CloudTable Service

Developer Guide

Issue 01

Date 2025-08-08

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 HBase Application Development Guide

1.1 HBase Application Development Process

This section describes how to call open source APIs of HBase in the CloudTable cluster mode to develop Java applications.

Figure 1-1 and **Table 1-1** describe the phases in the development process.

Figure 1-1 Application development process

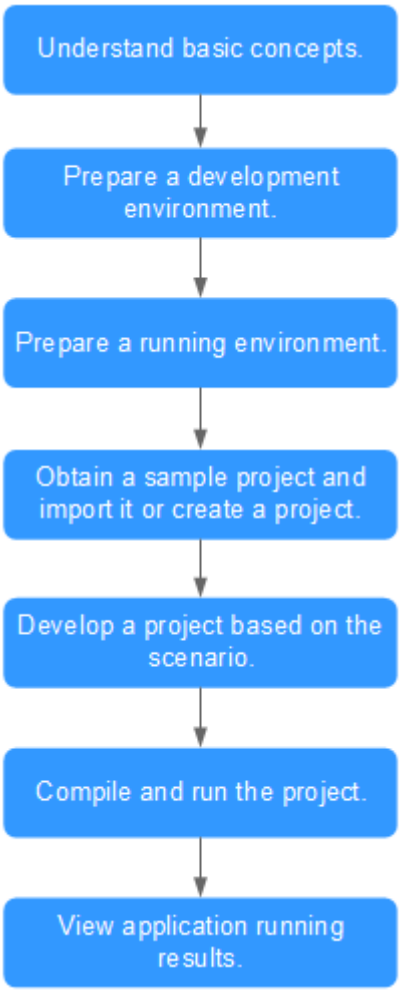


Table 1-1 Application development process details

Phase	Description	Reference
Understand basic concepts.	Before application development, learn basic concepts of HBase, understand the scenario requirements, and design tables.	**HBase**
Prepare a development environment.	The Java language is recommended for HBase application development. You can use the Eclipse tool.	**Preparing a Local Application Development Environment**
Prepare a running environment.	The application running environment is a client. Install and configure the client according to the guide.	**Preparing an HBase Application Running Environment**

Phase	Description	Reference
Prepare a project.	CloudTable provides example projects for different scenarios. You can import an example project to learn the application. You can also create a project according to the guide.	**Downloading an HBase Sample Project** **Importing and Configuring HBase Sample Projects**
Develop a project based on the scenario.	An example project using Java is provided, including creating a table, writing data into the table, and deleting the table.	**Developing HBase Applications**
Compile and run the application.	You can compile the developed application and submit it for running.	**Compiling and Running Applications** **Compiling and Running an Application When a Client Is Installed** or **Compiling and Running an Application When No Client Is Installed**
View application running results.	Application running results are exported to a path you specify. You can also view the application running status on the UI.	• In Windows: **Viewing Commissioning Results** • In Linux: **Viewing Commissioning Results**

1.2 Preparing an HBase Application Development Environment

1.2.1 Preparing a Local Application Development Environment

Table 1-2 describes the environment required for secondary development.

Table 1-2 Development environment

Item	Description
OS	Windows OS. Windows 7 or later is recommended.

Item	Description
JDK installation	Basic configurations of the development environment. JDK 1.7 or 1.8 is required. You are recommended to use JDK 1.8 for better compatibility of later versions. **NOTE** For security purpose, CloudTable supports only TLS 1.1 and TLS 1.2 encryption protocols. IBM JDK supports only 1.0 by default. If you use IBM JDK, set **com.ibm.jsse2.overrideDefaultTLS** to **true**. After the parameter setting, TLS1.0/1.1/1.2 can be supported at the same time. For details, see the related instructions on the IBM official website.
Eclipse installation and configuration	It is a tool used to develop CloudTable applications.
Network	Ensure that the development environment or client can communicate with the network of the CloudTable server.

1.2.2 Preparing an HBase Application Running Environment

Scenario

The running environment for CloudTable application development can be deployed on Windows. You can perform the following operations to prepare the running environment.

Procedure

Step 1 Check whether a CloudTable cluster is installed and run properly.

Step 2 Prepare a Windows ECS.

For details about how to prepare an ECS, see **Preparing an ECS**.

Step 3 Install JDK 1.7 or later on the Windows ECS. However, you are recommended to use JDK 1.8 or later and install Eclipse that uses JDK 1.7 or later.

NOTE

- If you use IBM JDK, ensure that the JDK configured in Eclipse is IBM JDK.
- If you use Oracle JDK, ensure that the JDK configured in Eclipse is Oracle JDK.
- Do not use the same workspace and the sample project in the same path for different Eclipse programs.

----End

1.2.3 Downloading an HBase Sample Project

Prerequisites

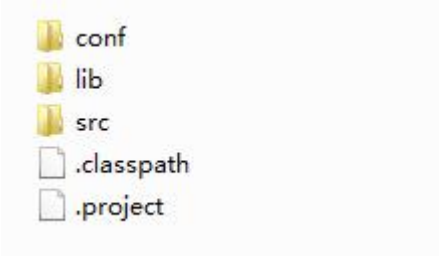
Ensure that CloudTable has been installed and is running properly.

Downloading a Sample Project

Step 1 Download the **Sample Code** project.

Step 2 After the download is complete, decompress the installation package of the sample code project to a local directory to obtain an Eclipse Java project. **Figure 1-2** shows the directory structure of the sample code project.

Figure 1-2 Directory structure of the sample code project



----End

Apache Maven Configuration

The sample project contains the HBase client JAR package. You can replace the JAR package with an open source HBase JAR package to access CloudTable. Open source HBase APIs later than 1.*X.X* are supported. If you need to import CloudTable's HBase JAR package to an application, configure the following dependencies in Maven:

```
<dependencies>
   <dependency>
      <groupId>org.apache.hbase</groupId>
      <artifactId>hbase-client</artifactId>
      <version>1.3.1.0305-cloudtable</version>
   </dependency>
   <dependency>
      <groupId>org.apache.hbase</groupId>
      <artifactId>hbase-common</artifactId>
      <version>1.3.1.0305-cloudtable</version>
   </dependency>
</dependencies>
```

Use either of the following methods to configure the address of the mirror warehouse.

- **Configuration method 1**

  Add the address of the open source mirror warehouse to the mirrors in **setting.xml**.

  ```
  <mirror>
     <id>repo2</id>
     <mirrorOf>central</mirrorOf>
     <url>https://repo1.maven.org/maven2/</url>
  </mirror>
  ```

  Add the following mirror warehouse address to the profiles in **setting.xml**.

  ```
  <profile>
     <id>xxxcloudsdk</id>
     <repositories>
        <repository>
           <id>xxxcloudsdk</id>
           <url>https://repo.xxxcloud.com/repository/maven/xxxcloudsdk/</url>
  ```



```
        <releases><enabled>true</enabled></releases>
        <snapshots><enabled>true</enabled></snapshots>
      </repository>
    </repositories>
</profile>
```

Add the following mirror warehouse address to the activeProfiles in **setting.xml**.

```
<activeProfile>xxxcloudsdk</activeProfile>
```

NOTE

The Huawei Cloud open source mirror center does not provide third-party open source JAR files. After configuring Huawei Cloud open source mirrors, you need to separately configure third-party Maven image repository address.

- **Configuration method 2**

  Add the following mirror warehouse address to the **pom.xml** file in the secondary development sample project.

```
  <repositories>
    <repository>
      <id>xxxcloudsdk</id>
      <url>https://mirrors.xxxcloud.com/repository/maven/xxxcloudsdk/</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </repository>

    <repository>
      <id>central</id>
      <name>Mavn Centreal</name>
      <url>https://repo1.maven.org/maven2/</url>
    </repository>
  </repositories>
```

1.2.4 Importing and Configuring HBase Sample Projects

Background Information

After importing the CloudTable sample code project to Eclipse, you can start learning CloudTable application development samples.

Prerequisites

You have correctly configured the running environment. For details about how to configure the running environment, see **Preparing an HBase Application Running Environment**.

Procedure

Step 1 Import the sample project to the Windows development environment. For details about how to obtain the sample project, see **Downloading an HBase Sample Project**.

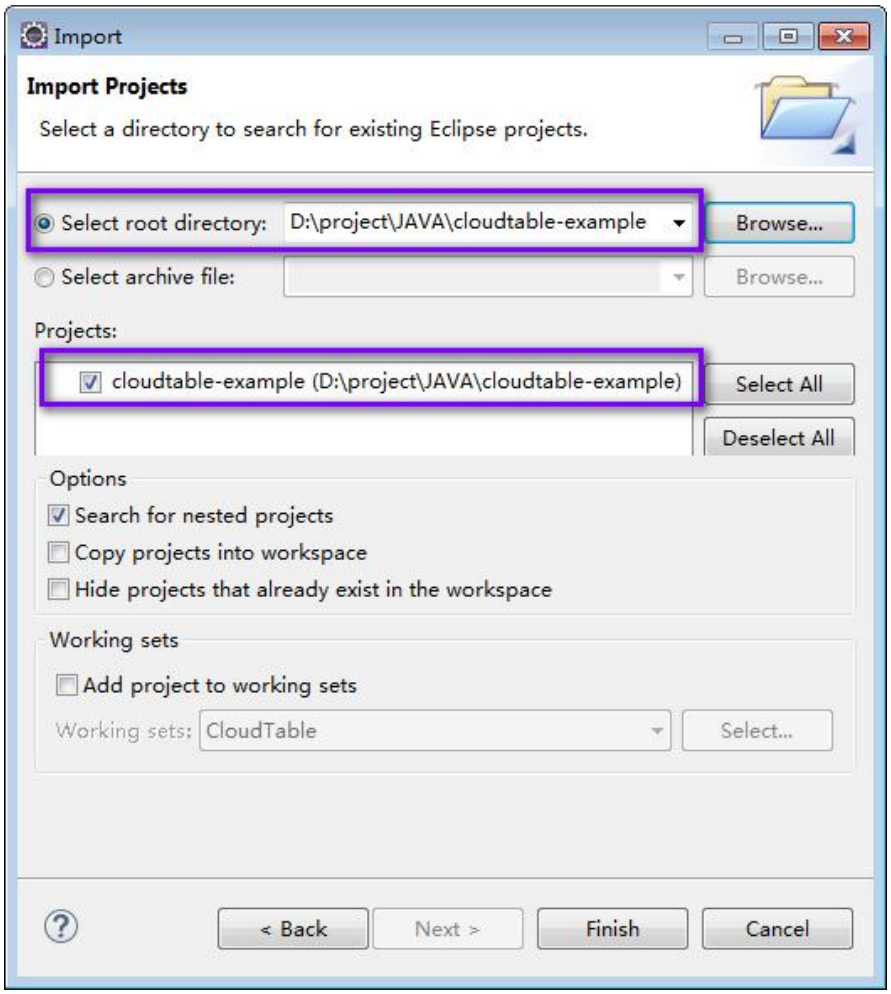
Step 2 In the application development environment, import the sample project to the Eclipse development environment.

1. Choose **File** > **Import** > **General** > **Existing Projects into Workspace** > **Next** > **Browse**.

   The **Browse Folder** dialog box is displayed, as shown in **Figure 1-3**.

2. Select the sample project folder, and click **Finish**.

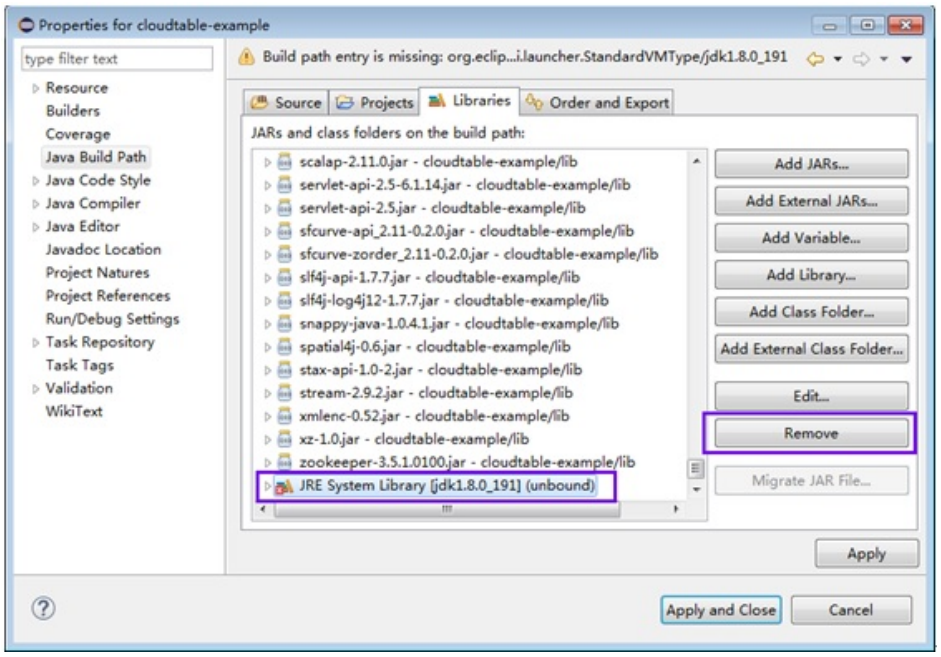
Figure 1-3 Importing a sample project



Step 3 Right-click the **cloudtable-example** project, and choose **Properties** from the shortcut menu. The **Properties for cloudtable-example** window is displayed.

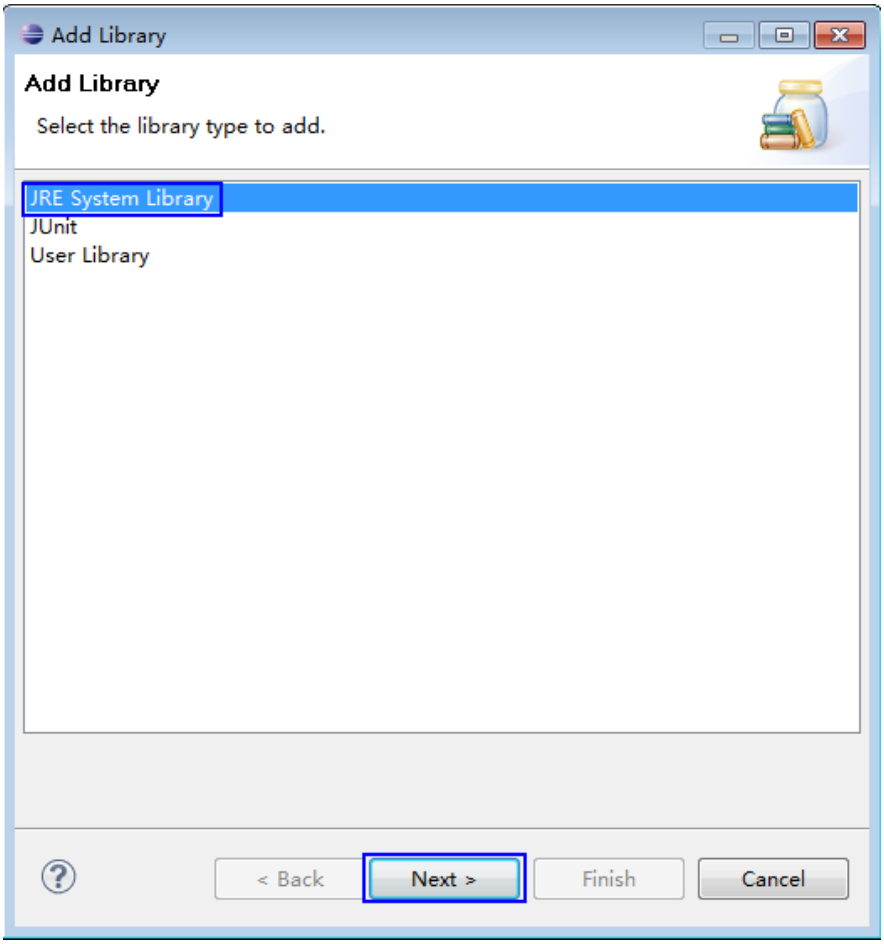
1. In the navigation tree, select **Java Build Path**. Click the **Libraries** tab, select all error JDKs, and click **Remove**, as shown in **Figure 1-4**.

Figure 1-4 Delete error JDKs



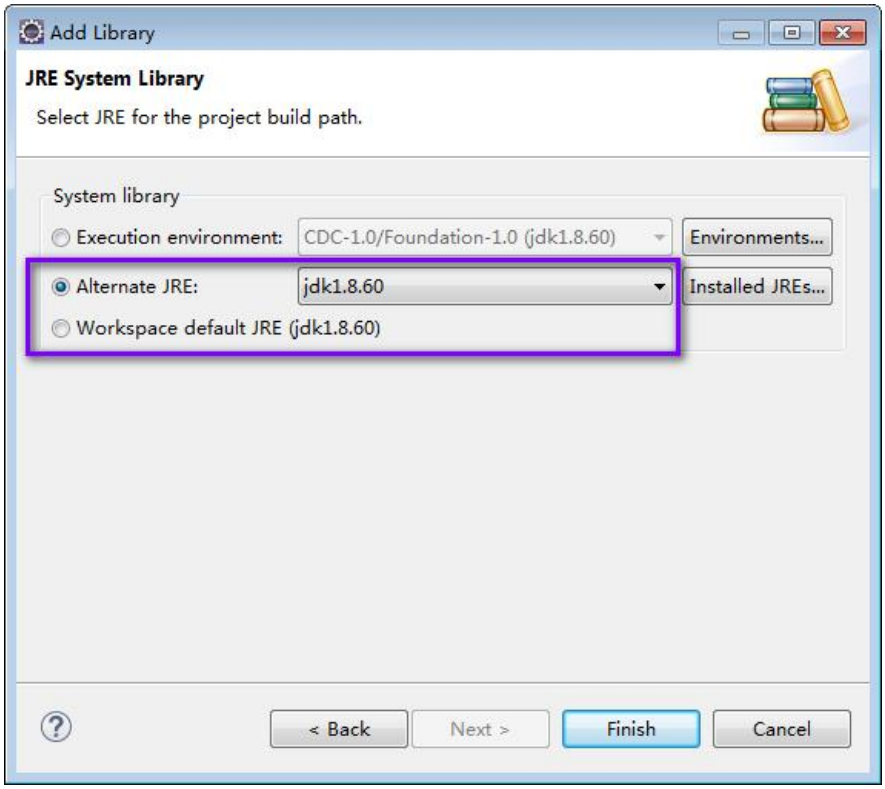
2. Click **Add Library...** shown in **Figure 1-5**. Select **JRE System Library** in the pop-up window.

Figure 1-5 Adding libraries



3. In the **Add Library** dialog box, select a JDK version from the drop-down list of **Alternate JRE** or **Workspace default JRE**. Select **Alternate JRE** and select the JDK version, as shown in **Figure 1-6**.

