DDL to manage indexes, see [Manage indexes](#).

Schemas are equivalent to namespaces in HBase. Both of them are used to manage tables. HBase users are allowed to create and delete namespaces. Each user can have multiple namespaces and each namespace can contain multiple tables.

The system provides a predefined namespace `default`. If you do not specify a user-created schema, the predefined schema is used.

**Note:** Schema names are case sensitive.

## create schema

```
CREATE SCHEMA IF NOT EXISTS my_schema;  
CREATE SCHEMA my_schema;
```

## use

```
USE my_schema  
USE default
```

If you want to manage a table in a specified schema, you can run the USE statement to use the schema before you run the corresponding DDL or Data Manipulation Language (DML) statement. You can also place the schema name before the table name and separate them with a period (.), for example, `my_schema.dt`. In this example, `dt` is the table name.

## drop schema

```
DROP SCHEMA IF EXISTS my_schema  
DROP SCHEMA my_schema
```

To delete a schema, you must first delete all tables in the schema.

## list schema

Call the metadata interface of Java Database Connectivity (JDBC) to query schema information. The syntax is as follows:

```
// Establish a database connection pconn.
public void testListSchema() throws SQLException {
    ResultSet rs = pconn.getMetaData().getSchemas();
    List<Map<String, Object>> retList = convertList(rs);
    System.out.println("size:" + retList.size());
    for (Object o : retList) {
        Map<String, Object> tmpMap = (Map<String, Object>) o;
        for (Map.Entry<String, Object> entry : tmpMap.entrySet()) {
            System.out.print(entry.getKey() + "=" + entry.getValue() + " ");
        }
        System.out.println();
    }
    rs.close();
}

private List<Map<String, Object>> convertList(ResultSet rs) throws SQLException {
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    ResultSetMetaData md = rs.getMetaData();
    int columnCount = md.getColumnCount();
    System.out.println("count=" + columnCount);
    while (rs.next()) {
        Map<String, Object> rowData = new LinkedHashMap<String, Object>();
        for (int i = 1; i <= columnCount; i++) {
            rowData.put(md.getColumnLabel(i), rs.getObject(i));
        }
        list.add(rowData);
    }
    return list;
}
```

## Tables

### CREATE TABLE

```
CREATE TABLE [IF NOT EXISTS] table_name (
    column_definition , ...
CONSTRAINT PK PRIMARY KEY( pk_definitions ))
table_options ; Example:
```

```
use my_schema;
create table dt (
  p1 integer, p2 binary(3), c1 varchar, c2 decimal(22, 2),
  constraint pk primary key(p1, p2 desc));
create table dt (
  p1 integer, p2 integer, c1 varchar, c2 bigint,
  constraint pk primary key(p1 desc))'COMPRESSION'='ZSTD', 'VERSION'='1', 'TTL'=2592000;
```

You can place the schema name before the table name instead of running the USE statement. Example:

```
create table my_schema.dt (
  p1 integer, p2 binary(3), c1 varchar, c2 decimal(22, 2),
  constraint pk primary key(p1, p2 desc));
```

### table\_options

- **COMPRESSION** : specifies the table compression algorithm. Valid values: SNAPPY and ZSTD. By default, no compression algorithm is specified.
- **TTL** : specifies the validity period of the table. By default, the table never expires. Table property names are not case sensitive. You must enclose a table property name in a pair of single quotation marks (').

## DROP TABLE

```
use my_schema;
DROP TABLE dt;
DROP TABLE IF EXISTS dt;
DROP TABLE my_schema.dt;
```

## ALTER TABLE

You can run the ALTER TABLE statement to add a column to a table, or modify the properties of a table. Currently, you cannot use this statement to delete a column or change the data type of a column.

```
ALTER TABLE test ADD c4 VARCHAR;           -- Add a column
ALTER TABLE test ADD c3 INTEGER, c4 INTEGER, c5 VARCHAR; -- Add multiple columns
ALTER TABLE SET 'DYNAMIC_COLUMNS' = 'true'; -- Modify a table property
```

For more information about the user-configurable table properties, see [CREATE TABLE](#) .