Figure 1-6 Selecting JRE



4. Click **Finish** to complete configuration and close the window.

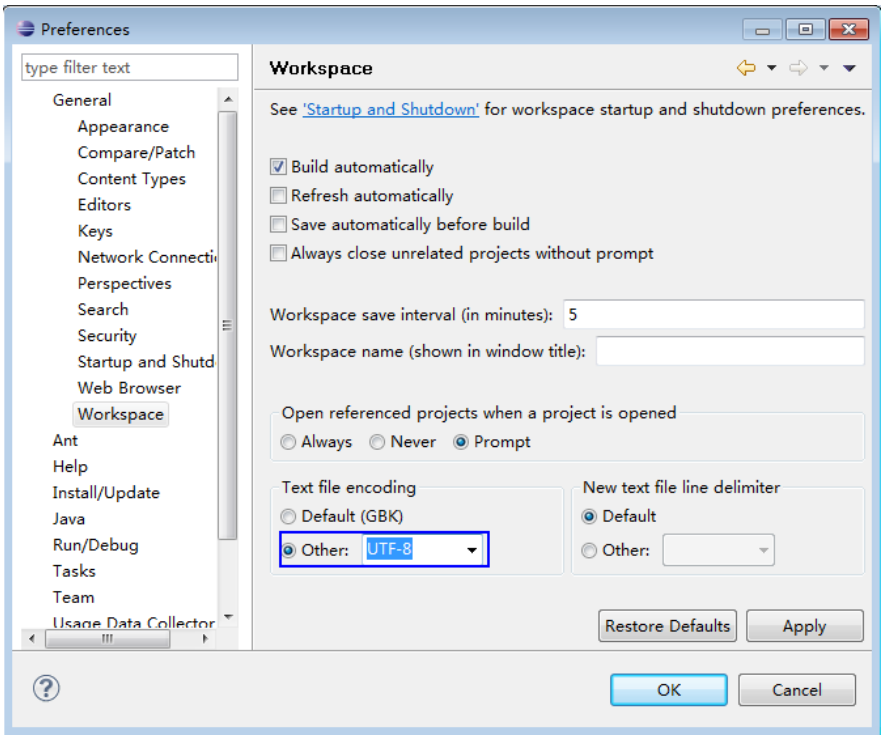
Step 4 Set the Eclipse text file coding format to prevent garbled characters.

1. On the Eclipse menu bar, choose **Window** > **Preferences**.

   The **Preferences** window is displayed.

2. In the navigation tree, choose **General** > **Workspace**. In the **Text file encoding** area, select **Other** and set the value to **UTF-8**. Click **Apply** and then **OK**. **Figure 1-7** shows the settings.

Figure 1-7 Setting the Eclipse encoding format



Step 5 Open the **conf/hbase-site.xml** file in the sample project and change the value of **hbase.zookeeper.quorum** to the correct ZooKeeper address.

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

value is the domain name of the ZooKeeper cluster. Log in to the CloudTable console and choose **Cluster Management**. In the cluster list, locate the required cluster and obtain the address in the **Access Address (Intranet)** column.

----End

1.3 Developing HBase Applications

1.3.1 HBase Data Read/Write Sample Project

1.3.1.1 Typical Scenarios of the HBase Data Read/Write Sample Program

You can quickly learn and master the HBase development process and know key interface functions in a typical application scenario.

Description

Develop an application to manage information about users who use service A in an enterprise. **Table 1-3** provides the user information. Procedures are as follows:

- Create a user information table.

- Add users' educational backgrounds and titles to the table.
- Query user names and addresses by user ID.
- Query information by user name.
- Query information about users whose age ranges from 20 to 29.
- Collect the number of users and their maximum, minimum, and average age.
- Deregister users and delete user data from the user information table.
- Delete the user information table after service A ends.

Table 1-3 User information

ID	Name	Gender	Age	Address
12005000201	A	Male	19	IPA, IPB
12005000202	B	Female	23	IPC, IPD
12005000203	C	Male	26	IPE, IPF
12005000204	D	Male	18	IPG, IPH
12005000205	E	Female	21	IPI, IPJ
12005000206	F	Male	32	IPK, IPL
12005000207	G	Female	29	IPM, IPN
12005000208	H	Female	30	IPO, IPP
12005000209	I	Male	26	IPQ, IPR
12005000210	J	Male	25	IPS, IPT

Data Planning

Proper design of a table structure, RowKeys, and column names enable you to make full use of HBase advantages. In the sample project, a unique ID is used as a RowKey, and columns are stored in the **info** column family.

1.3.1.2 HBase Data Read/Write Sample Program Development Plan

Function Description

Determine functions to be developed based on the preceding scenario. **Table 1-4** describes functions to be developed.

Table 1-4 Functions to be developed in HBase

No.	Procedure	Code Implementation
1	Create a table based on the information in **Typical Scenarios of the HBase Data Read/ Write Sample Program**.	For details, see **Creating an HBase Table**.
2	Import user data.	For details, see **Inserting HBase Data**.
3	Add an educational background column family, and add educational backgrounds and titles to the user information table.	For details, see **Modifying an HBase Table**.
4	Query user names and addresses by user ID.	For details, see **Reading HBase Data Using the GET Command**.
5	Query information by user name.	For details, see **Using an HBase Filter**.
6	Deregister users and delete user data from the user information table.	For details, see **Deleting HBase Data**.
7	Delete the user information table after service A ends.	For details, see **Deleting an HBase Table**.

Key Design Principles

HBase is a distributed database system based on the lexicographic order of RowKeys. The RowKey design has great impact on performance, so the RowKeys must be designed based on specific services.

1.3.1.3 Configuring the HBase ZooKeeper Address

Before executing sample code, configure the correct ZooKeeper cluster address in the **hbase-site.xml** configuration file.

The configuration items are as follows:

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

value is the domain name of the ZooKeeper cluster. Log in to the CloudTable console and choose **Cluster Management**. In the cluster list, locate the required cluster and obtain the address in the **Access Address (Intranet)** column.

1.3.1.4 Initializing HBase Configurations

Function Description

HBase obtains configuration items by loading a configuration file.

NOTE

1. Loading the configuration file is time-consuming. If unnecessary, use the same Configuration object.
2. Multi-thread synchronization is not considered in the sample code. If necessary, add it by yourself. Other sample codes are the same.

Sample Code

The following code snippets are in the **com.huawei.cloudtable.hbase.examples** packet.

```
private static void init() throws IOException {
  // Default load from conf directory
  conf = HBaseConfiguration.create(); // Note [1]
  String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
  Path hbaseSite = new Path(userdir + "hbase-site.xml");
  if (new File(hbaseSite.toString()).exists()) {
    conf.addResource(hbaseSite);
  }
}
```

Note [1] If the **conf** directory of the configuration file is added to the **classpath** path, the code for loading the specified configuration file can be skipped.

1.3.1.5 Creating an HBase Client Connection

Function Description

HBase creates a Connection object using the **ConnectionFactory.createConnection(configuration)** method. The transferred parameter is the Configuration created in the last step.

Connection encapsulates the connections between underlying applications and servers and ZooKeeper. Connection is instantiated using the **ConnectionFactory** class. Creating Connection is a heavyweight operation. Connection is thread-safe. Therefore, multiple client threads can share one Connection.

In a typical scenario, a client program uses a Connection, and each thread obtains its own Admin or Table instance and invokes the operation interface provided by the Admin or Table object. You are not advised to cache or pool Table and Admin. The lifecycle of Connection is maintained by invokers who can release resources by invoking **close()**.

NOTE

When the service code is connected to the same CloudTable cluster, you are advised to create one Connection and reuse it for multiple threads. You do not need to create a Connection for every thread. Connection is a connector for connecting to a CloudTable cluster. Excessive Connections will increase loads on ZooKeeper and deteriorate service read/write performance.

Sample Code

The following code snippet is an example of creating a Connection object:

```
private TableName tableName = null;
private Connection conn = null;

public HBaseSample(Configuration conf) throws IOException {
  this.tableName = TableName.valueOf("hbase_sample_table");
  this.conn = ConnectionFactory.createConnection(conf);
}
```

1.3.1.6 Creating an HBase Table

Function Description

In HBase, a table is created using the **createTable** method of the **org.apache.hadoop.hbase.client.Admin** object. You need to specify a table name and a column family name. You can create a table by using either of the following methods, but the latter one is recommended:

- Quickly create a table. A newly created table contains only one region, which will be automatically split into multiple new regions as data increases.
- Create a table using pre-assigned regions. You can pre-assign multiple regions before creating a table. This mode accelerates data write at the beginning of massive data write.

NOTE

The table name and column family name of a table consist of letters, digits, and underscores (_) but cannot contain any special characters.

Sample Code

```
public void testCreateTable() {
  LOG.info("Entering testCreateTable.");

  // Specify the table descriptor.
  HTableDescriptor htd = new HTableDescriptor(tableName); // (1)

  // Set the column family name to info.
  HColumnDescriptor hcd = new HColumnDescriptor("info"); // (2)

  // Set data encoding methods. HBase provides DIFF,FAST_DIFF,PREFIX
  // and PREFIX_TREE
  hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);  // Note [1]

  // Set compression methods, HBase provides two default compression
  // methods:GZ and SNAPPY
  // GZ has the highest compression rate,but low compression and
  // decompression efficiency,fit for cold data
  // SNAPPY has low compression rate, but high compression and
  // decompression efficiency,fit for hot data.
  // it is advised to use SANPPY
  hcd.setCompressionType(Compression.Algorithm.SNAPPY);
  htd.addFamily(hcd); // (3)

  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin(); // (4)
    if (!admin.tableExists(tableName)) {
      LOG.info("Creating table...");
```



```
      admin.createTable(htd); // Note [2] (5)
      LOG.info(admin.getClusterStatus());
      LOG.info(admin.listNamespaceDescriptors());
      LOG.info("Table created successfully.");
    } else {
      LOG.warn("table already exists");
    }
  } catch (IOException e) {
    LOG.error("Create table failed.", e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting testCreateTable.");
}
```

Explanation

(1) Create a table descriptor.

(2) Create a column family descriptor.

(3) Add the column family descriptor to the table descriptor.

(4) Obtain the Admin object. You use the Admin object to create a table and a column family, check whether the table exists, modify the table structure and column family structure, and delete the table.

(5) Invoke the Admin object to create a table.

Precautions

- Note [1] Use the following code to set the compression mode for a column family:

  ```
  // Set an encoding algorithm. HBase provides four encoding algorithms: DIFF, FAST_DIFF, PREFIX, and PREFIX_TREE.
   hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
  // Set a file compression mode. By default, HBase provides two compression algorithms: GZ and SNAPPY.
  // GZ has a high compression rate but low compression and decompression performance. It is applicable to cold data.
  // SNAPPY has a low compression rate but high compression and decompression performance. It is applicable to hot data.
  // It is recommended that SNAPPY compression be enabled by default.
   hcd.setCompressionType(Compression.Algorithm.SNAPPY);
  ```

- Note [2] Create a table by specifying the start and end RowKeys or pre-assigning regions using RowKey arrays. The code snippet is as follows:

  ```
  // Create a table with pre-split regions.
   byte[][] splits = new byte[4][];
   splits[0] = Bytes.toBytes("A");
   splits[1] = Bytes.toBytes("H");
   splits[2] = Bytes.toBytes("O");
   splits[3] = Bytes.toBytes("U");
   admin.createTable(htd, splits);
  ```


1.3.1.7 Deleting an HBase Table

Function Description

In HBase a table is deleted using the **deleteTable** method of **org.apache.hadoop.hbase.client.Admin**.

Sample Code

```
public void dropTable() {
  LOG.info("Entering dropTable.");
  Admin admin = null;
  try {
    admin = conn.getAdmin();
    if (admin.tableExists(tableName)) {
      // Disable the table before deleting it.
      admin.disableTable(tableName);
      // Delete table.
      admin.deleteTable(tableName);//Note [1]
    }
    LOG.info("Drop table successfully.");
  } catch (IOException e) {
    LOG.error("Drop table failed " ,e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
      }
    }
  }
  LOG.info("Exiting dropTable.");
}
```

Precautions

Note [1] Only after the **disableTable** API is called, the table can be deleted by calling the **deleteTable** API. Therefore, **deleteTable** is often used together with **disableTable**, **enableTable**, **tableExists**, **isTableEnabled**, and **isTableDisabled**.

1.3.1.8 Modifying an HBase Table

Function Description

In HBase, table information is modified using the **modifyTable** method of **org.apache.hadoop.hbase.client.Admin**.

Sample Code

```
public void testModifyTable() {
  LOG.info("Entering testModifyTable.");

  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("education");
  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin();
    // Obtain the table descriptor.
    HTableDescriptor htd = admin.getTableDescriptor(tableName);
```



```
    // Check whether the column family is specified before modification.
    if (!htd.hasFamily(familyName)) {
      // Create the column descriptor.
      HColumnDescriptor hcd = new HColumnDescriptor(familyName);
      htd.addFamily(hcd);
      // Disable the table to get the table offline before modifying
      // the table.
      admin.disableTable(tableName);
      // Submit a modifyTable request.
      admin.modifyTable(tableName, htd);  //Note [1]
      // Enable the table to get the table online after modifying the
      // table.
      admin.enableTable(tableName);
    }
    LOG.info("Modify table successfully.");
  } catch (IOException e) {
    LOG.error("Modify table failed " ,e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
      }
    }
  }
  LOG.info("Exiting testModifyTable.");
}
```

Precautions

Note [1] Only after the **disableTable** API is called, the table can be modified by calling the **modifyTable** API. Then, call the **enableTable** API to enable the table again.

1.3.1.9 Inserting HBase Data

Function Description

HBase is a column-based database. A row of data may have multiple column families, and a column family may contain multiple columns.

When writing data, you must specify the columns (including the column family names and column names) to which data is written.

In HBase, data (a row of data or data sets) is inserted using the **put** method of HTable.

Sample Code

```
public void testPut() {
  LOG.info("Entering testPut.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifiers = { Bytes.toBytes("name"), Bytes.toBytes("gender"),
      Bytes.toBytes("age"), Bytes.toBytes("address") };
  Table table = null;
  try {
    // Instantiate an HTable object.
    table = conn.getTable(tableName);
    List<Put> puts = new ArrayList<Put>();
    // Instantiate a Put object.
```



```
  Put put = new Put(Bytes.toBytes("012005000201"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("A"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("19"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPA, IPB"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000202"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("B"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("23"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPC, IPD"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000203"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("C"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPE, IPF"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000204"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("D"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("18"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPG, IPH"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000205"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("E"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("21"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPI, IPJ"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000206"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("F"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("32"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPK, IPL"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000207"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("G"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("29"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPM, IPN"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000208"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("H"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("30"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPO, IPP"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000209"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("I"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPQ, IPR"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000210"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("J"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("25"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("IPS, IPT"));
  puts.add(put);
  // Submit a put request.
  table.put(puts);

  LOG.info("Put successfully.");
} catch (IOException e) {
  LOG.error("Put failed " ,e);
} finally {
  if (table != null) {
    try {
```



```
      // Close the HTable object.
      table.close();
    } catch (IOException e) {
      LOG.error("Close table failed " ,e);
    }
  }
 }
 LOG.info("Exiting testPut.");
}
```

Precautions

Multiple threads are not allowed to use the same HTable instance at the same time. HTable is a non-thread-safe class. If an HTable instance is used by multiple threads at the same time, exceptions will occur.

1.3.1.10 Deleting HBase Data

Function Description

In HBase, data (a row of data or data sets) is deleted using the **delete** method of a Table instance.

The specified deletion method is based on the scenario.

Sample Code

```
public void testDelete() {
 LOG.info("Entering testDelete.");

 byte[] rowKey = Bytes.toBytes("012005000201");

 Table table = null;
 try {
  // Instantiate an HTable object.
  table = conn.getTable(tableName);

  // Instantiate a Delete object.
  Delete delete = new Delete(rowKey);

  // Submit a delete request.
  table.delete(delete);

  LOG.info("Delete table successfully.");
 } catch (IOException e) {
  LOG.error("Delete table failed " ,e);
 } finally {
  if (table != null) {
   try {
    // Close the HTable object.
    table.close();
   } catch (IOException e) {
    LOG.error("Close table failed " ,e);
   }
  }
 }
 LOG.info("Exiting testDelete.");
}
```


1.3.1.11 Reading HBase Data Using the GET Command

Function Description

Before reading data from a table, create a table instance and a Get object.

You can also set parameters for the Get object, such as the column family name and column name.

Query results are stored in the Result object that stores multiple Cells.

Sample Code

```
public void testGet() {
  LOG.info("Entering testGet.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifier = { Bytes.toBytes("name"), Bytes.toBytes("address") };
  // Specify RowKey.
  byte[] rowKey = Bytes.toBytes("012005000201");
  Table table = null;
  try {
    // Create the Table instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Get get = new Get(rowKey);
    // Set the column family name and column name.
    get.addColumn(familyName, qualifier[0]);
    get.addColumn(familyName, qualifier[1]);
    // Submit a get request.
    Result result = table.get(get);
    // Print query results.
    for (Cell cell : result.rawCells()) {
      LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
          + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
          + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
          + Bytes.toString(CellUtil.cloneValue(cell)));
    }
    LOG.info("Get data successfully.");
  } catch (IOException e) {
    LOG.error("Get data failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testGet.");
}
```

1.3.1.12 Reading HBase Data Using the Scan Command

Function Description

Before reading data from a table, instantiate the Table instance of the table, and then create a Scan object and set parameters for the Scan object based on search criteria. To improve query efficiency, you are advised to specify StartRow and StopRow. Query results are stored in the ResultScanner object, where each row of data is stored as a Result object that stores multiple Cells.

Sample Code

```
public void testScanData() {
  LOG.info("Entering testScanData.");
  Table table = null;
  // Instantiate a ResultScanner object.
  ResultScanner rScanner = null;
  try {
    // Create the Configuration instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
    // Set the cache size.
    scan.setCaching(1000);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Scan data successfully.");
  } catch (IOException e) {
    LOG.error("Scan data failed " ,e);
  } finally {
    if (rScanner != null) {
      // Close the scanner object.
      rScanner.close();
    }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testScanData.");
}
```

Precautions

1. You are advised to specify StartRow and StopRow to ensure good performance with a specified Scan scope.
2. You can set **Batch** and **Caching**.
   - Batch

     Batch indicates the maximum number of records returned each time when the **next** API is invoked using Scan. This parameter is related to the number of columns read each time.
   - Caching

     Caching indicates the maximum number of next records returned for a remote procedure call (RPC) request. This parameter is related to the number of rows read by an RPC.

1.3.1.13 Using an HBase Filter

Function Description

HBase Filter is used to filter data during Scan and Get. You can specify the filter criteria, such as filtering by RowKey, column name, or column value.

The specified filtering method is based on the scenario.

Sample Code

```
public void testSingleColumnValueFilter() {
  LOG.info("Entering testSingleColumnValueFilter.");
  Table table = null;
  ResultScanner rScanner = null;

  try {
    table = conn.getTable(tableName);
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
    // Set the filter criteria.
    SingleColumnValueFilter filter = new SingleColumnValueFilter(
        Bytes.toBytes("info"), Bytes.toBytes("name"), CompareOp.EQUAL,
        Bytes.toBytes("I"));
    scan.setFilter(filter);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Single column value filter successfully.");
  } catch (IOException e) {
    LOG.error("Single column value filter failed " ,e);
  } finally {
      if (rScanner != null) {
        // Close the scanner object.
        rScanner.close();
      }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testSingleColumnValueFilter.");
}
```

1.3.2 HBase Cold and Hot Separation Data Sample Project

1.3.2.1 Typical Scenarios of the HBase Cold and Hot Data Separation Sample Program

CloudTable HBase supports hot and cold data separation. With this feature, you can store hot and cold data in different types of storage media to reduce storage costs.

In a big data storage scenario, business data such as order data or monitoring data grows over time. As your business develops, such data can be of a large volume and rarely used. Companies may want to use cost-effective storage to store this type of data to reduce costs.