## LIST TABLE

Call the JDBC metadata interface to list all tables in a specified schema. The syntax is as follows:

```
public void testListTable() throws SQLException {
    ResultSet rs = pconn.getMetaData().getTables(null, "my_schema", null, null);
    List<Map<String, Object>> retList = convertList(rs);
    for (Object o : retList) {
        Map<String, Object> tmpMap = (Map<String, Object>) o;
        for (Map.Entry<String, Object> entry : tmpMap.entrySet()) {
            System.out.print(entry.getKey() + "=" + entry.getValue() + " ");
        }
        System.out.println();
    }
    rs.close();
}

private List<Map<String, Object>> convertList(ResultSet rs) throws SQLException {
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    ResultSetMetaData md = rs.getMetaData();
    int columnCount = md.getColumnCount();
    System.out.println("count=" + columnCount);
    while (rs.next()) {
        Map<String, Object> rowData = new LinkedHashMap<String, Object>();
        for (int i = 1; i <= columnCount; i++) {
            rowData.put(md.getColumnLabel(i), rs.getObject(i));
        }
        list.add(rowData);
    }
    return list;
}
```

## DESCRIBE TABLE

Call the JDBC metadata interface to query the metadata of a specified table. The syntax is as follows:

```

public void testGetTableColumns() throws SQLException {
    ResultSet result = pconn.getMetaData().getColumns(null, "my_schema", "t1", null);
    List<Map<String, Object>> retList = convertList(result);
    for (Object o : retList) {
        Map<String, Object> tmpMap = (Map<String, Object>) o;
        System.out.print("namespace=" + tmpMap.get("NAMESPACE") + " ");
        System.out.print("column_name=" + tmpMap.get("COLUMN_NAME") + " ");
        System.out.print("is_primary_key=" + tmpMap.get("IS_PRIMARY_KEY") + " ");
        System.out.print("type_name=" + tmpMap.get("TYPE_NAME") + " ");
        System.out
            .print("lindorm_type_name=" + tmpMap.get("LINDORM_TYPE_NAME") + " ");
        System.out.println();
    }
    result.close();
}

private List<Map<String, Object>> convertList(ResultSet rs) throws SQLException {
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    ResultSetMetaData md = rs.getMetaData();
    int columnCount = md.getColumnCount();
    System.out.println("count=" + columnCount);
    while (rs.next()) {
        Map<String, Object> rowData = new LinkedHashMap<String, Object>();
        for (int i = 1; i <= columnCount; i++) {
            rowData.put(md.getColumnLabel(i), rs.getObject(i));
        }
        list.add(rowData);
    }
    return list;
}

```

## 7.4. DML syntax

ApsaraDB for HBase Performance-enhanced Edition supports the Data Manipulation Language (DML) statements such as SELECT, UPSERT, and DELETE. For more information, see [Manage indexes](#).

Unlike Relational Database Management System (RDBMS), ApsaraDB for HBase Performance-enhanced Edition does not support nested query statements such as GROUP BY and JOIN. Example:



```
-- Simple query statements.
select * from dt;      -- Scan the table dt.
select * from dt limit 10;
select * from dt where pk = 10;
select * from dt where pk > 10 limit 5, 20;  -- Skip 5 rows and return up to 20 rows.
select * from dt where pk > 10 limit 20 offset 5; -- This is equivalent to the statement 'limit 5,20'.
select c1 from dt where pk > 10;
select c1 as xx from dt where pk > 10;
-- Basic aggregated queries.
select count(*) from dt;
select count(c1) from dt;
select sum(c2) from dt;
select sum(c2), avg(c2), min(c2), max(c2) from dt;
```

The following section shows the statements that are not supported.

- Nested queries.
- ORDER BY. If you want to sort the result data, you can change the primary key order, create a secondary index, or sort data at the business logic layer.
- GROUP BY.
- JOIN.
- CASE WHEN.
- Functions and User Define Functions (UDFs) are not available.