You can quickly learn and master the development process of HBase cold and hot data separation and know the functions of key APIs in a typical use case.

Scenario

Assume that a user develops an application to record and query weather information about a city in real time. The following table lists the recorded data.

Table 1-5 Raw data

City	District	Date	Temperature	Humidity
Shenzhen	Longgang	2017/7/1 00:00:00	28	54
Shenzhen	Longgang	2017/7/1 01:00:00	27	53
Shenzhen	Longgang	2017/7/1 02:00:00	27	52
Shenzhen	Longgang	2017/7/1 03:00:00	27	51
Shenzhen	Longgang	2017/7/1 04:00:00	27	50
Shenzhen	Longgang	2017/7/1 05:00:00	27	49
Shenzhen	Longgang	2017/7/1 06:00:00	27	48
Shenzhen	Longgang	2017/7/1 07:00:00	27	46
Shenzhen	Longgang	2017/7/1 08:00:00	29	46
Shenzhen	Longgang	2017/7/1 09:00:00	30	48
Shenzhen	Longgang	2017/7/1 10:00:00	32	48
Shenzhen	Longgang	2017/7/1 11:00:00	32	49
Shenzhen	Longgang	2017/7/1 12:00:00	33	49
Shenzhen	Longgang	2017/7/1 13:00:00	33	50
Shenzhen	Longgang	2017/7/1 14:00:00	32	50
Shenzhen	Longgang	2017/7/1 15:00:00	32	50
Shenzhen	Longgang	2017/7/1 16:00:00	31	51

City	District	Date	Temperature	Humidity
Shenzhen	Longgang	2017/7/1 17:00:00	30	51
Shenzhen	Longgang	2017/7/1 18:00:00	30	51
Shenzhen	Longgang	2017/7/1 19:00:00	29	51
Shenzhen	Longgang	2017/7/1 20:00:00	29	52
Shenzhen	Longgang	2017/7/1 21:00:00	29	53
Shenzhen	Longgang	2017/7/1 22:00:00	28	54
Shenzhen	Longgang	2017/7/1 23:00:00	28	54
Shenzhen	Longgang	2017/7/2 00:00:00	28	54
Shenzhen	Longgang	2017/7/2 01:00:00	27	53
Shenzhen	Longgang	2017/7/2 02:00:00	27	52
Shenzhen	Longgang	2017/7/2 03:00:00	27	51
Shenzhen	Longgang	2017/7/2 04:00:00	27	50
Shenzhen	Longgang	2017/7/2 05:00:00	27	49
Shenzhen	Longgang	2017/7/2 06:00:00	27	48
Shenzhen	Longgang	2017/7/2 07:00:00	27	46
Shenzhen	Longgang	2017/7/2 08:00:00	29	46
Shenzhen	Longgang	2017/7/2 09:00:00	30	48
Shenzhen	Longgang	2017/7/2 10:00:00	32	48
Shenzhen	Longgang	2017/7/2 11:00:00	32	49
Shenzhen	Longgang	2017/7/2 12:00:00	33	49
Shenzhen	Longgang	2017/7/2 13:00:00	33	50
Shenzhen	Longgang	2017/7/2 14:00:00	32	50
Shenzhen	Longgang	2017/7/2 15:00:00	32	50
Shenzhen	Longgang	2017/7/2 16:00:00	31	51
Shenzhen	Longgang	2017/7/2 17:00:00	30	51
Shenzhen	Longgang	2017/7/2 18:00:00	30	51
Shenzhen	Longgang	2017/7/2 19:00:00	29	51
Shenzhen	Longgang	2017/7/2 20:00:00	29	52
Shenzhen	Longgang	2017/7/2 21:00:00	29	53

City	District	Date	Temperature	Humidity
Shenzhen	Longgang	2017/7/2 22:00:00	28	54
Shenzhen	Longgang	2017/7/2 23:00:00	28	54

Data Planning

Proper design of a table structure, RowKeys, and column names enable you to make full use of HBase advantages. In this sample project, the city, region, and date are used as RowKeys, and columns are stored in the **info** column family.

Data is written on the hour of the current day. The data of the previous day is seldom accessed and is automatically archived to the cold storage to save storage space.

1.3.2.2 HBase Cold and Hot Data Separation Sample Program Development Plan

Function Description

Table 1-6 shows the implementation details of functions of HBase cold and hot data separation.

Table 1-6 Implementing functions of HBase cold and hot data separation

No.	Procedure	Code Implementation
1	Create a table based on the information in **Typical Scenarios of the HBase Cold and Hot Data Separation Sample Program**.	For details, see **Creating an HBase Cold and Hot Data Separation Table**.
2	Write data.	For details, see **Inserting HBase Cold and Hot Separation Data**.
4	Query the temperature and humidity by city, district, and date using the **GET** command.	For details, see **Reading HBase Cold and Hot Separation Data Using the GET Command**.
5	Perform a query by city, district, and date using the **SCAN**command.	For details, see **Reading HBase Cold and Hot Separation Data Using the Scan Command**.

Key Design Principles

Rows in HBase are sorted lexicographically by row key. This mechanism improves scan query performance if the row keys are properly designed. It is a good choice to design the row keys based on your service requirements.

1.3.2.3 Configuring the HBase ZooKeeper Address

Before executing sample code, configure the correct ZooKeeper cluster address in the **hbase-site.xml** configuration file. The configuration items are as follows:

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

value is the domain name of the ZooKeeper cluster. Log in to the CloudTable console and choose **Cluster Management**. In the cluster list, locate the required cluster and obtain its ZK link in the **ZK Link** column.

1.3.2.4 Creating the Configuration Object

Function Description

HBase obtains configuration items by loading a configuration file.

NOTE

- Loading the configuration file is time-consuming. It is recommended to use the same Configuration object.
- Multi-thread synchronization is not considered in the sample code. If necessary, add it by yourself. Other sample codes are the same.

Sample Code

The following code snippets are in the **com.huawei.cloudtable.hbase.examples.coldhotexample** packet.

```
private static void init() throws IOException {
  // Default load from conf directory
  conf = HBaseConfiguration.create(); // Note [1]
  String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
  Path hbaseSite = new Path(userdir + "hbase-site.xml");
  if (new File(hbaseSite.toString()).exists()) {
    conf.addResource(hbaseSite);
  }
}
```

Precautions

- Note [1] If the **conf** directory of the configuration file is added to the **classpath** path, the code for loading the specified configuration file can be skipped.

  NOTE

  Note [1] refers to **conf = HBaseConfiguration.create(); //Note [1]** in the sample code.

1.3.2.5 Creating the Connection Object

Function Description

HBase creates a Connection object using the **ConnectionFactory.createConnection(configuration)** method. The transferred parameter is the Configuration created in the last step.

Connection encapsulates the connections between underlying applications and servers and ZooKeeper. Connection is instantiated using the **ConnectionFactory** class. Creating Connection is a heavyweight operation. Connection is thread-safe. Therefore, multiple client threads can share one Connection.

A client program usually uses a Connection, and each thread obtains its own Admin or Table instance and invokes the operation interface provided by the Admin or Table object. You are not advised to cache or pool Table and Admin. The lifecycle of Connection is maintained by invokers who can release resources by invoking **close()**.

NOTE

When service code is connected to the same CloudTable cluster, you are advised to create one Connection and reuse it for multiple threads. You do not need to create a Connection for every thread. Connection is a connector for connecting to a CloudTable cluster. Excessive Connections will increase loads on ZooKeeper and deteriorate service read/write performance.

Sample Code

The following code snippet is an example of creating a Connection object:

```
private TableName tableName = null;
private Connection conn = null;

public HBaseSample(Configuration conf) throws IOException {
  this.tableName = TableName.valueOf("hbase_sample_table");
  this.conn = ConnectionFactory.createConnection(conf);
}
```

1.3.2.6 Creating an HBase Cold and Hot Data Separation Table

Function Description

HBase allows you to create a table using the **createTable** method of the **org.apache.hadoop.hbase.client.Admin** object. You need to specify a table name, a column family name, and the time boundary.

You can create a table by using either of the following methods, but the latter one is recommended:

- Quickly create a table. A newly created table contains only one region, which will be automatically split into multiple new regions as data increases.
- Create a table using pre-assigned regions. You can pre-assign multiple regions before creating a table. This mode accelerates data write at the beginning of massive data write.

> **NOTE**
>
> The table name and column family name of a table consist of letters, digits, and underscores (_) but cannot contain any special characters.

Sample Code

```
public void testCreateTable() {
  LOG.info("Entering testCreateTable.");

  // Specify the table descriptor.
  HTableDescriptor htd = new HTableDescriptor(tableName); // (1)

  // Set the column family name to info.
  HColumnDescriptor hcd = new HColumnDescriptor("info"); // (2)

  // Set hot and cold data boundary
  hcd.setValue(HColumnDescriptor.COLD_BOUNDARY, "86400");
  htd.addFamily(hcd); // (3)

  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin(); // (4)
    if (!admin.tableExists(tableName)) {
      LOG.info("Creating table...");
      admin.createTable(htd); // Note [1] (5)
      LOG.info(admin.getClusterStatus());
      LOG.info(admin.listNamespaceDescriptors());
      LOG.info("Table created successfully.");
    } else {
      LOG.warn("table already exists");
    }
  } catch (IOException e) {
    LOG.error("Create table failed.", e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting testCreateTable.");
}
```

- Description of code numbers
  - (1) Create a table descriptor.
  - (2) Create a column family descriptor.
  - (3) Add the column family descriptor to the table descriptor.
  - (4) Obtain the Admin object. You use the Admin object to create a table and a column family, check whether the table exists, modify the table structure and column family structure, and delete the table.
  - (5) Invoke the Admin object to create a table.
- Precautions
  - For details about how to set other attributes of the table and column family, see "Developing HBase Applications".

    > **NOTE**
    >
    > **Note [1]** refers to **admin.createTable(htd); // Note [1] (5)** in the sample code.

1.3.2.7 Deleting an HBase Cold and Hot Data Separation Table

Function Description

HBase allows you to delete a table using the **deleteTable** method of **org.apache.hadoop.hbase.client.Admin**.

Sample Code

```
public void dropTable() {
  LOG.info("Entering dropTable.");
  Admin admin = null;
  try {
    admin = conn.getAdmin();
    if (admin.tableExists(tableName)) {
      // Disable the table before deleting it.
      admin.disableTable(tableName);
      // Delete table.
      admin.deleteTable(tableName);//Note [1]
    }
    LOG.info("Drop table successfully.");
  } catch (IOException e) {
    LOG.error("Drop table failed " ,e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
      }
    }
  }
  LOG.info("Exiting dropTable.");
}
```

Precautions

Note [1] Only after the **disableTable** API is called, the table can be deleted by calling the **deleteTable** API.

Therefore, **deleteTable** is often used together with **disableTable**, **enableTable**, **tableExists**, **isTableEnabled**, and **isTableDisabled**.

NOTE

Note [1] refers to **admin.deleteTable(tableName);//Note[1]** in the sample code.

1.3.2.8 Modifying an HBase Cold and Hot Data Separation Table

Function Description

HBase allows you to modify table information using the **modifyTable** method of **org.apache.hadoop.hbase.client.Admin**.

Sample Code

- Disable the time boundary.

```
public void testModifyTable() {
  LOG.info("Entering testModifyTable.");
```

```
// Specify the column family name.
byte[] familyName = Bytes.toBytes("info");
Admin admin = null;
try {
  // Instantiate an Admin object.
  admin = conn.getAdmin();
  // Obtain the table descriptor.
  HTableDescriptor htd = admin.getTableDescriptor(tableName);
  // Check whether the column family is specified before modification.
  if (!htd.hasFamily(familyName)) {
    // Create the column descriptor.
    HColumnDescriptor hcd = new HColumnDescriptor(familyName);

    //Disable hot and cold separation.
    hcd .setValue(HColumnDescriptor.COLD_BOUNDARY, null);

    htd.addFamily(hcd);
    // Disable the table to get the table offline before modifying
    // the table.
    admin.disableTable(tableName);
    // Submit a modifyTable request.
    admin.modifyTable(tableName, htd);  //Note [1]
    // Enable the table to get the table online after modifying the
    // table.
    admin.enableTable(tableName);
  }
  LOG.info("Modify table successfully.");
} catch (IOException e) {
  LOG.error("Modify table failed " ,e);
} finally {
  if (admin != null) {
    try {
      // Close the Admin object.
      admin.close();
    } catch (IOException e) {
      LOG.error("Close admin failed " ,e);
    }
  }
}
 LOG.info("Exiting testModifyTable.");
}
```

Note [1] Only after the **disableTable** API is called, the table can be modified by calling the **modifyTable** API. Then, call the **enableTable** API to enable the table again.

NOTE

Note [1] refers to **admin.modifyTable(tableName, htd); //Note[1]** in the sample code.

- Enable cold and hot data separation for an existing table.

```
public void testModifyTable() {
 LOG.info("Entering testModifyTable.");

 // Specify the column family name.
 byte[] familyName = Bytes.toBytes("info");
 Admin admin = null;
 try {
  // Instantiate an Admin object.
  admin = conn.getAdmin();
  // Obtain the table descriptor.
  HTableDescriptor htd = admin.getTableDescriptor(tableName);
  // Check whether the column family is specified before modification.
  if (!htd.hasFamily(familyName)) {
    // Create the column descriptor.
    HColumnDescriptor hcd = new HColumnDescriptor(familyName);

    //Set the hot and cold separation function for an existing table.
```



```
    hcd .setValue(HColumnDescriptor.COLD_BOUNDARY, "86400");

    htd.addFamily(hcd);
    // Disable the table to get the table offline before modifying
    // the table.
    admin.disableTable(tableName);
    // Submit a modifyTable request.
    admin.modifyTable(tableName, htd);  //Note[1]
    // Enable the table to get the table online after modifying the
    // table.
    admin.enableTable(tableName);
  }
  LOG.info("Modify table successfully.");
 } catch (IOException e) {
  LOG.error("Modify table failed " ,e);
 } finally {
  if (admin != null) {
    try {
      // Close the Admin object.
      admin.close();
    } catch (IOException e) {
      LOG.error("Close admin failed " ,e);
    }
  }
 }
 LOG.info("Exiting testModifyTable.");
}
```

Note [1] Only after the **disableTable** API is called, the table can be modified by calling the **modifyTable** API. Then, call the **enableTable** API to enable the table again.

NOTE

Note [1] refers to **admin.modifyTable(tableName, htd); //Note[1]** in the sample code.

1.3.2.9 Inserting HBase Cold and Hot Separation Data

Function Description

HBase is a column-based database. A row of data may have multiple column families, and a column family may contain multiple columns. When writing data, you must specify the columns (including the column family names and column names) to which data is written. HBase allows you to insert data (a row of data or data sets) using the **put** method of HTable.

You can write data to a table that separately stores cold and hot data in a similar manner that you write data to a standard table.

Sample Code

```
public void testPut() {
  LOG.info("Entering testPut.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifiers = { Bytes.toBytes("temp"), Bytes.toBytes("hum") };
  Table table = null;
  try {
    // Instantiate an HTable object.
    table = conn.getTable(tableName);
    // Instantiate a Put object. Every Hour insert one data.
    Put put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 00:00:00"));
```



```
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 01:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
table.put(put);


put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 02:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 03:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
puts.add(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 04:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 05:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 06:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 07:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 08:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 09:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 10:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 11:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 12:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 13:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);
```



```
put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 14:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 15:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 16:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("31.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 17:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);


put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 18:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 19:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 20:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 21:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 22:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 23:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 00:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 01:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 02:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 03:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
```



```
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
puts.add(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 04:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 05:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 06:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 07:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 08:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 09:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 10:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 11:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 12:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 13:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 14:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 15:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 16:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("31.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 17:00:00"));
```



```
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 18:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
    puts.clear();
    puts.add(put);
    table.put(puts);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 19:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 20:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 21:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 22:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 23:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
    table.put(put);

    LOG.info("Put successfully.");
  } catch (IOException e) {
    LOG.error("Put failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testPut.");
}
```

1.3.2.10 Reading HBase Cold and Hot Separation Data Using the GET Command

Function Description

Before reading data from a table, create a table instance and a Get object. You can also set parameters for the Get object, such as the column family name and column name. Query results are stored in the Result object that stores multiple Cells.

For a column family with the cold and hot data separation feature enabled, you can choose to only query the cold or hot data by specifying parameters.

Sample Code

- The query that does not contain the **HOT_ONLY** hint may hit cold data.

```
public void testGet() {
  LOG.info("Entering testGet.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifier = { Bytes.toBytes("temp"), Bytes.toBytes("hum") };
  // Specify RowKey.
  byte[] rowKey = Bytes.toBytes("Shenzhen#Longgang#2017/7/1 03:00:00");
  Table table = null;
  try {
    // Create the Table instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Get get = new Get(rowKey);
    // Set the column family name and column name.
    get.addColumn(familyName, qualifier[0]);
    get.addColumn(familyName, qualifier[1]);
    // Submit a get request.
    Result result = table.get(get);
    // Print query results.
    for (Cell cell : result.rawCells()) {
      LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
          + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
          + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
          + Bytes.toString(CellUtil.cloneValue(cell)));
    }
    LOG.info("Get data successfully.");
  } catch (IOException e) {
    LOG.error("Get data failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testGet.");
}
```

- The query that contains the **HOT_ONLY** hint hits only hot data.

```
public void testGet() {
  LOG.info("Entering testGet.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifier = { Bytes.toBytes("temp"), Bytes.toBytes("hum") };
  // Specify RowKey.
  byte[] rowKey = Bytes.toBytes("Shenzhen#Longgang#2017/7/2 10:00:00");
  Table table = null;
  try {
    // Create the Table instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Get get = new Get(rowKey);
    // Set HOT_ONLY.
    get.setAttribute(HBaseConstants.HOT_ONLY, Bytes.toBytes(true));
    // Set the column family name and column name.
    get.addColumn(familyName, qualifier[0]);
    get.addColumn(familyName, qualifier[1]);
    // Submit a get request.
    Result result = table.get(get);
    // Print query results.
    for (Cell cell : result.rawCells()) {
      LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
```



```
        + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
        + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
        + Bytes.toString(CellUtil.cloneValue(cell)));
    }
    LOG.info("Get data successfully.");
  } catch (IOException e) {
    LOG.error("Get data failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testGet.");
}
```

1.3.2.11 Reading HBase Cold and Hot Separation Data Using the Scan Command

Function Description

Before reading data from a table, instantiate the Table instance of the table, and then create a Scan object and set parameters for the Scan object based on search criteria. To improve query efficiency, you are advised to specify StartRow and StopRow. Query results are stored in the ResultScanner object, where each row of data is stored as a Result object that stores multiple Cells.