## UPSERT

UPSERT is a combination statement of INSERT and UPDATE. The system executes UPDATE when the row exists and executes INSERT when the row does not exist. The usage of UPSERT is the same as that of PUT in ApsaraDB for HBase. Example:

```
upsert into dt (p1,p2,c1,c2) values(10, 20, 30, 40);
upsert into dt (p1,p2,c2) values(10, 20, 40); -- Write data to some of the columns of a row.
upsert into dt (p1,p2,c1) values(10, 20, 30); -- Write data into the other columns of the same row.
-- If the row already exists, the system does not write data. If the row does not exist, the system inserts a row. This is the same situation as you use INSERT statement to write data into an existing row.
upsert into dt (p1,p2,c2) values(10, 20, 5) on duplicate key ignore;
-- Update data under certain conditions. Write data when the row exists and c1 equals 30.
upsert into dt (p1,p2,c2) values(10, 20, 5) on duplicate key update c1 = 30;
-- Insert multiple rows into the table.
upsert into dt (p1,p2,c1,c2) values(1,2,3,4), (2,3,4,5), (3,4,5,6);
```

## DELETE

Use the DELETE statement with caution. To ensure data security, ApsaraDB for HBase Performance-enhanced Edition limits on the use of DELETE. You can delete only one row by using a primary key .

- When you delete data by using the WHERE statement, you must provide a primary key that specifies a unique row.
- You are not allowed to run the DELETE statement to delete multiple rows in each request.
- When you delete data, the system does not query the data first and then delete the data. The data is attached with a delete marker. Therefore, the delete marker can be attached no matter whether the row to be deleted exists or not.

Example:

```
delete from dt where p1 = 10 and p2 = 20;
-- The following statement is invalid.
delete from dt where p1 = 10; -- The table dt has two primary keys. The statement p1=10 cannot specify a unique row.
delete from dt where c1 = 30; -- You must provide the primary key that can specify a unique row.
```

## 7.5. Manage indexes

ApsaraDB for HBase currently only supports global secondary indexes. Therefore, the Data Definition Language (DDL) statements listed in this topic can only be used to manage global secondary indexes.

### CREATE INDEX

Before you create an index, make sure that you have specified the **MUTABILITY** property. This property specifies the write constraints of the primary tables. The data organization and consistency requirements of index tables vary based on different write constraints that are specified by the **MUTABILITY** property. You can set the **MUTABILITY** property to **IMMUTABLE** for the best performance and lowest costs of your service. If you set the **MUTABILITY** property to **MUTABLE\_ALL**, the service has the lowest performance and incurs the highest costs. We recommend that you set the **MUTABILITY** property to **MUTABLE\_LATEST** .

The following table lists the constraints of **MUTABILITY** .

| MUTABILITY     | Description  |
|----------------|--|
| IMMUTABLE      | Write data by rows. Modifying and deleting data are not supported.                           |
| IMMUTABLE_ROWS | Write data by rows. Modifying data is not supported, but deleting data by rows is supported. |

| MUTABILITY     | Description   |
|----------------|---|
| MUTABLE_LATEST | Write the data of one row in multiple times. Modifying and deleting data are supported, but writing data with custom timestamps is not supported. |
| MUTABLE_ALL    | No constraint is defined.   |

Syntax of CREATE INDEX:

```
CREATE INDEX [IF NOT EXISTS] index_name ON data_table_name ( indexed_column_definition , ... )  
[INCLUDE ( include_definition )][ index_table_options ];
```