Sample Code

- The query that does not contain the **HOT_ONLY** hint may hit cold data.

```
public void testScanData() {
  LOG.info("Entering testScanData.");
  Table table = null;
  // Instantiate a ResultScanner object.
  ResultScanner rScanner = null;
  try {
    // Create the Configuration instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Scan scan = new Scan();
    byte[] startRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/1 00:00:00);
    byte[] stopRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/3 00:00:00);
    scan.setStartRow(startRow);
    scan.setStopRow(stopRow);
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("temp"));
    // Set the cache size.
    scan.setCaching(1000);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Scan data successfully.");
  } catch (IOException e) {
    LOG.error("Scan data failed " ,e);
```



```
  } finally {
    if (rScanner != null) {
      // Close the scanner object.
      rScanner.close();
    }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testScanData.");
}
```

- The query that contains the **HOT_ONLY** hint hits only hot data.

```
public void testScanData() {
  LOG.info("Entering testScanData.");
  Table table = null;
  // Instantiate a ResultScanner object.
  ResultScanner rScanner = null;
  try {
    // Create the Configuration instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Scan scan = new Scan();
    byte[] startRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/1 00:00:00);
    byte[] stopRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/3 00:00:00);
    scan.setStartRow(startRow);
    scan.setStopRow(stopRow);
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("temp"));

    // Set HOT_ONLY.
    scan.setAttribute(HBaseConstants.HOT_ONLY, Bytes.toBytes(true));
    // Set the cache size.
    scan.setCaching(1000);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Scan data successfully.");
  } catch (IOException e) {
    LOG.error("Scan data failed " ,e);
  } finally {
    if (rScanner != null) {
      // Close the scanner object.
      rScanner.close();
    }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testScanData.");
}
```


1.3.3 Configuring HBase Multi-Language Access

Scenario

Users access the corresponding ThriftServer instance based on the specified host and port to create and delete HBase tables.

Prerequisites

- ThriftServer has been enabled for the cluster. You have obtained the ThriftServer IP address from the cluster details page.
- You have downloaded the Thrift installation package from **link**.
- You have downloaded the HBase Thrift definition file from **address**.

Procedure

Step 1 Log in to the CloudTable console.

Step 2 Select a region in the upper left corner of the page.

Step 3 Click **Cluster Management** to go to the cluster management page.

Step 4 Click the name of an HBase cluster to go to the cluster details page and check the Thrift Server status. If Thrift Server is enabled, no further action is required. If Thrift Server is disabled, return to the cluster management page and choose **More** > **Enable Thrift Server**.

NOTE

- ThriftServer does not support load balancing. To prevent overloading a single instance, avoid accessing the same ThriftServer instance concurrently in your code.
- Implement a retry mechanism in the application code to ensure that other ThriftServer instances are retried if a single instance fails or is restarted.

Step 5 Install the Thrift installation package on the client node. For details, see the **Thrift official guide**.

Step 6 Run the Thrift command to convert the HBase Thrift definition file to the interface file of the corresponding language. The supported languages include C++ and Python. The following command is used as an example.

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

NOTE

<Language> indicates the target language to be generated. The value can be **cpp** (C++) or **py** (Python).

Take Python as an example. Run **thrift --gen py hbase.thrift**.

----End

C++ Code Example

```
#include "THBaseService.h"
#include <config.h>
#include <vector>
#include <ostream>
#include <iostream>
#include "transport/TSocket.h"
```



```
#include <transport/TBufferTransports.h>
#include <protocol/TBinaryProtocol.h>
using namespace std;
using namespace apache::thrift;
using namespace apache::thrift::protocol;
using namespace apache::thrift::transport;
using namespace apache::hadoop::hbase::thrift2;
using boost::shared_ptr;
int main(int argc, char **argv) {
// IP address and port number of ThriftServer
    std::string host = "x.x.x.x";
    int port = 9090;
    boost::shared_ptr<TSocket> socket(new TSocket(host, port));
    boost::shared_ptr<TTransport> transport(new TBufferedTransport(socket));
    boost::shared_ptr<TProtocol> protocol(new TBinaryProtocol(transport));
// Set the table name.
    std::string ns("default");
    std::string table("test");
    TTableName tableName;
    tableName.__set_ns(ns);
    tableName.__set_qualifier(table);
     try {
        // Create a connection.
        transport->open();
        printf("Opened connection\n");
// Initialize the client.
        THBaseServiceClient  client(protocol);

       // Create a table.
        TColumnFamilyDescriptor column;
        column.__set_name("f1");
        column.__set_maxVersions(10);
        std::vector<TColumnFamilyDescriptor> columns;
        columns.push_back(column);

        TTableDescriptor tableDescriptor;
        tableDescriptor.__set_tableName(tableName);
        tableDescriptor.__set_columns(columns);
        std::vector<std::string> splitKeys;
        splitKeys.push_back("row2");
        splitKeys.push_back("row4");
        splitKeys.push_back("row8");
        printf("Creating table: %s\n", table.c_str());
        try {
            client.createTable(tableDescriptor, splitKeys);
        } catch (const TException  &te) {
            std::cerr << "ERROR: " << te.what() << std::endl;
        }
// Put a single piece of data.
        TColumnValue columnValue;
        columnValue.__set_family("f1");
        columnValue.__set_qualifier("q1");
        columnValue.__set_value("val_001");
        std::vector<TColumnValue> columnValues;
        columnValues.push_back(columnValue);
        TPut put;
        put.__set_row("row1");
        put.__set_columnValues(columnValues);
        client.put(table, put);
        printf("Put single row success\n");
// Put multiple pieces of data.
        TColumnValue columnValue2;
        columnValue2.__set_family("f1");
        columnValue2.__set_qualifier("q1");
        columnValue2.__set_value("val_003");
        std::vector<TColumnValue> columnValues2;
        columnValues2.push_back(columnValue2);
        TPut put2;
        put2.__set_row("row3");
```



```
        put2.__set_columnValues(columnValues2);
        TColumnValue columnValue3;
        columnValue3.__set_family("f1");
        columnValue3.__set_qualifier("q1");
        columnValue3.__set_value("val_005");
        std::vector<TColumnValue> columnValues3;
        columnValues3.push_back(columnValue3);
        TPut put3;
        put3.__set_row("row5");
        put3.__set_columnValues(columnValues3);
        std::vector<TPut> puts;
        puts.push_back(put2);
        puts.push_back(put3);
        client.putMultiple(table, puts);
        printf("Put multiple rows success\n");
// Get a single data record.
        TResult result;
        TGet get;
        get.__set_row("row1");
        client.get(result, table, get);
        std::vector<TColumnValue> list=result.columnValues;
        std::vector<TColumnValue>::const_iterator iter;
        std::string row = result.row;
        for(iter=list.begin();iter!=list.end();iter++) {
            printf("%s=%s, %s,%s\n",row.c_str(),(*iter).family.c_str(),(*iter).qualifier.c_str(),(*iter).value.c_str());
        }
        printf("Get single row success.\n");
// Get multiple data records.
        std::vector<TGet> multiGets;
        TGet get1;
        get1.__set_row("row1");
        multiGets.push_back(get1);
        TGet get2;
        get2.__set_row("row5");
        multiGets.push_back(get2);

        std::vector<TResult> multiRows;
        client.getMultiple(multiRows, table, multiGets);
        for(std::vector<TResult>::const_iterator iter1=multiRows.begin();iter1!=multiRows.end();iter1++) {
            std::vector<TColumnValue> list=(*iter1).columnValues;
            std::vector<TColumnValue>::const_iterator iter2;
            std::string row = (*iter1).row;
            for(iter2=list.begin();iter2!=list.end();iter2++) {
                printf("%s=%s, %s,%s\n",row.c_str(),(*iter2).family.c_str(),(*iter2).qualifier.c_str(),
(*iter2).value.c_str());
            }
        }
        printf("Get multiple rows success.\n");
// Scan to query data.
        TScan scan;
        scan.__set_startRow("row1");
        scan.__set_stopRow("row7");
        int32_t nbRows = 2;
        std::vector<TResult> results;
        TResult* current = NULL;
        while (true) {
            client.getScannerResults(results, table, scan, nbRows);
            if (results.size() == 0) {
                printf("No more result.\n");
                break;
            }
            std::vector<TResult>::const_iterator itx;
            for(itx=results.begin();itx!=results.end();itx++) {
                current = (TResult*) &(*itx);
                if (current == NULL) {
                    break;
                } else {
                    std::vector<TColumnValue> values=(*current).columnValues;
                    std::vector<TColumnValue>::const_iterator iterator;
```



```
                for(iterator=list.begin();iterator!=list.end();iterator++) {
                    printf("%s=%s, %s,%s\n",(*current).row.c_str(),(*iterator).family.c_str(),
(*iterator).qualifier.c_str(),(*iterator).value.c_str());
                }
            }
        }
        if (current == NULL) {
            printf("Scan data done.\n");
            break;
        } else {
            scan.__set_startRow((*current).row + (char)0);
        }
    }
// Disable and delete a table.
    client.disableTable(tableName);
    printf("Disabled %s\n", table.c_str());
    client.deleteTable(tableName);
    printf("Deleted %s\n", table.c_str());
    transport->close();
    printf("Closed connection\n");
  } catch (const TException &tx) {
    std::cerr << "ERROR(exception): " << tx.what() << std::endl;
  }
  return 0;
}
```

Python Code Example

```
# -*- coding: utf-8  -*-

# Import the common module.
import sys
import os

# Import the built-in module of Thrift. If the module does not exist, run the pip install thrift command to
install it.
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
from thrift.transport import THttpClient
from thrift.transport import TSocket

# Import the module generated by hbase.thrift.
gen_py_path = os.path.abspath('gen-py')
sys.path.append(gen_py_path)
from hbase import THBaseService
from hbase.ttypes import TColumnValue, TColumn, TTableName, TTableDescriptor,
TColumnFamilyDescriptor, TGet, TPut, TScan
# Configure the IP address of ThriftServer in the CloudTable HBase cluster. You can obtain the IP address
from the cluster details page.
host = "x.x.x.x"

socket = TSocket.TSocket(host, 9090)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = THBaseService.Client(protocol)
transport.open()

# Test table name
tableNameInBytes = "test".encode("utf8")

tableName = TTableName(ns="default".encode("utf8"), qualifier=tableNameInBytes)
# Split key for region pre-splitting
splitKeys=[]
splitKeys.append("row3".encode("utf8"))
splitKeys.append("row5".encode("utf8"))
# Create a table.
client.createTable(TTableDescriptor(tableName=tableName,
columns=[TColumnFamilyDescriptor(name="cf1".encode("utf8"))]), splitKeys)
print("Create table %s success." % tableName)
```



```
# Put a single data record.
put = TPut(row="row1".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))])
client.put(tableNameInBytes, put)
print("Put single row success.")

# Put multiple data records.
puts = []
puts.append(TPut(row="row4".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))]))
puts.append(TPut(row="row6".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))]))
puts.append(TPut(row="row8".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))]))
client.putMultiple(tableNameInBytes, puts)
print("Put rows success.")

# Get a single data record.
get = TGet(row="row1".encode("utf8"))
result = client.get(tableNameInBytes, get)
print("Get Result: ", result)

# Get multiple data records.
gets = []
gets.append(TGet(row="row4".encode("utf8")))
gets.append(TGet(row="row8".encode("utf8")))
results = client.getMultiple(tableNameInBytes, gets)
print("Get multiple rows: ", results)

# Scan data.
startRow, stopRow = "row4".encode("utf8"), "row9".encode("utf8")
scan = TScan(startRow=startRow, stopRow=stopRow)
caching=1
results = []
while True:
    scannerResult = client.getScannerResults(tableNameInBytes, scan, caching)
    lastOne = None
    for result in scannerResult:
        results.append(result)
        print("Scan Result: ", result)
        lastOne = result
# No more data. Exit.
    if lastOne is None:
        break
    else:
# Regenerate the start row of the next scan.
        newStartRow = bytearray(lastOne.row)
        newStartRow.append(0x00)
        scan = TScan(startRow=newStartRow, stopRow=stopRow)


# Disable and delete a table.
client.disableTable(tableName)
print("Disable table %s success." % tableName)
client.deleteTable(tableName)
print("Delete table %s success." % tableName)

# Close the connection after all operations are complete.
transport.close()
```

1.4 Commissioning Applications

1.4.1 Commissioning Applications on Windows

1.4.1.1 Compiling and Running Applications

Scenario

You can run applications in the Windows development environment after application code development is complete.

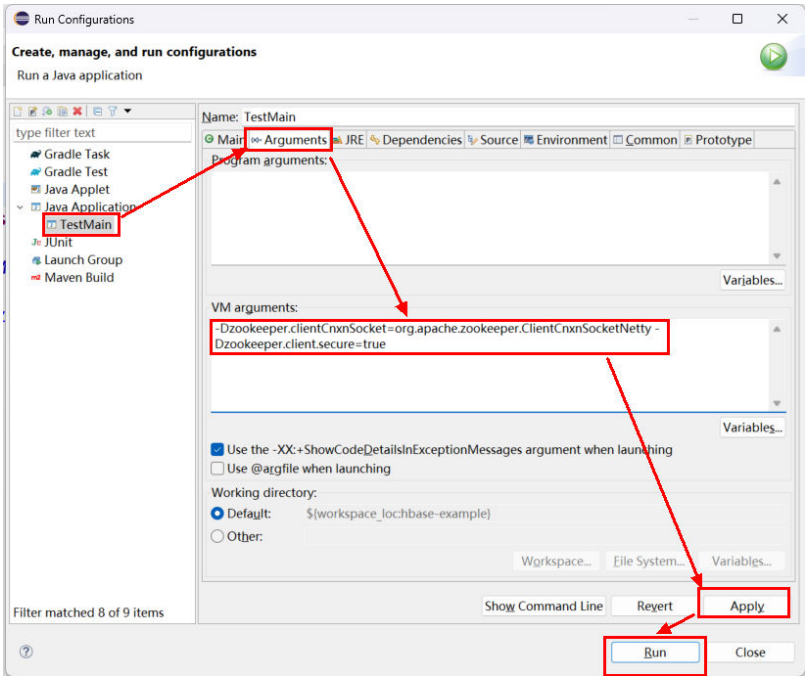
Procedure

- **HBase clusters with the encryption stream disabled**

  In the development environment (for example, Eclipse), right-click **TestMain.java** and choose **Run as** > **Java Application** from the shortcut menu to run the corresponding application project.

- **HBase clusters with the encryption stream enabled**

  In the development environment (for example, Eclipse), right-click **TestMain.java**, choose **Run as** > **Run Configurations**, add the environment variable "**-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true**", and run the corresponding application project.



1.4.1.2 Viewing Commissioning Results

If no exception or failure information is displayed, the application running is successful.

Figure 1-8 Running succeeded

```
2016-07-13 14:36:12,736 INFO  [main] basic.CreateTableSample: Create table sampleNameSpace:sampleTable successful!
2016-07-13 14:36:15,426 INFO  [main] basic.ModifyTableSample: Modify table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:16,708 INFO  [main] basic.MultiSplitSample: Mmulti split table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:17,299 INFO  [main] basic.PutDataSample: Successfully put 9 items data into sampleNameSpace:sampleTable.
2016-07-13 14:36:18,992 INFO  [main] basic.ScanSample: Scan data successfully.
2016-07-13 14:36:20,532 INFO  [main] basic.DeletaDataSample: Successfully delete data from table sampleNameSpace:sampleTable.
2016-07-13 14:36:21,006 INFO  [main] acl.AclSample: Grant ACL for table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:27,836 INFO  [main] index.CreateIndexSample: Successfully add index for table sampleNameSpace:sampleTable.
```

NOTE

The following exception may occur when the sample code is running in the Windows OS, but it will not affect services.

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

Log description:

The log level is INFO by default and you can view more detailed information by changing the log level, such as DEBUG, INFO, WARN, ERROR, and FATAL. You can modify the **log4j.properties** file to change log levels, for example:

```
hbase.root.logger=INFO,console
log4j.logger.org.apache.zookeeper=INFO
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG
log4j.logger.org.apache.hadoop.hbase=INFO
# Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO
```

1.4.2 Commissioning Applications on Linux

1.4.2.1 Compiling and Running an Application When a Client Is Installed

Scenario

HBase applications can run in a Linux environment where an HBase client is installed. After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

Prerequisites

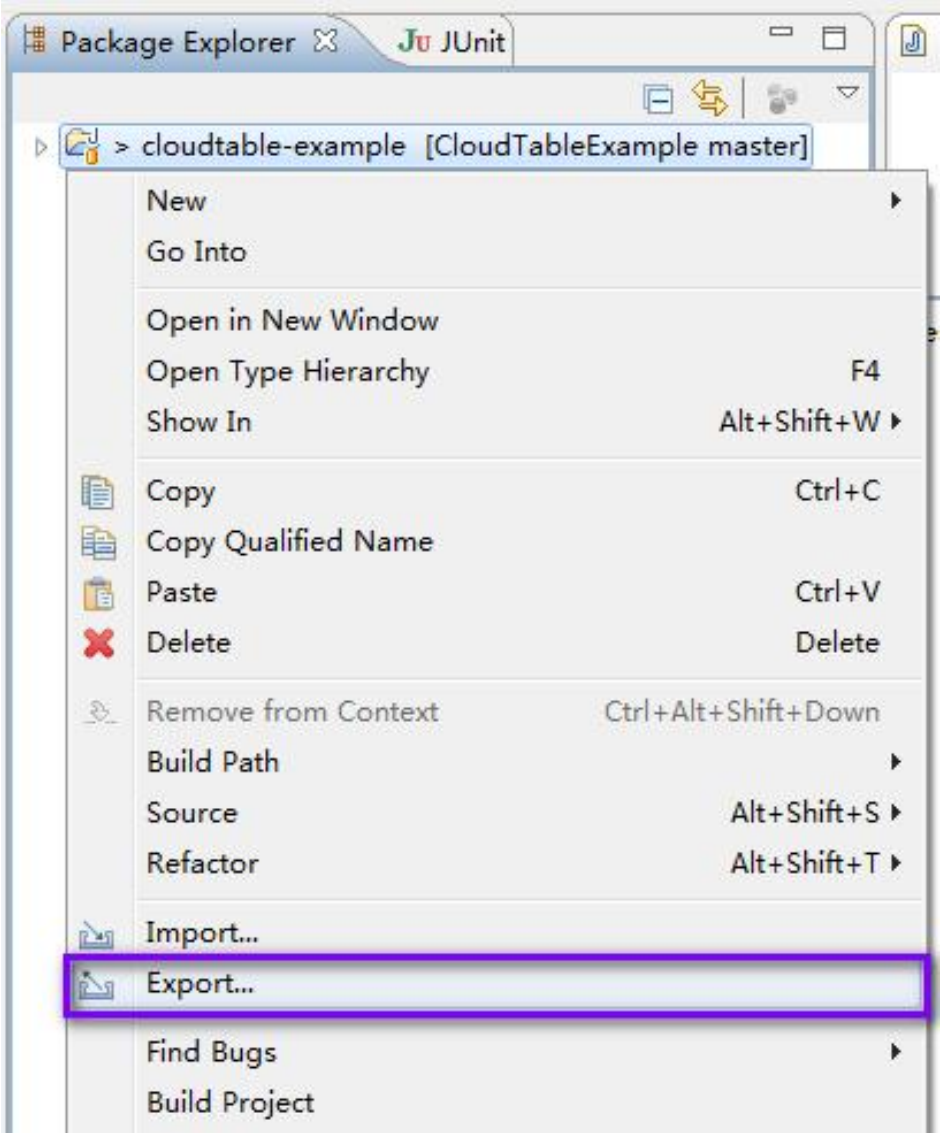
- You have installed an HBase client.
- You have installed a JDK in the Linux environment. The version of the JDK must be consistent with that of the JDK used by Eclipse to export the JAR file.

Procedure

Step 1 Export a JAR file.