Example:

```
-- You must specify the CONSISTENCY and MUTABILITY properties when you create a primary table.  
create table dt(  
  p1 integer, p2 integer, c1 integer, c2 integer, c3 integer,  
  constraint pk primary key(p1, p2)) 'CONSISTENCY'='strong', 'MUTABILITY'='mutable_latest';  
-- Create indexes on the primary table named dt.  
create index dt_idx1 on dt(c1);          -- Create a single-column index.  
create index dt_idx2 on dt(c1, c2 desc); -- Create a composite index in which columns are sorted in an order  
different from that in the primary table.  
create index dt_idx3 on dt(p2, p1);      -- Create an index in which the primary keys are sorted in an order dif  
ferent from that in the primary table.  
create index dt_idx4 on dt(p1 desc);  
create index dt_idx5 on dt(c3) include(c1, c2); -- Create an index that contains columns of the primary table  
: columns c1 and c2.  
create index dt_idx6 on dt(c3) 'INDEX_COVERED_TYPE'='COVERED_ALL_COLUMNS_IN_SCHEMA', 'COMPRESSIO  
N'='ZSTD';  
create index dt_idx7 on dt(c3) 'INDEX_COVERED_TYPE'='COVERED_DYNAMIC_COLUMNS';
```

**Note:** The value of the CONSISTENCY property cannot be changed in ApsaraDB for HBase. You must set the CONSISTENCY property to strong in your statements.

(1) The sort order for index key columns

When you create an index, you can specify a sort order for the columns in the index table. If you do not specify a sort order, columns in the index are sorted in ascending order. By default, if the columns in the primary table are sorted in descending order, the columns in the index table are not sorted in the same order. Example:

```
create table dt (p1 integer, p2 integer, c1 integer,  
  constraint pk primary key(p1, p2 desc)) 'CONSISTENCY'='strong', 'MUTABILITY'='mutable_latest';  
create index idx on dt(p2); -- Create an index that uses the ascending order of column p2, which is different f  
rom the sort order of the primary table.
```

## (2) Redundant columns

We recommend that you create indexes that contain columns of the primary table to avoid querying data from the primary table and reduce the query time. The indexes with redundant columns help you speed up queries. You can use the following methods to include redundant columns:

- Create an index that contains specified columns of the primary table by using the include statement, such as the index `dt_idx5`.
- Create an index that contains all columns of the primary table, such as the index `dt_idx6`. When you add columns to the primary table by using the `ALTER TABLE` statement, the columns can be **automatically** included in the index table. However, the dynamic columns in the primary table are not included in the index table.
- Create an index that contains all columns (including dynamic columns) of the primary table, such as the index `dt_idx7`. All columns of the primary table, including dynamic columns, are included in the index table.

**Note:** Indexes that include columns of the primary table consume the storage of the index table and increase the network traffic used to update the index.

## (3) Notes

- The data in the specified column of an index can be duplicate. Unique indexes are not supported in ApsaraDB for HBase.
- You can create index tables with the same name for different primary tables. For example, you can create indexes with the same index name `idx` for primary tables `dt1` and `dt2`. ApsaraDB for HBase does not allow you to create multiple indexes with the same name for a primary table.
- The system automatically checks duplicate index columns and forbids duplicate index tables. For example, if the `dt_idx1` and `dt_idx2` indexes are duplicate and you have created the `dt_idx2` index, when you create the `dt_idx1` index, an error is returned.

## DROP INDEX

Delete an index

```
DROP INDEX dt_idx1 ON dt;  
DROP INDEX IF EXISTS dt_idx1 ON dt;
```

When you delete a primary table, all indexes are also deleted.

## ALTER INDEX

```
ALTER INDEX idx ON dt DISABLE;  
ALTER INDEX idx ON dt REBUILD;  
ALTER INDEX idx ON dt UNUSABLE;  
ALTER INDEX idx ON dt USABLE;
```

- **DISABLE:** Disables an index. The index still exists. The data written to the primary table is no longer synchronized to the index and queries no longer hit the disabled index.
- **REBUILD:** Iterates the primary table and rebuilds all data in the primary table. During the rebuilding

process, queries cannot hit the index but the data written to the primary table is still synchronized to the index tables.

- UNUSABLE: Synchronizes data written to the primary table to index tables. The index tables cannot be queried during the synchronization.
- USABLE: Reuses an unusable index and queries can hit the index.