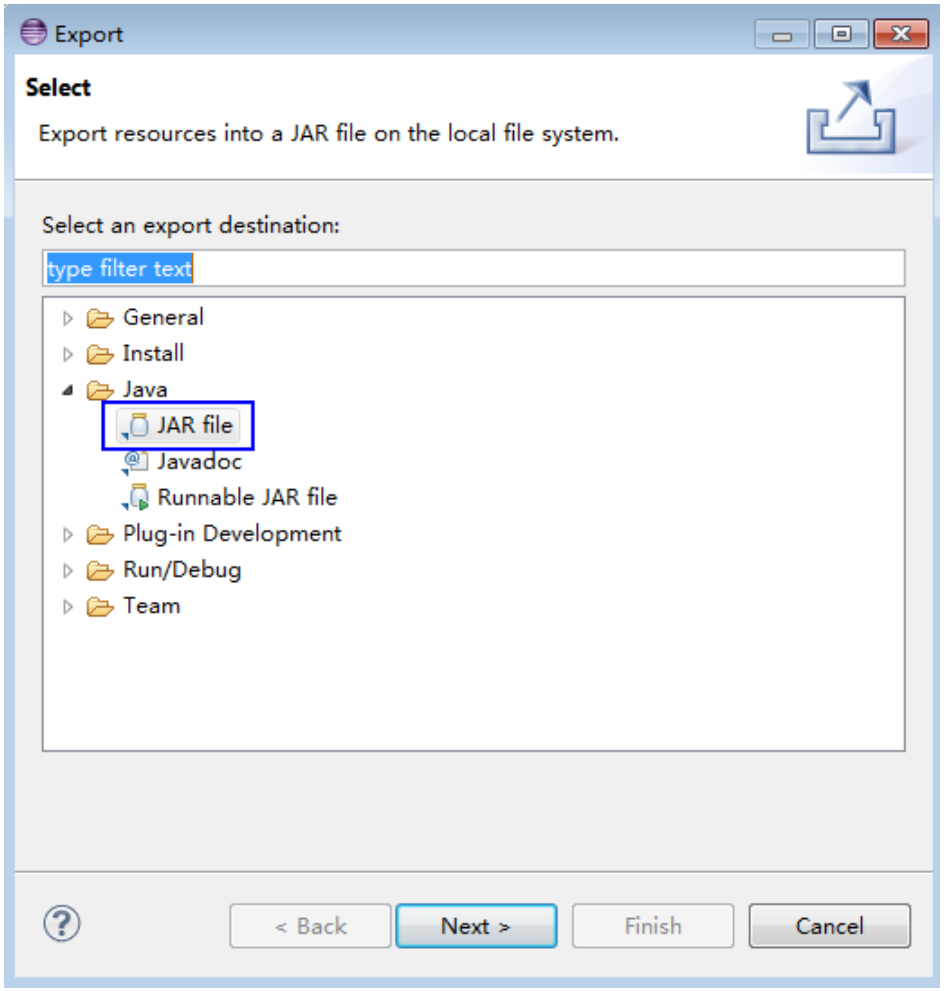
1. Right-click the sample project and choose **Export** from the shortcut menu.

Figure 1-9 Exporting a JAR file



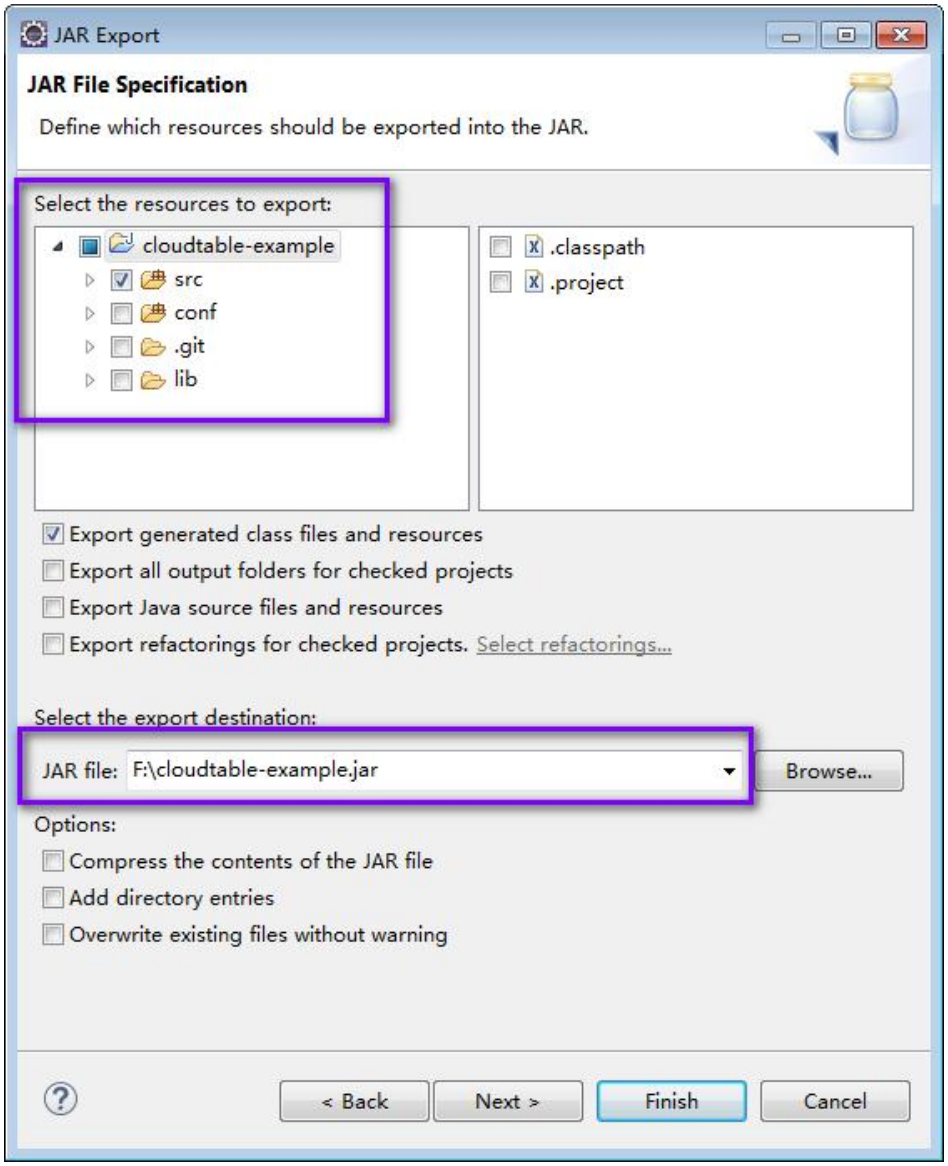
2. Select **JAR file** and click **Next**.

Figure 1-10 Selecting JAR file



3. Select the **src** and **conf** directories, and export the JAR file to the specified location. Click **Next** twice.

Figure 1-11 Selecting a path for exporting the JAR file



4. Click **Finish**. Exporting the JAR file is complete.

Step 2 Run the JAR file.

1. Before running the JAR file on the Linux client, copy the JAR file generated in the application development environment to the **lib** directory in the client installation directory, and ensure that the permission of the JAR file is the same as that of other files.
2. Switch to the **bin** directory in the client directory as the installation user, and then run the following command to Run the JAR file:

```
[Ruby@cloudtable-08261700-hmaster-1-1 bin]# ./hbase
com.huawei.cloudtable.hbase.examples.TestMain
```

com.huawei.cloudtable.hbase.examples.TestMain is used as an example. Use the actual code instead.

----End

1.4.2.2 Compiling and Running an Application When No Client Is Installed

Scenario

HBase applications can run in a Linux environment where an HBase client is not installed. After application code development is complete, you can upload a JAR file to the Linux environment to run applications.

Prerequisites

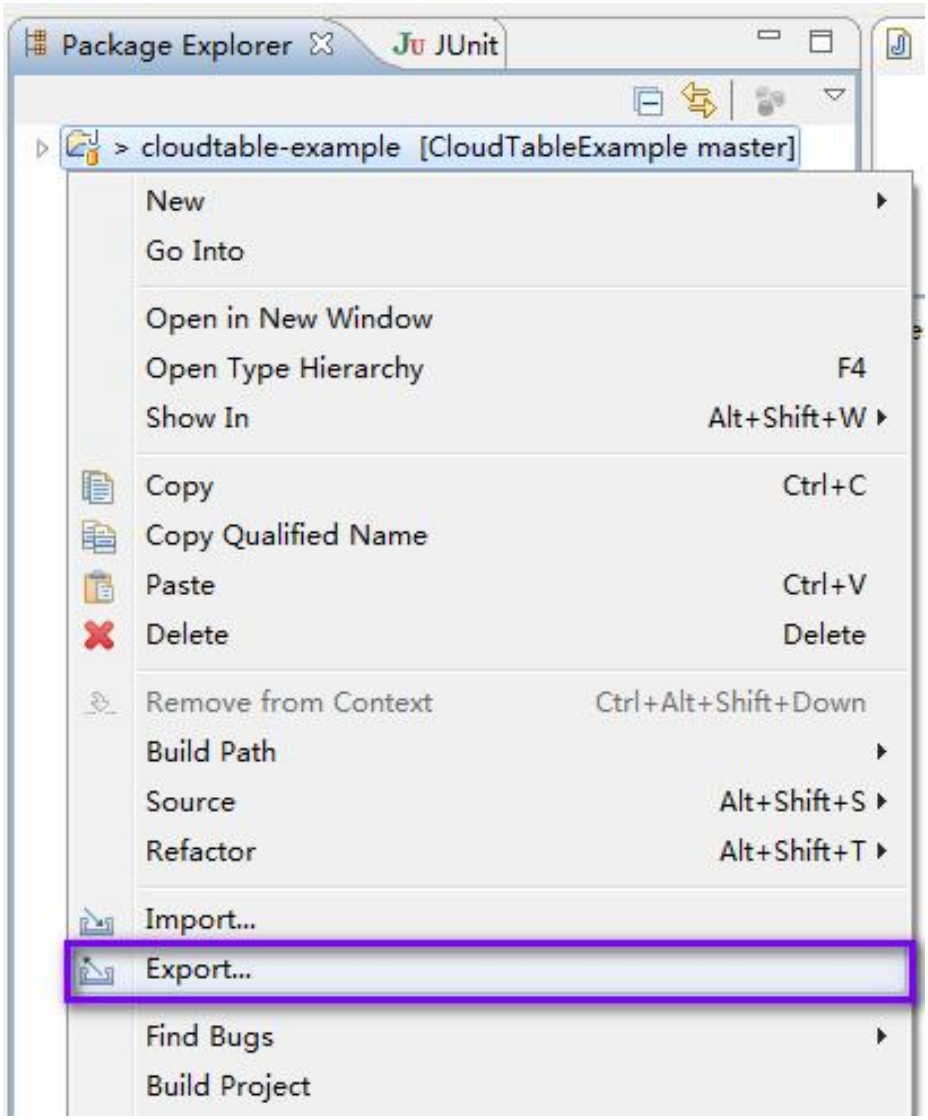
You have installed a JDK in the Linux environment. The version of the JDK must be consistent with that of the JDK used by Eclipse to export the JAR file.

Procedure

Step 1 Export a JAR file.

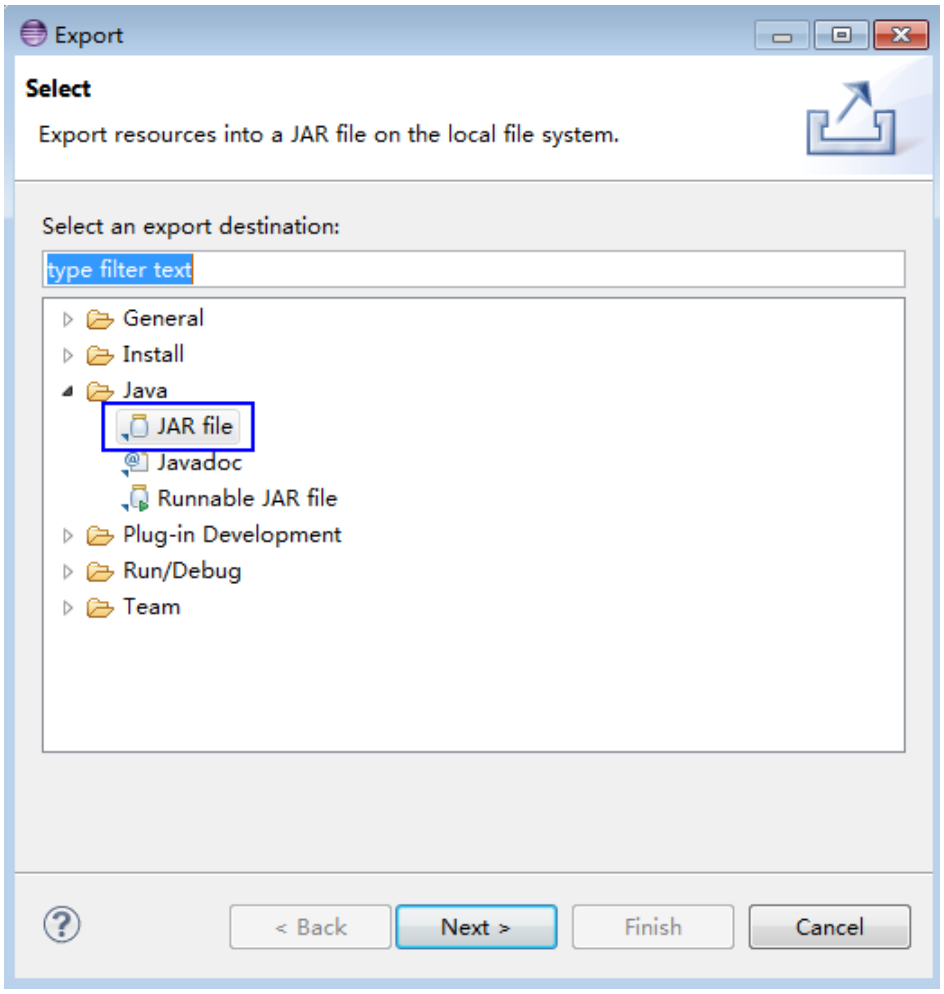
1. Right-click the sample project and choose **Export** from the shortcut menu.

Figure 1-12 Exporting a JAR file



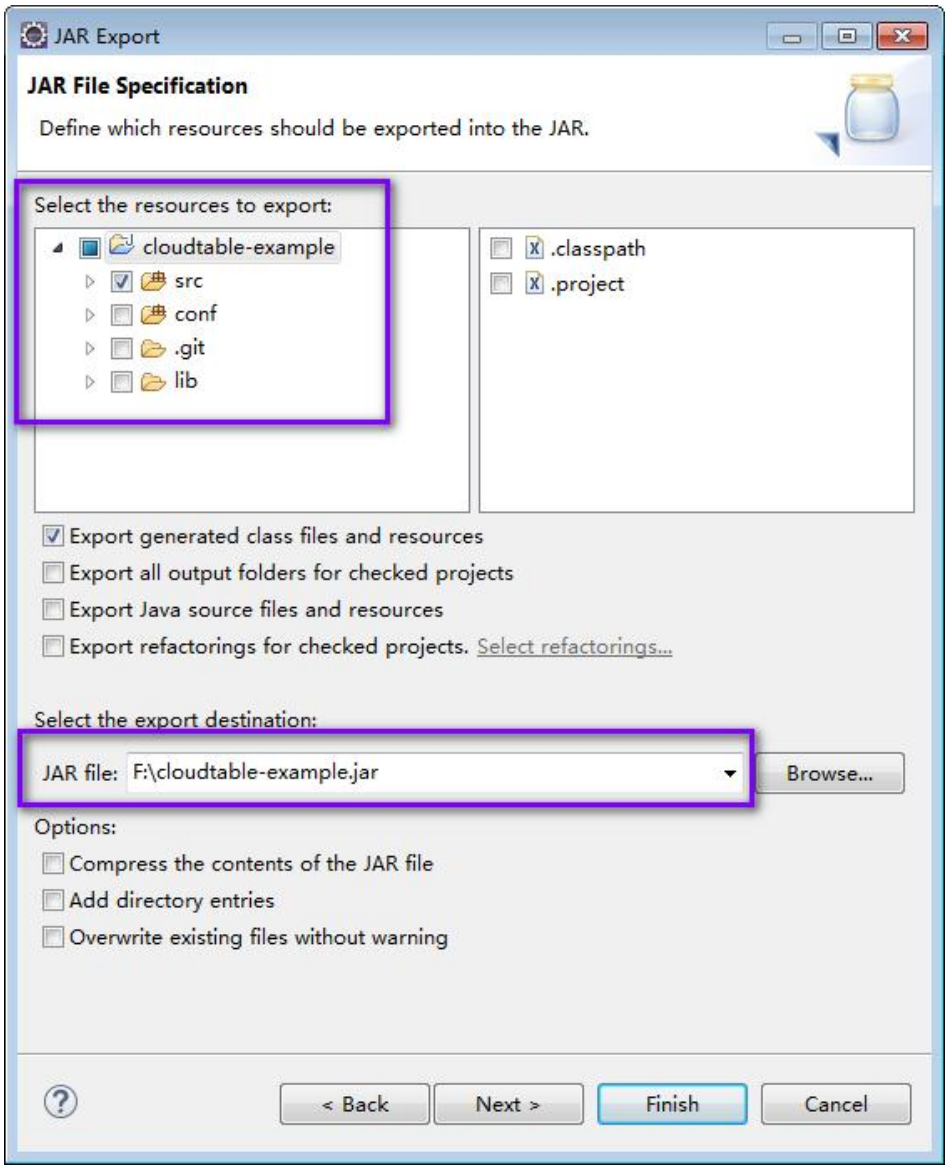
2. Select **JAR file** and click **Next**.

Figure 1-13 Selecting JAR file



3. Select the **src** directory, and export the JAR file to the specified location. Click **Next** twice.

Figure 1-14 Selecting a path for exporting the JAR file



4. Click **Finish**. Exporting the JAR file is complete.

Step 2 Prepare the required JAR file and configuration file.

1. In the Linux environment, create a directory, for example, **/opt/test**, and create subdirectories **lib** and **conf**. Upload the JAR file in **lib** in the sample project and the JAR file exported in **Step 1** to the **lib** directory on Linux. Upload the configuration file in **conf** in the sample project to the **conf** directory on Linux.
2. In the **/opt/test** root directory, create the **run.sh** script, modify the following content, and save the file:

```
#!/bin/sh
BASEDIR=`pwd`
SECURE=""
if [ $# -eq 1 ]; then
  SECURE="-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -
Dzookeeper.client.secure=true"
fi
cd ${BASEDIR}
for file in ${BASEDIR}/lib/*.jar
```



```
do
i_cp=$i_cp:$file
echo "$file"
done
for file in ${BASEDIR}/conf/*
do
i_cp=$i_cp:$file
done
java -cp .${i_cp} ${SECURE} com.huawei.cloudtable.hbase.examples.TestMain
```

Step 3 Go to **/opt/test** and run the following command to run the JAR file:

- **HBase clusters with the encryption stream disabled**

  sh run.sh

- **HBase clusters with the encryption stream enabled**

  sh run.sh secure

  NOTE

  If you use other methods to access an HBase cluster with the encryption stream enabled, you need to add the parameter "**-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true**".

----End

1.4.2.3 Viewing Commissioning Results

If no exception or failure information is displayed, the application running is successful.