If you have `disabled` an index, you must `rebuild` the data before you can reuse this index.

## LIST INDEX

You can run this statement to query information about index tables associated with a primary table and retrieve the metadata of ApsaraDB for HBase Performance-enhanced Edition.

```
public void testListIndex() throws SQLException {
    ResultSet rs = pconn.getMetaData().getIndexInfo(null, null, "test_index", false, false);
    List<Map<String, Object>> retList = convertList(rs);
    for (Object o : retList) {
        Map<String, Object> tmpMap = (Map<String, Object>) o;
        System.out.print("namespace=" + tmpMap.get("NAMESPACE"));
        System.out.print("column_name=" + tmpMap.get("COLUMN_NAME"));
        System.out.print("is_primary_key=" + tmpMap.get("IS_PRIMARY_KEY"));
        System.out.print("type_name=" + tmpMap.get("TYPE_NAME"));
        System.out.print("lindorm_type_name=" + tmpMap.get("LINDORM_TYPE_NAME"));
        System.out.println();
    }
    rs.close();
}

private List<Map<String, Object>> convertList(ResultSet rs) throws SQLException {
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    ResultSetMetaData md = rs.getMetaData();
    int columnCount = md.getColumnCount();
    System.out.println("count=" + columnCount);
    while (rs.next()) {
        Map<String, Object> rowData = new LinkedHashMap<String, Object>();
        for (int i = 1; i <= columnCount; i++) {
            rowData.put(md.getColumnLabel(i), rs.getObject(i));
        }
        list.add(rowData);
    }
    return list;
}
```

## Manage indexes by using DML

You cannot directly query an index table. Index tables are queried when you query the associated primary table. When you insert a row into the primary table, ApsaraDB for HBase Performance-enhanced Edition automatically synchronizes the changes to all the associated index tables. If you delete data from the primary table, the data is also deleted from all associated index tables. The following section shows how to hit the index when you query data and how to use the hint feature to choose indexes to improve the query efficiency.

Example:

```
-- Create a primary table and indexes.
create table dt (
  p1 integer, p2 integer, c1 integer, c2 integer,
  constraint pk primary key(p1, p2 desc)) 'CONSISTENCY'='strong', 'MUTABILITY'='mutable_latest';
create index idx1 on dt (c1);
create index idx2 on dt (c2) 'INDEX_COVERED_TYPE'='COVERED_ALL_COLUMNS_IN_SCHEMA';
-- Query data.
select * from dt where c1 = 7; -- Query data by using the idx1 index. The system needs to query the primary
table after it looks up the index table.
select * from dt where c2 = 7; -- Query data by using the idx2 index. The index contains all columns of the pri
mary table. The system does not need to look up the primary table.
```

## 7.6. Advanced features

This topic describes how to use SQL statements in ApsaraDB for HBase Performance-enhanced Edition to implement NoSQL features of native HBase. These features include multi-versioning, timestamps, and schema-free. This topic also introduces some advanced features only available in ApsaraDB for HBase Performance-enhanced Edition.

In ApsaraDB for HBase Performance-enhanced Edition, the data in a cell can have multiple versions (column-oriented versioning). Versions are described by using timestamps. Therefore, you can perform the following tasks:

- Label the data that you write into a table with a specified timestamp.
- Read data of a specified version. For example, you can read the data of the latest five versions or read the data of all versions.
- Read data within a specified timestamp range. For example, you can read data with a timestamp later than  $t_1$ , or read data with a timestamp in the range of  $[t_1, t_2]$ .
- Read data with a combination of version numbers and timestamps. For example, you can read data of the latest five versions with a timestamp range of  $[t_1, t_2]$ .

The concepts of multi-versioning and timestamps are not defined in standard SQL statements. Therefore, we need to extend the standard SQL statements.

- Primary key columns do not support multi-versioning and timestamps. Only non-primary key columns support these two features.
- Multi-versioning is column-oriented. However, in SQL, each column can have only one version. Therefore, we need to transform column-oriented versioning to row-oriented versioning.