Figure 1-15 Running succeeded

```
2016-07-13 14:36:12,736 INFO  [main] basic.CreateTableSample: Create table sampleNameSpace:sampleTable successful!
2016-07-13 14:36:15,426 INFO  [main] basic.ModifyTableSample: Modify table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:16,708 INFO  [main] basic.MultiSplitSample: Mmulti split table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:17,299 INFO  [main] basic.PutDataSample: Successfully put 9 items data into sampleNameSpace:sampleTable.
2016-07-13 14:36:18,992 INFO  [main] basic.ScanSample: Scan data successfully.
2016-07-13 14:36:20,532 INFO  [main] basic.DeletaDataSample: Successfully delete data from table sampleNameSpace:sampleTable.
2016-07-13 14:36:21,006 INFO  [main] acl.AclSample: Grant ACL for table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:27,836 INFO  [main] index.CreateIndexSample: Successfully add index for table sampleNameSpace:sampleTable.
```

The log level is INFO by default and you can view more detailed information by changing the log level, such as DEBUG, INFO, WARN, ERROR, and FATAL. You can modify the **log4j.properties** file to change log levels, for example:

```
hbase.root.logger=INFO,console
log4j.logger.org.apache.zookeeper=INFO
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG
log4j.logger.org.apache.hadoop.hbase=INFO
# Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO
```


2 Doris Application Development Guide

2.1 Using JDBC to Connect to Doris Clusters

2.1.1 Using JDBC to Connect to Doris Clusters in Non-SSL Mode

When code retry and load balancing are performed at the application layer, multiple Doris FE node addresses need to be configured for an application. For example, if a connection exits abnormally, the system automatically retries on another connection.

JDBC Connector

If the MySQL JDBC Connector is used to connect to Doris, the automatic retry mechanism of JDBC can be used.

```
private static String URL = "jdbc:mysql:loadbalance://" +
      "[FE1_host]:[FE1_port],[FE2_host]:[FE2_port],[FE3_host]:[FE3_port]/demo?" +
      "loadBalanceConnectionGroup=first&ha.enableJMX=true";
```

Sample code:

```
public class Test {
   private static String URL = "jdbc:mysql:loadbalance://" +
      "FE1:9030,FE2:9030,FE3:9030/demo?" +
      "loadBalanceConnectionGroup=first&ha.enableJMX=true";
   static Connection getNewConnection() throws SQLException, ClassNotFoundException {
      Class.forName("com.mysql.cj.jdbc.Driver");
      // There will be security risks if the password used for authentication is directly written into code. Encrypt the password in the configuration file or environment variables for storage;
      // In this example, the password is stored in environment variables for identity authentication. Before running the code in this example, configure environment variables first.
      String password = System.getenv("USER_PASSWORD");
      return DriverManager.getConnection(URL, "admin", password);
   }
   public static void main(String[] args) throws Exception {
      Connection c = getNewConnection();
      while (true) {
         try {
            String query = "your sqlString";
            c.setAutoCommit(false);
            Statement s = c.createStatement();
```



```
            ResultSet resultSet = s.executeQuery(query);
            System.out.println("begin print");
            while(resultSet.next()) {
                int id = resultSet.getInt(1);
                System.out.println("id is: "+id);
            }
            System.out.println("end print");
            Thread.sleep(Math.round(100 * Math.random()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  }
}
```

2.1.2 Using JDBC to Connect to Doris in SSL Mode (Certificate Verification Needed)

When code retry and load balancing are performed at the application layer, multiple Doris FE node addresses need to be configured for an application. If a connection exits abnormally, the system automatically retries on another connection.

NOTE

The HTTPS must be enabled for the cluster in advance.

Download the certificate on the cluster details page.

1. Run the following command on the ECS where the MySQL client has been installed to import the server certificate:
   - **your_certificate_path**: path for storing the custom certificate
   - **your_truststore_name**: user-defined truststore name
   - **your_truststore_password**: user-defined truststore password

   ```
   keytool -importcert -alias MySQLCACert -file your_certificate_path -keystore your_truststore_name -storepass your_truststore_password
   ```

2. Enter **yes** when prompted, as shown in the following figure.

   Figure 2-1 Running the command

   ```
   Owner: CN=MySQL_Server_5.7.17_Auto_Generated_CA_Certificate
   Issuer: CN=MySQL_Server_5.7.17_Auto_Generated_CA_Certificate
   Serial number: 1
   Valid from: Thu Feb 16 11:42:43 EST 2017 until: Sun Feb 14 11:42:43 EST 2027
   Certificate fingerprints:
        MD5:  18:87:97:37:EA:CB:0B:5A:24:AB:27:76:45:A4:78:C1
        SHA1: 2B:0D:D9:69:2C:99:BF:1E:2A:25:4E:8D:2D:38:B8:70:66:47:FA:ED
        SHA256: C3:29:67:1B:E5:37:06:F7:A9:93:DF:C7:B3:27:5E:09:C7:FD:EE:2D:18:86:F4:9C:40:D8:26:CB:DA:95:A0:24
   Signature algorithm name: SHA256withRSA
   Subject Public Key Algorithm: 2048-bit RSA key
   Version: 1
   Trust this certificate? [no]:  yes
   Certificate was added to keystore
   ```

3. Execute the following sample code.

   your_truststore_path indicates the truststore file path and **your_truststore_password** indicates the truststore password set in the command.

   ```
   public class Main {
    private static String URL = "jdbc:mysql:loadbalance://" +
            "[FE1_host]:[FE1_port],[FE2_host]:[FE2_port],[FE3_host]:[FE3_port]/[your_database]?" +
            "loadBalanceConnectionGroup=first&ha.enableJMX=true";
    static Connection getNewConnection() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        // There will be security risks if the password used for authentication is directly written into code.
   ```



```
Encrypt the password in the configuration file or environment variables for storage;
    // In this example, the password is stored in environment variables for identity authentication.
Before running the code in this example, configure environment variables first.
    String storePassword = System.getenv("STORE_PASSWORD");
    String userPassword = System.getenv("USER_PASSWORD");
    System.setProperty("javax.net.ssl.trustStore","your_truststore_path");
    System.setProperty("javax.net.ssl.trustStorePassword",storePassword);
    String user = "your_username";
    Properties props = new Properties();
    props.setProperty("user", user);
    props.setProperty("password", userPassword);
    props.setProperty("useSSL", "true");
    props.setProperty("requireSSL", "true");
    props.setProperty("verifyServerCertificate", "true");
    props.setProperty("sslMode", "VERIFY_CA");
    return DriverManager.getConnection(URL, props);
 }
 public static void main(String[] args) throws Exception {
    Connection c = getNewConnection();
    try {
        System.out.println("begin print");
        String query = "your sqlString";
        c.setAutoCommit(false);
        Statement s = c.createStatement();
        ResultSet resultSet = s.executeQuery(query);
        while(resultSet.next()) {
            int id = resultSet.getInt(1);
            System.out.println("id is: "+id);
        }
        System.out.println("end print");
        Thread.sleep(Math.round(100 * Math.random()));
        c.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
 }
}
```

2.1.3 Using JDBC to Connect to Doris in SSL Mode (Certificate Verification Not Needed)

When code retry and load balancing are performed at the application layer, multiple Doris FE node addresses need to be configured for an application. If a connection exits abnormally, the system automatically retries on another connection.

- The HTTPS must be enabled for the cluster in advance.
- Download the certificate on the cluster details page.

Sample code:

```
public class Main {
  private static String URL = "jdbc:mysql:loadbalance://" +
          "[FE1_host]:[FE1_port],[FE2_host]:[FE2_port],[FE3_host]:[FE3_port]/[your_database]?" +
          "loadBalanceConnectionGroup=first&ha.enableJMX=true";
  static Connection getNewConnection() throws SQLException, ClassNotFoundException {
     Class.forName("com.mysql.cj.jdbc.Driver");
     // There will be security risks if the password used for authentication is directly written into code.
Encrypt the password in the configuration file or environment variables for storage;
     // In this example, the password is stored in environment variables for identity authentication. Before
running the code in this example, configure environment variables first.
     String password = System.getenv("USER_PASSWORD");
     String user = "your_username";
     Properties props = new Properties();
     props.setProperty("user", user);
     props.setProperty("password", password);
```



```
        props.setProperty("useSSL", "true");
        props.setProperty("requireSSL", "true");
        return DriverManager.getConnection(URL, props);
    }
    public static void main(String[] args) throws Exception {
        Connection c = getNewConnection();
        try {
            System.out.println("begin print");
            String query = "your sqlString";
            c.setAutoCommit(false);
            Statement s = c.createStatement();
            ResultSet resultSet = s.executeQuery(query);
            while(resultSet.next()) {
                int id = resultSet.getInt(1);
                System.out.println("id is: "+id);
            }
            System.out.println("end print");
            Thread.sleep(Math.round(100 * Math.random()));
            c.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.2 Doris Usage Specifications

2.2.1 Doris Table Creation Rules

This topic describes the rules and suggestions for creating Doris tables.

Table Creation Rules

- [Mandatory] Ensure each bucket's data size is between 100 MB and 3 GB when defining buckets for table creation. A single partition's maximum data size should not exceed 5000 GB.
- [Mandatory] Implement a bucket policy when a table's data records surpass 500 million.
- [Mandatory] Limit bucket columns to one or two. Balance data distribution and query throughput to avoid data skew, which can impact overall balance and efficiency. Optimize query performance by tailoring bucket columns to evenly distributed data that is frequently used in query conditions.
- [Mandatory] Avoid dynamic partitions for datasets smaller than 20 billion records to prevent excessive bucket creation.
- Set the replication factor for a new table to a minimum of two (with three as the default). Single backups are not recommended.
- [Optional] Limit materialized views to six.
- For substantial historical data with small, unbalanced sizes or low query likelihood, do as follows:
  - Create dedicated historical partitions (yearly or monthly) to store all relevant data.
  - Historical partition creation example: FROM ("2000-01-01") TO ("2022-01-01") INTERVAL 1 YEAR.

- [Optional] For data records ranging from 10 million to 200 million, bypass partitioning in favor of direct bucket policy application. Doris defaults to a standard partition in the absence of user-defined partitions.
- [Optional] If bucket fields experience over 30% data skew, switch from hash to random bucketing policies. DISTRIBUTED BY RANDOM BUCKETS 10 ...
- [Optional] Prioritize the most frequently queried column as the first field when creating a table to leverage the prefix index's quick query feature. For high-cardinality columns outside the bucket, choose those with extensive query lengths. Note that the prefix index is limited to 36 bits; overly long columns may not utilize this feature.
- [Optional] Utilize inverted indexes or Bloom filters for fuzzy matches or equivalent/in conditions on large datasets. Bitmap indexes are best suited for orthogonal queries with low-cardinality columns.
- [Optional] Manually create buckets based on service requirements. Do not use the AUTO policy.
- [Recommendation] For 2.1 and later versions, if a large amount of data is frequently imported to the database, MOR is recommended. If the query performance is preferentially considered and a long time for importing data to the database is acceptable, MOW is recommended. MOW is not recommended for 2.0 and earlier versions.

2.2.2 Doris Data Change Rules

This topic describes the rules and suggestions for changing Doris data.

Data Modification Rules

- [Mandatory] Applications must not directly execute delete or update statements for data modifications. Instead, employ the CDC's upsert method.
  - The upsert method is suitable for infrequent operations, such as updates every few minutes.
  - Specify partition conditions when using delete statements.
- [Mandatory] Use **INSERT INTO tbl1 VALUES ("1"), ("a")** for small-scale data imports. For larger imports, opt for Doris-provided methods like StreamLoad, BrokerLoad, SparkLoad, or Flink Connector.
- [Optional] For extended SQL operations, set session variables in hint mode using **SELECT /*+ SET_VAR(query_timeout = xxx*/ from table**. Avoid altering global system variables.

2.2.3 Doris Naming Rules

This topic describes the rules and suggestions for naming databases and tables.

Naming Rules

- [Mandatory] The database character set must be UTF-8, as it is the only supported format.
- [Optional] Use lowercase letters with underscores (_) as separators for database names, ensuring the name does not surpass 62 bytes.
- [Optional] Table names are case-sensitive. Use lowercase letters with underscores (_) as separators, ensuring the name does not surpass 64 bytes.

2.2.4 Doris Data Query Rules

This topic describes the rules and suggestions for querying Doris data.

Data Query Rules

- [Mandatory] Doris does not support foreign table query because foreign tables are unstable.
- [Mandatory] Modify the query to a subquery if the **in** conditions exceed 2,000.
- [Mandatory] Refrain from using the REST API for executing numerous SQL queries. It is intended solely for cluster maintenance.
- [Optional] When **INSERT INTO SELECT** statements exceed 100 million, divide them into smaller batches for execution. To expedite data import during idle cluster resources, adjust concurrency settings.

  For example, setting **parallel_fragment_exec_instance_num** to 8, which is half the CPU cores on a single Backend (BE), is recommended.
- [Mandatory] For result sets larger than 50,000, utilize JDBC Catalog or the **outfile** method for export to prevent overloading Frontend (FE) resources and ensure cluster stability.
  - Employ pagination (offset limit) with an **ORDER BY** clause for interactive queries.
  - Consider **outfile** or **export** methods when exporting data to third parties.
- [Mandatory] Utilize Colocation Join for joining more than two tables with over 300 million records.
- [Mandatory] Avoid using **SELECT *** for querying large tables with hundreds of millions of records and specify the required fields instead.
  - Use the SQL Block method to forbid broad queries.
  - For high-concurrency point queries, consider enabling row storage (version 2.*x*).
  - Use PreparedStatement for queries.
- [Mandatory] Specify bucket conditions when querying tables with over 100 million records.
- [Optional] void using **OR** as a JOIN condition.
- [Optional] Narrow down the data range before sorting large datasets (whose records exceeds 500 million) to enhance performance.

  For example, instead of **FROM table ORDER BY datetime DESC LIMIT 10**, use **FROM table WHERE datetime='2023-10-20' ORDER BY datetime DESC LIMIT 10**.

2.2.5 Doris Data Import Suggestions

This topic describes the technical suggestions for importing Doris data.

Data Import

- [Optional] When Flink writes data to Doris in real time, set the checkpoint based on the data volume of each batch. If the data volume of each batch is

too small, a large number of small files will be generated. The recommended value is 60s.

- [Optional] Import data in batches at a low frequency. The average interval for importing a single table must be greater than 30s. Import 10000 to 100000 rows of data each time at a recommended interval of 60s.

2.2.6 Doris Partition Rules

Partitions are used to divide data into different intervals. Logically, an original table is divided into multiple sub-tables. Data can be easily managed by partition.

- You can specify one or more columns as the partitioning columns, but they have to be KEY columns. The usage of multi-column partitions is described further below.
- Regardless of the type of the partitioning columns, double quotes are required for partition values.
- Theoretically, there is no upper limit on the number of partitions.
- If users create a table without specifying the partitions, the system will automatically generate a partition with the same name as the table. This Partition contains all data in the table and is neither visible to users nor modifiable.
- Partitions should not have overlapping ranges.

Range Partitioning

Partitioning columns are usually time columns for easy management of old and new data.

Range partitioning supports specifying only the upper bound by **VALUES LESS THAN (...)**. The system will use the upper bound of the previous partition as the lower bound of the next partition, and generate a left-closed right-open interval.

- The following takes the VALUES [...) method as an example since it is more comprehensible. It shows how the partition ranges change as we use the **VALUES LESS THAN (...)** statement to add or delete partitions.

```
CREATE TABLE IF NOT EXISTS example_db.expamle_range_tbl
   (
      `user_id` LARGEINT NOT NULL COMMENT "User ID",
      `date` DATE NOT NULL COMMENT "Data import date and time",
      `timestamp` DATETIME NOT NULL COMMENT "Data import timestamp",
      `city` VARCHAR(20) COMMENT "City where the user locates",
      `age` SMALLINT COMMENT "User age",
      `sex` TINYINT COMMENT "User gender",
      `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "Last visit date of
the user",
      `cost` BIGINT SUM DEFAULT "0" COMMENT "Total consumption",
      `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum residence time",
      `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum residence time",
   )
   ENGINE=OLAP
   AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
   PARTITION BY RANGE(`date`)
   (
      PARTITION `p201701` VALUES LESS THAN ("2017-02-01"),
      PARTITION `p201702` VALUES LESS THAN ("2017-03-01"),
      PARTITION `p201703` VALUES LESS THAN ("2017-04-01")
   )
   DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
```



```
PROPERTIES
(
   "replication_num" = "3"
);
```

- View the partitions.

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+---------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------+----------------+---------
+---------------+--------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                           | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+---------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------+----------------+---------
+---------------+--------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
| 16040       | p201701       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3             |
HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16041       | p201702       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3             |
HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16042       | p201703       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-03-01]; ..types: [DATE]; keys: [2017-04-01]; ) | user_id         | 16      | 3             |
HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    | false      |
tag.location.default: 3 |
+-------------+---------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------+----------------+---------
+---------------+--------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
3 rows in set (0.01 sec)
```

- Add partition **p201705**.

```
mysql> alter table example_db.expamle_range_tbl add partition p201705 VALUES LESS THAN
("2017-06-01");
Query OK, 0 rows affected (0.02 sec)
```

View the partitions.

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+---------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------+----------------+---------
+---------------+--------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                           | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+---------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------+----------------+---------
+---------------+--------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
| 16040       | p201701       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3             |
HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16041       | p201702       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3             |
HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16042       | p201703       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-03-01]; ..types: [DATE]; keys: [2017-04-01]; ) | user_id         | 16      | 3             |
HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16237       | p201705       | 1             | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3             |
```



```
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
+------------+--------------+---------------+--------------------+--------+-------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+--------------------+-------------------------+----------
+-----------+------------------------+
4 rows in set (0.00 sec)
```

- Delete partition **p201703**.

```
mysql> alter table example_db.expamle_range_tbl drop partition p201703;
Query OK, 0 rows affected (0.01 sec)
```

View the partitions.

```
mysql> show partitions from example_db.expamle_range_tbl;
+------------+--------------+---------------+--------------------+--------+-------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+--------------------+-------------------------+----------
+-----------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                 | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+------------+--------------+---------------+--------------------+--------+-------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+--------------------+-------------------------+----------
+-----------+------------------------+
| 16040       | p201701       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3             |
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
| 16041       | p201702       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3             |
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
| 16237       | p201705       | 1             | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3             |
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
+------------+--------------+---------------+--------------------+--------+-------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+--------------------+-------------------------+----------
+-----------+------------------------+
3 rows in set (0.00 sec)
```

NOTE

Note that the partition range of p201702 and p201705 has not changed, and there is a gap between the two partitions: [2017-03-01, 2017-04-01). That means, if the imported data is within this gap range, the import would fail.

- Delete **partition p201702**.

```
mysql> alter table example_db.expamle_range_tbl drop partition p201702;
Query OK, 0 rows affected (0.00 sec)
```

View the partitions.

```
mysql> show partitions from example_db.expamle_range_tbl;
+------------+--------------+---------------+--------------------+--------+-------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+--------------------+-------------------------+----------
+-----------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                 | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+------------+--------------+---------------+--------------------+--------+-------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+--------------------+-------------------------+----------
+-----------+------------------------+
| 16040       | p201701       | 1             | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3             |
```

```
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
| 16237      | p201705      | 1            | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3             |
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
+-------------+---------------+----------------+---------------------+--------+--------------
+-------------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
2 rows in set (0.00 sec)
```

NOTE

The gap range expands to: [2017-02-01, 2017-04-01)

- Add partition **p201702new VALUES LESS THAN ("2017-03-01")**.

```
mysql> alter table example_db.expamle_range_tbl add partition p201702new VALUES LESS THAN
("2017-03-01");
Query OK, 0 rows affected (0.01 sec)
```

View the partitions.

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+---------------+----------------+---------------------+--------+--------------
+-------------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                 | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+---------------+----------------+---------------------+--------+--------------
+-------------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
| 16040      | p201701      | 1            | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3             |
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
| 16302      | p201702new    | 1            | 2023-04-11 08:14:25 | NORMAL | date         | [types:
[DATE]; keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3             |
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
| 16237      | p201705      | 1            | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3             |
HDD          | 9999-12-31 15:59:59 |                  | NULL                 | 0.000    | false      |
tag.location.default: 3 |
+-------------+---------------+----------------+---------------------+--------+--------------
+-------------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+--------------------------+----------
+------------+-------------------------+
3 rows in set (0.00 sec)
```

NOTE

In summary, the deletion of a partition does not change the range of the existing partitions, but might result in gaps. When a partition is added via the **VALUES LESS THAN** statement, the lower bound of one partition is the upper bound of its previous partition.

- Multi-column partitioning

  In addition to the single-column partitioning mentioned above, range partitioning also supports multi-column partitioning. Examples are as follows:

```
CREATE TABLE IF NOT EXISTS example_db.expamle_range_multi_partiton_key_tbl
(
`user_id` LARGEINT NOT NULL COMMENT " User ID",
`date` DATE NOT NULL COMMENT "Data import date and time",
`timestamp` DATETIME NOT NULL COMMENT "Data import timestamp",
```



```
`city` VARCHAR(20) COMMENT "City where the user locates",
`age` SMALLINT COMMENT "User age",
`sex` TINYINT COMMENT "User gender",
`last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "Last visit date of
the user",
`cost` BIGINT SUM DEFAULT "0" COMMENT "Total consumption",
`max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum residence time"
`min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum residence time"
)
ENGINE=OLAP
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY RANGE(`date`, `user_id`)
(
PARTITION `p201701_1000` VALUES LESS THAN ("2017-02-01", "1000"),
PARTITION `p201702_2000` VALUES LESS THAN ("2017-03-01", "2000"),
PARTITION `p201703_all`  VALUES LESS THAN ("2017-04-01")
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
"replication_num" = "3"
);
```

In the above example, we specify **date** (DATE type) and **user_id** (INT type) as the partitioning columns, so the resulting partitions will be as follows:

```
mysql> show partitions from example_db.expamle_range_multi_partiton_key_tbl;
+-------------+---------------+----------------+---------------------+--------+---------------
+-------------------------------------------------------------------------------------------------------------------------------
-------------------------+-----------------+---------+----------------+---------------+---------------------
+---------------------+-------------------------+----------+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey  |
Range                                                                                                                          |
DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        |
RemoteStoragePolicy | LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+---------------+----------------+---------------------+--------+---------------
+-------------------------------------------------------------------------------------------------------------------------------
-------------------------+-----------------+---------+----------------+---------------+---------------------
+---------------------+-------------------------+----------+------------+-------------------------+
| 16367       | p201701_1000  | 1              | 2023-04-11 08:28:12 | NORMAL | date, user_id | [types:
[DATE, LARGEINT]; keys: [0000-01-01, -170141183460469231731687303715884105728]; ..types:
[DATE, LARGEINT]; keys: [2017-02-01, 1000]; ) | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 3 |
| 16368       | p201702_2000  | 1              | 2023-04-11 08:28:12 | NORMAL | date, user_id | [types:
[DATE, LARGEINT]; keys: [2017-02-01, 1000]; ..types: [DATE, LARGEINT]; keys: [2017-03-01,
2000]; )                                | user_id         | 16      | 3              | HDD           | 9999-12-31
15:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 3 |
| 16369       | p201703_all   | 1              | 2023-04-11 08:28:12 | NORMAL | date, user_id | [types:
[DATE, LARGEINT]; keys: [2017-03-01, 2000]; ..types: [DATE, LARGEINT]; keys: [2017-04-01,
-170141183460469231731687303715884105728]; ) | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 3 |
+-------------+---------------+----------------+---------------------+--------+---------------
+-------------------------------------------------------------------------------------------------------------------------------
-------------------------+-----------------+---------+----------------+---------------+---------------------
+---------------------+-------------------------+----------+------------+-------------------------+
3 rows in set (0.00 sec)
```

NOTE

Note that in the last partition, the user only specifies the partition value of the **date** column, so the system fills in **MIN_VALUE** as the partition value of the **user_id** column by default. When data is imported, the system will compare them with the partition values in order, and put the data in their corresponding partitions.

Examples are as follows:

```
* Data  -->  Partition
* 2017-01-01, 200     --> p201701_1000
* 2017-01-01, 2000    --> p201701_1000
* 2017-02-01, 100     --> p201701_1000
* 2017-02-01, 2000    --> p201702_2000
```



```
* 2017-02-15, 5000    --> p201702_2000
* 2017-03-01, 2000    --> p201703_all
* 2017-03-10, 1       --> p201703_all
* 2017-04-01, 1000    --> unable to import
* 2017-05-01, 1000    --> unable to import
```

Verification method:

Insert a piece of data and check the partition to which the data is stored. Fields **VisibleVersionTime** and **VisibleVersion** are updated in the partition where the inserted data is located.

```
insert into example_db.expamle_range_multi_partiton_key_tbl values (200, '2017-01-01', '2017-01-01 12:00:05', 'A', 25, 1, '2017-01-01 12:00:05', 100, 30, 10);
insert into example_db.expamle_range_multi_partiton_key_tbl values (2000, '2017-01-01', '2017-01-01 16:10:05', 'B', 33, 1, '2017-01-01 16:10:05', 800, 50, 1);
insert into example_db.expamle_range_multi_partiton_key_tbl values (200, '2017-02-01', '2017-01-01 16:10:05', 'C', 22, 0, '2017-02-01 16:10:05', 80, 200, 1);
show partitions from example_db.expamle_range_multi_partiton_key_tbl\G
```

List Partitioning

- The partitioning columns support the BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DATE, DATETIME, CHAR, VARCHAR data types, and the partition values are enumeration values. Partitions can be only hit if the data is one of the enumeration values in the target partition.
- List partitioning supports using **VALUES IN (...)** to specify the enumeration values contained in each partition.
- The following example illustrates how partitions change when adding or deleting a partition.

```
CREATE TABLE IF NOT EXISTS example_db.expamle_list_tbl
(
`user_id` LARGEINT NOT NULL COMMENT " User ID",
`date` DATE NOT NULL COMMENT "Data import date and time",
`timestamp` DATETIME NOT NULL COMMENT "Data import timestamp",
`city` VARCHAR(20) NOT NULL COMMENT "City where the user locates",
`age` SMALLINT COMMENT "User age",
`sex` TINYINT COMMENT "User gender",
`last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "Last visit date of the user",
`cost` BIGINT SUM DEFAULT "0" COMMENT "Total consumption",
`max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum residence time"
`min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum residence time"
)
ENGINE=olap
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY LIST(`city`)
(
PARTITION `p_cn` VALUES IN ("A", "B", "F"),
PARTITION `p_usa` VALUES IN ("G", "H"),
PARTITION `p_jp` VALUES IN ("I")
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
"replication_num" = "3"
);
```

- As shown in the preceding table, when the table is created, the following three partitions are automatically created.

```
mysql> show partitions from
example_db.expamle_list_tbl;
                              +-------------+---------------+----------------+---------------------+--------
+--------------+-----------------------------------------------------------------------------------------------
+-----------------+---------+----------------+---------------+---------------------+--------------------
+-------------------------+----------+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
```



```
Range                                                                | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+---------------+----------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------+-----------------
+---------+----------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+------------------------+
| 16764       | p_cn          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [A]; , types: [VARCHAR]; keys: [B]; , types: [VARCHAR]; keys: [F]; ] | user_id         | 16      |
3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false
| tag.location.default: 3 |
| 16765       | p_usa         | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types:
[VARCHAR]; keys: [G]; , types: [VARCHAR]; keys: [H]; ]                               | user_id         | 16      |
3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false
| tag.location.default: 3 |
| 16766       | p_jp          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [I]; ]                                                   | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 3 |
+-------------+---------------+----------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------+-----------------
+---------+----------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+------------------------+
3 rows in set (0.00 sec)
```

- Add partition **p_uk VALUES IN ("L")**.

```
mysql> alter table example_db.expamle_list_tbl add partition p_uk VALUES IN ("L");
Query OK, 0 rows affected (0.01 sec)
```

  View the partitions.

```
mysql> show partitions from example_db.expamle_list_tbl;
+-------------+---------------+----------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------+-----------------
+---------+----------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+---------------+----------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------+-----------------
+---------+----------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+------------------------+
| 16764       | p_cn          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [A]; , types: [VARCHAR]; keys: [B]; , types: [VARCHAR]; keys: [F]; ] | user_id         | 16      |
3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false
| tag.location.default: 3 |
| 16765       | p_usa         | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types:
[VARCHAR]; keys: [G]; , types: [VARCHAR]; keys: [H]; ]                               | user_id         | 16      |
3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false
| tag.location.default: 3 |
| 16766       | p_jp          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [I]; ]                                                   | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 3 |
| 16961       | p_uk          | 1              | 2023-04-11 09:24:39 | NORMAL | city         | [types: [VARCHAR];
keys: [L]; ]                                                   | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 3 |
+-------------+---------------+----------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------+-----------------
+---------+----------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+------------------------+
4 rows in set (0.00 sec)
```

- Delete partition **p_jp**.

```
mysql> alter table example_db.expamle_list_tbl drop partition p_jp;
Query OK, 0 rows affected (0.01 sec)
```

  View the partitions.

```
mysql> show partitions from example_db.expamle_list_tbl;
+-------------+---------------+----------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------+-----------------
```

```
+---------+---------------+---------------+--------------------+--------------------+-------------------------
+----------+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+---------------+---------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------------+-----------------
+---------+---------------+---------------+--------------------+--------------------+-------------------------
+----------+------------+-------------------------+
| 16764       | p_cn          | 1             | 2023-04-11 09:21:34 | NORMAL | city          | [types: [VARCHAR];
keys: [A]; , types: [VARCHAR]; keys: [B]; , types: [VARCHAR]; keys: [F]; ] | user_id         | 16      |
3             | HDD           | 9999-12-31 15:59:59 |                    | NULL                  | 0.000    | false
| tag.location.default: 3 |
| 16765       | p_usa         | 1             | 2023-04-11 09:21:34 | NORMAL | city          | [types:
[VARCHAR]; keys: [G]; , types: [VARCHAR]; keys: [H]; ]                          | user_id         | 16      |
3             | HDD           | 9999-12-31 15:59:59 |                    | NULL                  | 0.000    | false
| tag.location.default: 3 |
| 16961       | p_uk          | 1             | 2023-04-11 09:24:39 | NORMAL | city          | [types: [VARCHAR];
keys: [L]; ]                                                | user_id         | 16      | 3             | HDD           |
9999-12-31 15:59:59 |                    | NULL                  | 0.000    | false      | tag.location.default: 3 |
+-------------+---------------+---------------+--------------------+--------+--------------
+-----------------------------------------------------------------------------------------------------+-----------------
+---------+---------------+---------------+--------------------+--------------------+-------------------------
+----------+------------+-------------------------+
3 rows in set (0.00 sec)
```

- List partitioning also supports multi-column partitioning. Examples are as follows:

```
CREATE TABLE IF NOT EXISTS example_db.expamle_list_multi_partiton_key_tbl
(
`user_id` LARGEINT NOT NULL COMMENT " User ID",
`date` DATE NOT NULL COMMENT "Data import date and time",
`timestamp` DATETIME NOT NULL COMMENT "Data import timestamp",
`city` VARCHAR(20) NOT NULL COMMENT "City where the user locates",
`age` SMALLINT COMMENT "User age",
`sex` TINYINT COMMENT "User gender",
`last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "Last visit date of
the user",
`cost` BIGINT SUM DEFAULT "0" COMMENT "Total consumption",
`max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum residence time"
`min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum residence time"
)
ENGINE=olap
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY LIST(`user_id`, `city`)
(
PARTITION `p1_city` VALUES IN (("1", "A"), ("1", "B")),
PARTITION `p2_city` VALUES IN (("2", "A"), ("2", "B")),
PARTITION `p3_city` VALUES IN (("3", "A"), ("3", "B"))
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
"replication_num" = "3"
);
```

In the above example, we specify **user_id** (INT type) and **city** (VARCHAR type) as the partitioning columns, so the resulting partitions will be as follows:

```
mysql> show partitions from example_db.expamle_list_multi_partiton_key_tbl;
+-------------+---------------+---------------+--------------------+--------+---------------
+-----------------------------------------------------------------------------------------------------+-----------------
+---------+---------------+---------------+--------------------+--------------------+-------------------------
+----------+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey  |
Range                                                                | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
```



```
+-------------+---------------+----------------+---------------------+--------+---------------
+-------------------------------------------------------------------------------------------+-----------------
+---------+---------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+------------------------+
| 17026       | p1_city       | 1             | 2023-04-11 09:31:33 | NORMAL | user_id, city | [types:
[LARGEINT, VARCHAR]; keys: [1, A]; , types: [LARGEINT, VARCHAR]; keys: [1, B]; ] | user_id         |
16      | 3             | HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    |
false      | tag.location.default: 3 |
| 17027       | p2_city       | 1             | 2023-04-11 09:31:33 | NORMAL | user_id, city | [types:
[LARGEINT, VARCHAR]; keys: [2, A]; , types: [LARGEINT, VARCHAR]; keys: [2, B]; ] | user_id         |
16      | 3             | HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    |
false      | tag.location.default: 3 |
| 17028       | p3_city       | 1             | 2023-04-11 09:31:33 | NORMAL | user_id, city | [types:
[LARGEINT, VARCHAR]; keys: [3, A]; , types: [LARGEINT, VARCHAR]; keys: [3, B]; ] | user_id         |
16      | 3             | HDD           | 9999-12-31 15:59:59 |                    | NULL                    | 0.000    |
false      | tag.location.default: 3 |
+-------------+---------------+----------------+---------------------+--------+---------------
+-------------------------------------------------------------------------------------------+-----------------
+---------+---------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+------------------------+
3 rows in set (0.00 sec)
```

- When data is imported, the system will compare them with the partition values in order, and put the data in their corresponding partitions. Examples are as follows:

```
* Data  --->  Partition
* 1, A     ---> p1_city
* 1, B    ---> p1_city
* 2, B    ---> p2_city
* 3, A    ---> p3_city
* 1, M    ---> Unable to import
* 4, A    ---> Unable to import
```

  Verification method:

  Insert a piece of data and check the partition to which the data is stored. Fields **VisibleVersionTime** and **VisibleVersion** are updated in the partition where the inserted data is located.

```
INSERT INTO example_db.expamle_list_multi_partiton_key_tbl values (1, '2017-01-01', '2017-01-01
12:00:05', 'A', 25, 1, '2017-01-01 12:00:05', 100, 30, 10);
```

  Insert a piece of data into the partition and check the corresponding fields.

```
mysql> SHOW partitions from example_db.expamle_list_multi_partiton_key_tbl\G
*************************** 1. row ***************************
            PartitionId: 17026
          PartitionName: p1_city
         VisibleVersion: 3
     VisibleVersionTime: 2023-04-11 09:42:34
                  State: NORMAL
          PartitionKey: user_id, city
                  Range: [types: [LARGEINT, VARCHAR]; keys: [1, A]; , types: [LARGEINT, VARCHAR]; keys:
[1, B]; ]
        DistributionKey: user_id
                Buckets: 16
         ReplicationNum: 3
          StorageMedium: HDD
           CooldownTime: 9999-12-31 15:59:59
    RemoteStoragePolicy:
LastConsistencyCheckTime: NULL
               DataSize: 9.340 KB
             IsInMemory: false
      ReplicaAllocation: tag.location.default: 3
*************************** 2. row ***************************
            PartitionId: 17027
          PartitionName: p2_city
         VisibleVersion: 1
     VisibleVersionTime: 2023-04-11 09:31:33
                  State: NORMAL
          PartitionKey: user_id, city
```



```
                Range: [types: [LARGEINT, VARCHAR]; keys: [2, A]; , types: [LARGEINT, VARCHAR]; keys:
[2, B]; ]
        DistributionKey: user_id
                Buckets: 16
         ReplicationNum: 3
          StorageMedium: HDD
           CooldownTime: 9999-12-31 15:59:59
     RemoteStoragePolicy:
LastConsistencyCheckTime: NULL
               DataSize: 0.000
             IsInMemory: false
      ReplicaAllocation: tag.location.default: 3
*************************** 3. row ***************************
           PartitionId: 17028
         PartitionName: p3_city
        VisibleVersion: 1
    VisibleVersionTime: 2023-04-11 09:31:33
                 State: NORMAL
          PartitionKey: user_id, city
                Range: [types: [LARGEINT, VARCHAR]; keys: [3, A]; , types: [LARGEINT, VARCHAR]; keys:
[3, B]; ]
        DistributionKey: user_id
                Buckets: 16
         ReplicationNum: 3
          StorageMedium: HDD
           CooldownTime: 9999-12-31 15:59:59
     RemoteStoragePolicy:
LastConsistencyCheckTime: NULL
               DataSize: 0.000
             IsInMemory: false
      ReplicaAllocation: tag.location.default: 3
3 rows in set (0.00 sec)
```

2.2.7 Doris Bucketing Rules

Data is divided into different buckets based on the hash values of bucketing columns.

- If you use the Partition method, the **DISTRIBUTED ...** statement will describe how data is divided among partitions. If you do not use the Partition method, that statement will describe how data of the whole table is divided.
- You can specify multiple columns as the bucketing columns. In AGGREGATE KEY and UNIQUE KEY models, bucketing columns must be Key columns. In the DUPLICATE model, bucketing columns can be Key columns and Value columns. Bucketing columns can either be partitioning columns or not.
- The choice of bucketing columns is a trade-off between query throughput and query concurrency.
  - If you choose to specify multiple bucketing columns, the data will be more evenly distributed. However, if the query condition does not include the equivalent conditions for all bucketing columns, the system will scan all buckets, largely increasing the query throughput and decreasing the latency of a single query. This method is suitable for high-throughput and low-concurrency query scenarios.
  - If you choose to specify only one or a few bucketing columns, point queries might scan only one bucket. when multiple point queries are performed concurrently, they might scan various buckets, with no interaction between the I/O operations (especially when the buckets are stored on various disks). This approach is suitable for high-concurrency point query scenarios.

- AutoBucket: Calculates the number of partition buckets based on the amount of data. For partitioned tables, you can determine a bucket based on the amount of data, the number of machines, and the number of disks in the historical partition.
- Theoretically, there is no upper limit on the number of buckets.

2.2.8 Rules for the Number and Data Volume for Partitions and Buckets

Recommendations on the Number and Data Volume for Partitions and Buckets

- The total number of tablets in a table is equal to the product of partition number and bucket number.
- The recommended number of tablets in a table, regardless of capacity expansion, is slightly more than the number of disks in the entire cluster.
- The data volume of a single tablet does not have an upper or lower limit theoretically, but is recommended to be in the range of 1 GB to 10 GB. Overly small data volume of a single tablet can impose a stress on data aggregation and metadata management, while overly large data volume can cause trouble in data migration and completion, and increase the cost of Schema Change or Rollup operation failures (These operations are performed on the Tablet level).
- For the tablets, if you cannot have the ideal data volume and the ideal quantity at the same time, it is advised to prioritize the ideal data volume.
- Upon table creation, you specify the same number of buckets for each partition. However, when dynamically increasing partitions (**ADD PARTITION**), you can specify the number of buckets for the new partitions separately. This feature can help you cope with data reduction or expansion.
- Once the number of buckets for a partition is specified, it cannot be changed. Therefore, when determining the number of Buckets, you need to consider the need of cluster expansion in advance. For example, if there are only 3 hosts, and each host has only 1 disk, and you have set the number of Buckets is only set to 3 or less, then no amount of newly added machines can increase concurrency.
- Assume that there are 10 BEs and each BE has one disk. If the total size of a table is 500 MB, you can consider dividing it into 4 to 8 tablets.