Two system column names are introduced to define timestamps and version numbers.

`_l_ts_` : specifies timestamps in milliseconds. `_l_versions_` : specifies version numbers. It is used to specify the number of versions to be selected.

The following are some examples:

```
-- Create a multi-version table. A maximum of five versions are reserved.
create table dt (p1 integer, p2 integer, c1 integer, c2 integer,
  constraint pk primary key(p1, p2 desc)) 'VERSIONS'='5';
-- Label the data written into a table with a specified timestamp in milliseconds.
-- Set the timestamps of both columns c1 and c2 to 15367736217189.
upsert into dt (p1,p2,c1,c2, _l_ts_) values (1,2, 10, 20, 15367736217189);
-- Delete all data versions with a timestamp earlier than or equal to 15367736217189 in a specified row.
delete from dt where p1 = 1 and p2 = 2 and _l_ts_ = 15367736217189;
```

Query examples:

```
-- Create a table that does not support multi-versioning. This means that the table has only one version by default.
create table dt (p1 integer, p2 integer, c1 integer, c2 integer,
  constraint pk primary key(p1, p2 desc)) 'VERSIONS'='5';
-- Label the data written into a table with a specified timestamp. Each non-primary key column has four versions.
-- In this example, timestamps 100 and 200 are used for you to distinguish the difference.
upsert into dt (p1,p2,c1,c2, _l_ts_) values (1,2, 10, 20, 100);
upsert into dt (p1,p2,c1,c2, _l_ts_) values (1,2, 30, 40, 200);
upsert into dt (p1,p2,c1,c2, _l_ts_) values (1,2, 50, 60, 300);
upsert into dt (p1,p2,c1,c2, _l_ts_) values (1,2, 70, 80, 400);
-- Label the data written into a table with a specified timestamp. Data inserted into different columns are labeled with different timestamps.
upsert into dt (p1,p2,c1,c2, _l_ts_) values (3,4, 70, 80, 500);
upsert into dt (p1,p2,c1, _l_ts_) values (3,4, 90, 600);
upsert into dt (p1,p2, c2, _l_ts_) values (3,4, 90, 700);
-- Query data of all versions in all rows of the table dt.
select * from dt where _l_versions_ = 5
```

The following result is returned:

```
0: jdbc:lindorm: select * from dt where _l_versions_ = 5;
```

| p1 | p2 | c1   | c2   | c1_l_ts | c2_l_ts |
|----|----|------|------|---------|---------|
| 1  | 2  | 10   | 20   | 100     | 100     |
| 1  | 2  | 30   | 40   | 200     | 200     |
| 1  | 2  | 50   | 60   | 300     | 300     |
| 1  | 2  | 70   | 80   | 400     | 400     |
| 3  | 4  | 70   | 80   | 500     | 500     |
| 3  | 4  | 90   | null | 600     | null    |
| 3  | 4  | null | 90   | null    | 700     |

```
7 rows selected (0.006 seconds)
```

null

- Each non-primary key column has a timestamp. The column names are `c1_l_ts` and `c2_l_ts`, which are a combination of the original column name and `_l_ts`.
- All non-primary key values in a row must be labeled with the same timestamp. Otherwise, they are dispersed to different rows based on their timestamps. Empty columns are imputed with nulls.

-- Each column can have no more than two versions.

```
select * from dt where _l_versions_ = 2;
```

```
0: jdbc:lindorm: select * from dt where _l_versions_ = 2;
```

| p1 | p2 | c1   | c2   | c1_l_ts | c2_l_ts |
|----|----|------|------|---------|---------|
| 1  | 2  | 50   | 60   | 300     | 300     |
| 1  | 2  | 70   | 80   | 400     | 400     |
| 3  | 4  | 70   | 80   | 500     | 500     |
| 3  | 4  | 90   | null | 600     | null    |
| 3  | 4  | null | 90   | null    | 700     |

```
5 rows selected (0.006 seconds)
```

null

-- Query all versions with a timestamp in the range of [200, 500].

```
select * from dt where p1 = 1 and p2 = 2 and _l_ts_ >= 200 and _l_ts_ < 500
and _l_versions_ = 5;
```

```
0: jdbc:lindorm: select * from dt where p1 = 1 and p2 = 2 and _l_ts_ >= 200 and _l_ts_ < 500 and _l_versions_ = 5;
```

| p1 | p2 | c1 | c2 | c1_l_ts | c2_l_ts |
|----|----|----|----|---------|---------|
| 1  | 2  | 30 | 40 | 200     | 200     |
| 1  | 2  | 50 | 60 | 300     | 300     |
| 1  | 2  | 70 | 80 | 400     | 400     |

```
3 rows selected (0.005 seconds)
```

null

-- Query the latest version with a timestamp in the range of [200, 500].

```
select * from dt where p1 = 1 and p2 = 2 and _l_ts_ >= 200 and _l_ts_ < 500;
```



```
0: jdbc:lindorm: select * from dt where p1 = 1 and p2 = 2 and _l_ts_ >= 200 and _l_ts_ < 500;
+-----+-----+-----+-----+-----+-----+
|      p1      |      p2      |      c1      |      c2      |      c1_l_ts      |      c2_l_ts      |
+-----+-----+-----+-----+-----+-----+
|      1      |      2      |      70      |      80      |      400      |      400      |
+-----+-----+-----+-----+-----+-----+
1 row selected (0.006 seconds)
```

null By default, only the latest version is retrieved. If you want to retrieve more than one version, you must set the value of `_l_versions_`.

Note: Multi-versioning requires additional storage space and incurs additional runtime overheads. We recommend that you confirm the business scenario before you use multi-versioning.

## Dynamic columns

Schema-free is a powerful feature of HBase. It allows you to define different columns for rows to be inserted. With schema-free, you do not need to predefine a schema or alter the table after the schema is updated. Schemas are required subject to SQL standards. Therefore, we need to add some constraints for SQL to support schema-free.

When you create a table, you must set `'DYNAMIC_COLUMNS' = 'true'` for the table to support dynamic columns. Example:

```
create table dt (p1 integer, p2 integer, c1 integer, c2 integer,
constraint pk primary key(p1, p2 desc)) 'DYNAMIC_COLUMNS' = 'true';
```

When you use a dynamic column, you must specify the data type of the column in the SQL statement because dynamic columns are not defined in the schema. Phoenix uses this solution. ApsaraDB for HBase uses another solution as a compromise: dynamic columns must be `varbinary` type. In this way, the SQL statements using dynamic columns are the same as those not using dynamic columns. Example:

```
upsert into dt(p1, p2, d1, d2) values(1, 2, '5A6B77FF', '12345');
-- Only four columns are defined in the schema of table dt: p1, p2, c1, and c2. Two dynamic columns, d1 and d
2, are inserted into the table, and hexadecimal values are inserted into the dynamic columns.
-- No constraint is set for queries.
select * from dt;
select d1 from dt;
```

Note: Dynamic columns are non-primary key columns. Currently, you are not allowed to modify primary key columns after you create a table.

## 7.7. Considerations and limits

The case sensitivity of SQL statements in ApsaraDB for HBase Performance-enhanced Edition is similar to MySQL. The rules are as follows:

- All system keywords are case insensitive. For example, `SELECT` is equivalent to `select` and `Select`.
- Column names are case insensitive. Other identifiers are case sensitive. Example:
  - `create schema my_schema;` `create schema My_Schema;` The two statements create two different schemas.

- create table myTable1 (...); create table MYTABLE1 (...); The two statements create two different tables.
- create table dt (pk integer, PK integer, ...); The two column names in this statement are duplicate because column names are case insensitive.

## Data limits

- The size of a primary key column must not exceed 2 KB.
- The total size of all primary key columns must not exceed 30 KB.
- The size of a non-primary key column must not exceed 2 MB.