  5 GB: 8 to 16 tablets

  50 GB: 32 tablets

  500 GB: You can consider dividing it into partitions, with each partition about 50 GB in size, and 16 to 32 tablets per partition.

  5 TB: You can consider dividing it into partitions, with each partition about 50 GB in size, and 16 to 32 tablets per partition.

Settings and Usage Scenarios of Random Distribution

- If the OLAP table does not have columns of REPLACE type, set the data bucketing mode of the table to RANDOM. This can avoid severe data skew. (When loading data into the partition corresponding to the table, each batch of data in a single load task will be written into a randomly selected tablet).

- When the bucketing mode of the table is set to RANDOM, since there are no specified bucketing columns, it is impossible to query only a few buckets, so all buckets in the hit partition will be scanned when querying the table. Thus, this setting is only suitable for aggregate query analysis of the table data as a whole, but not for high-concurrency point queries.
- If the data distribution of the OLAP table is Random Distribution, you can set load to single tablet to true when importing data. In this way, when importing large amounts of data, in one task, data will be only written in one tablet of the corresponding partition. This can improve both the concurrency and throughput of data import and reduce write amplification caused by data import and compaction, and thus, ensure cluster stability.

Composite Partitioning vs Single Partitioning

- Compound partitioning
  - The first layer of data partitioning is called Partition. You can specify a dimension column as the partitioning column (currently only supports columns of INT and TIME types), and specify the value range of each partition.
  - The second layer is called Distribution, which means bucketing. You can perform HASH distribution on data by specifying the number of buckets and one or more dimension columns as the bucketing columns, or perform random distribution on data by setting the mode to Random Distribution.

  NOTE

  Compound partitioning is recommended for the following scenarios:

  - Scenarios with time dimensions or similar dimensions with ordered values, which can be used as partitioning columns. The partitioning granularity can be evaluated based on data import frequency, data volume, etc.
  - Scenarios with a need to delete historical data: If, for example, you only need to keep the data of the last N days), you can use compound partitioning so you can delete historical partitions. To remove historical data, you can also send a **DELETE** statement within the specified partition.
  - Scenarios with a need to avoid data skew. You can specify the number of buckets individually for each partition. For example, if you choose to partition the data by day, and the data volume per day varies greatly, you can customize the number of buckets for each partition. For the choice of bucketing column, it is advised to select the column(s) with variety in values.

- Single partitioning

  You can also choose single partitioning, which is about HASH distribution.

3 ClickHouse Application Development Guide

3.1 Preparing a ClickHouse Application Development Environment

3.1.1 Preparing the ClickHouse Development and Runtime Environment

Preparing a Development Environment

Table 1 lists the development and running environment to be prepared for application development.

Table 3-1 Development environment

Item	Description
OS	• Development environment: Windows 7 or later. • Running environment: Linux If the program needs to be commissioned locally, the running environment must be able to communicate with network on the cluster service plane.
JDK	Install JDK 1.8.0_272.

Item	Description
IntelliJ IDEA installation and configuration	Basic configuration of the development environment. The version must be 2019.1 or other compatible versions. **NOTE** • If you are using an IBM JDK, ensure that the JDK configured in IntelliJ IDEA is the IBM JDK. • If you are using an Oracle JDK, ensure that the JDK configured in IntelliJ IDEA is the Oracle JDK. • If you are using an open JDK, ensure that the JDK configured in IntelliJ IDEA is the Open JDK. • Do not use the same workspace and the sample project in the same path for different IntelliJ IDEA projects.
Maven installation	Basic configuration of the development environment. It can be used for project management throughout the lifecycle of software development.
Development user	Prepare the ClickHouse cluster user for application development and grant permissions to the user.
7-zip	Tool used to decompress ***.zip** and ***.rar** files. **7-Zip 16.04** is supported.

3.1.2 Configuring and Importing a ClickHouse Sample Project

Background Information

Obtain the ClickHouse development sample project and import the project to IntelliJ IDEA to learn the sample project.

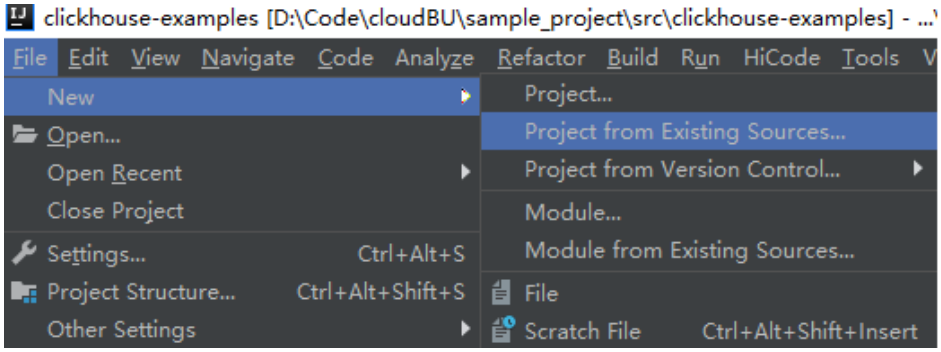
Scenario

ClickHouse provides sample projects for multiple scenarios to help you quickly learn ClickHouse projects.

Procedure

Step 1 In the application development environment, import the sample project to the IntelliJ IDEA development environment.

1. On the IDEA page, choose **File** > **New** > **Project from Existing Sources**.



2. In the displayed **Select File or Directory to Import** dialog box, select the **pom.xml** file in the **clickhouse-examples** folder and click **OK**.
3. Confirm subsequent configurations and click **Next**. If there is no special requirement, use the default values.
4. Select the recommended JDK version and click **Finish**.

Step 2 After the project is imported, modify the **clickhouse-example.properties** file in the **conf** directory of the sample project based on the actual environment information.

```
ipList=
sslUsed=false
httpPort=8123
httpsPort=
CLICKHOUSE_SECURITY_ENABLED=false
user=default
password=
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
clickhouse_dataSource_ip_list=ip:8123,ip:8123
native_dataSource_ip_list=ip:9000,ip:9000
```

Table 3-2 Configuration parameters

Parameter	Default Value	Description
iPList	-	Cluster access address list of the clickhouse node. This parameter is mandatory. Log in to the CloudTable console, click the cluster name to go to the cluster details page, and obtain the cluster access address. Use commas (,) to separate multiple addresses, for example, **cloudtable-wlr-cli-server-1-1-***.com,cloudtable-wlr-cli-server-2-1-***.com**.
sslUsed	false	Whether to enable SSL encryption. The default value is **false**.
httpPort	8123	HTTP port number for connection. The value is **8123**.
httpsPort	-	HTTPS port used for connection. The value is 8443.
CLICKHOUSE_SECURITY_ENABLED	false	Whether to enable the security mode. Set this parameter to **false** for a cluster in normal mode.
user	default	Development user prepared in Table 1
password	-	Password of the development user.

Parameter	Default Value	Description
clusterName	default_cluster	ClickHouse logical cluster name. Retain the default value.
databaseName	testdb	Name of the database to be created in the sample code project. You can change the database name based on the site requirements.
tableName	testtb	Name of the table to be created in the sample code project. You can change the table name based on the site requirements.
batchRows	10000	Number of data records written in a batch.
batchNum	10	Total number of batches in which data is written.
clickhouse_dataSource_ip_list	-	IP address and HTTP port set of the ClickHouse node.
native_dataSource_ip_list	-	IP address and TCP port set of the ClickHouse node.

----End

3.2 Developing a ClickHouse Application

3.2.1 ClickHouse Application Scenarios

This section describes the application development in a typical scenario, helping you quickly learn and master the ClickHouse development process and know key functions.

Scenario

Assume that a user needs to develop an application to store or query the name, age, and onboarding date of a person based on specified search criteria. The procedure is as follows:

1. Set up the database connection.
2. Create an information table.
3. Insert data. (Data in the sample code is randomly generated.)
4. Query the data based on specified search criteria.

3.2.2 ClickHouse Development Plan

As an independent DBMS system, ClickHouse allows you to use the SQL language to perform common operations. In the development program example, the clickhouse-jdbc API is used for description.

- **Setting Properties**: Set the parameters for connecting to a ClickHouse service instance.
- **Setting Up a Connection**: Set a connection to the ClickHouse service instance.
- **Creating a Database**: Create a ClickHouse database.
- **Creating a Table**: Create a table in the ClickHouse database.
- **Inserting Data**: Insert data into the ClickHouse table.
- **Querying Data**: Query data in the ClickHouse table.
- **Deleting a Table**: Delete a ClickHouse table.

3.2.3 Configuring ClickHouse Connection Properties

Function Description

You can set the connection properties through the **Properties** parameter.

In the following example, the socket timeout interval is set to 60s and SSL is not used.

Sample Code

```
Properties clickHouseProperties = new Properties();
clickHouseProperties.setProperty(ClickHouseClientOption.CONNECTION_TIMEOUT.getKey(),
Integer.toString(60000));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL.getKey(), Boolean.toString(false));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL_MODE.getKey(), "none");
```

3.2.4 Setting Up a ClickHouse Connection

Function Description

When creating a connection, use **ClickHouseDataSource** to configure the URL and properties used by the connection.

The user and password configured in **clickhouse-example.properties** are used as authentication credentials. ClickHouse performs security authentication on the server with the user and password.

Sample Code

```
ClickHouseDataSource clickHouseDataSource =new ClickHouseDataSource(JDBC_PREFIX +
serverList.get(tries - 1), clickHouseProperties);
connection = clickHouseDataSource.getConnection(user, password);
```

NOTE

There will be huge security risks if the passwords used for authentication are directly written into the code. You are advised to store the password in ciphertext in the configuration file or environment variables and decrypt them when using them.

3.2.5 Creating a ClickHouse Database

Function Description

In the following example, the **on cluster** statement is used to create a database on all Server nodes in the cluster.

The database name is specified by the **databaseName** field in the **clickhouse-example.properties** file.

Sample Code

```
private void createDatabase(String databaseName, String clusterName) throws Exception  {
        String createDbSql = "create database if not exists " + databaseName + " on cluster " + clusterName;
        util.exeSql(createDbSql);
}
```

3.2.6 Creating a ClickHouse Table

Function Description

In the following example, the **on cluster** statement is used to create distributed and local tables on all Server nodes in the cluster.

createSql is used to create a local table, and **createDisSql** is used to create a distributed table based on the local table.

Sample Code

```
private void createTable(String databaseName, String tableName, String clusterName) throws Exception {
    String createSql = "create table " + databaseName + "." + tableName + " on cluster " + clusterName
            + " (name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/" +
databaseName
            + "." + tableName + "'," + "'{replica}') partition by toYYYYMM(date) order by age";
    String createDisSql = "create table " + databaseName + "." + tableName + "_all" + " on cluster " +
clusterName + " as "
            + databaseName + "." + tableName + " ENGINE = Distributed(default_cluster," + databaseName +
"," + tableName + ", rand());";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(createSql);
    sqlList.add(createDisSql);
    util.exeSql(sqlList);
}
```

3.2.7 Inserting ClickHouse Data

Function Description

The following sample code is used to construct sample data and insert data in batches using the **executeBatch ()** method of **PreparedStatement** for **batchNum** times.

The data types are the three fields specified in the created table, which are **String**, **UInt8**, and **Date**.

Sample Code

```
String insertSql = "insert into " + databaseName + "." + tableName + " values (?,?,?)";
PreparedStatement preparedStatement = connection.prepareStatement(insertSql);
```



```
long allBatchBegin = System.currentTimeMillis();
for (int j = 0; j < batchNum; j++) {
    for (int i = 0; i < batchRows; i++) {
        preparedStatement.setString(1, "xxx_" + (i + j * 10));
        preparedStatement.setInt(2, ((int) (Math.random() * 100)));
        preparedStatement.setDate(3, generateRandomDate("2018-01-01", "2021-12-31"));
        preparedStatement.addBatch();
    }
    long begin = System.currentTimeMillis();
    preparedStatement.executeBatch();
    long end = System.currentTimeMillis();
    log.info("Inert batch time is {} ms", end - begin);
}
long allBatchEnd = System.currentTimeMillis();
log.info("Inert all batch time is {} ms", allBatchEnd - allBatchBegin);
```

3.2.8 Querying ClickHouse Data

Function Description

Query statement 1: **querySql1** queries random 10 records in the **tableName** table.

Query statement 2: **querySql2** uses a built-in function to obtain the year and month from the date field in the **tableName** table and then aggregate the data.

Sample Code

```
private void queryData(String databaseName, String tableName) throws Exception {
    String querySql1 = "select * from " + databaseName + "." + tableName + "_all" + " order by age limit
10";
    String querySql2 = "select toYYYYMM(date),count(1) from " + databaseName + "." + tableName + "_all"
+ " group by toYYYYMM(date) order by count(1) DESC limit 10";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(querySql1);
    sqlList.add(querySql2);
    ArrayList<ArrayList<ArrayList<String>>> result = util.exeSql(sqlList);
    for (ArrayList<ArrayList<String>> singleResult : result) {
        for (ArrayList<String> strings : singleResult) {
            StringBuilder stringBuilder = new StringBuilder();
            for (String string : strings) {
                stringBuilder.append(string).append("\t");
            }
            log.info(stringBuilder.toString());
        }
    }
}
```

3.2.9 Deleting a ClickHouse Table

Function Description

Delete the replica table and distributed table created in the new table.

Statement 1: Use **drop table** to delete a local table from the cluster.

Statement 2: Use **drop table** to delete a distributed table from the cluster.

Sample Code

```
private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " +
clusterName;
```



```
    String dropDisTableSql = "drop table if exists " + databaseName + "." + tableName + "_all" + " on
cluster " + clusterName;
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(dropLocalTableSql);
    sqlList.add(dropDisTableSql);
    util.exeSql(sqlList);
}
```

3.3 Commissioning a ClickHouse Application

3.3.1 Commissioning a ClickHouse Application in a Linux Environment

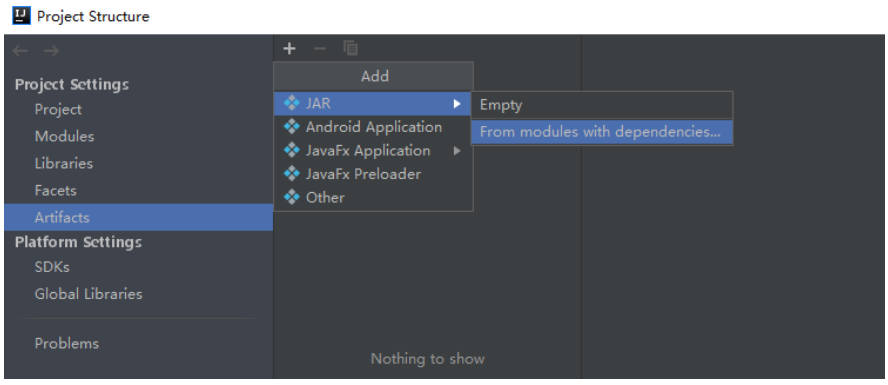
ClickHouse applications can run in a Linux environment. After the application code is developed, you can upload the JAR package to the prepared Linux environment. The environment must be in the same VPC and security group as the ClickHouse cluster to ensure network connectivity.

Prerequisites

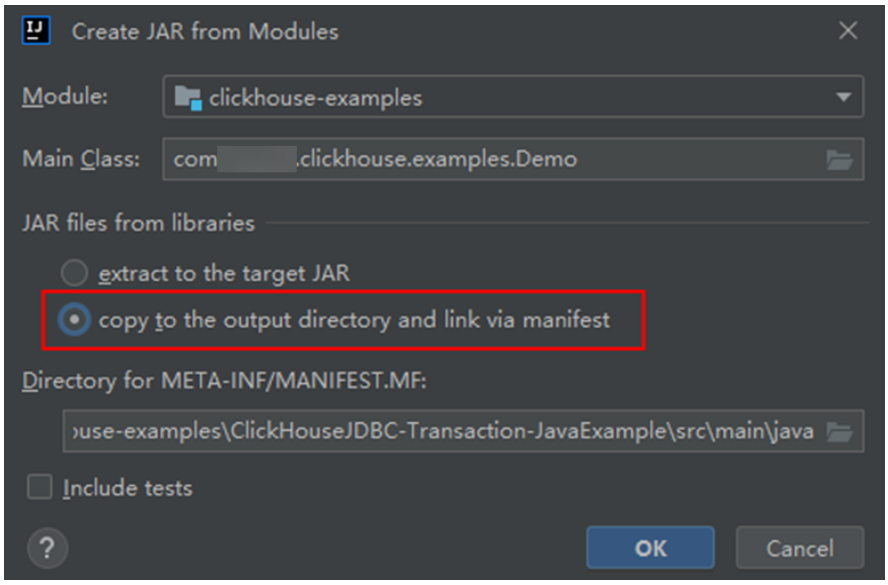
JDK has been installed on Linux. The version must be the same as JDK version of the JAR file exported from IntelliJ IDEA. Java environment variables have been set.

Compiling and Running Applications

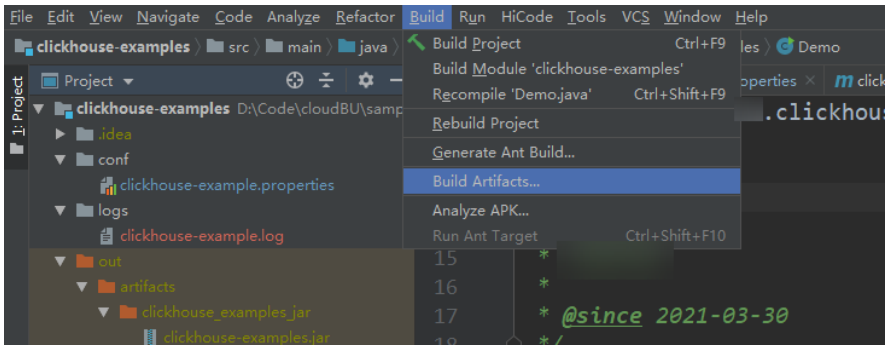
1. Export the JAR file.
   a. Log in to IntelliJ IDEA and choose **File** > **Project Structure** > **Artifacts**.
   b. Click the plus sign (+) and choose **JAR** > **From modules with dependencies**.



   c. Choose **com.xxx.clickhouse.examples.Demo** from the **Main Class** drop-down list and click **OK**.



d. Choose **Build** > **Build Artifacts...**. After the compilation is successful, view and obtain all JAR files in the **clickhouse-examples\out\artifacts\clickhouse_examples_jar** directory.



2. Copy all JAR files in the **clickhouse-examples\out\artifacts\clickhouse_examples.jar** directory and the **conf** folder in the **clickhouse-examples** directory to the same directory of the ECS.
3. Log in to the client node, go to the directory where the JAR file is uploaded, and change the file permission to 700.

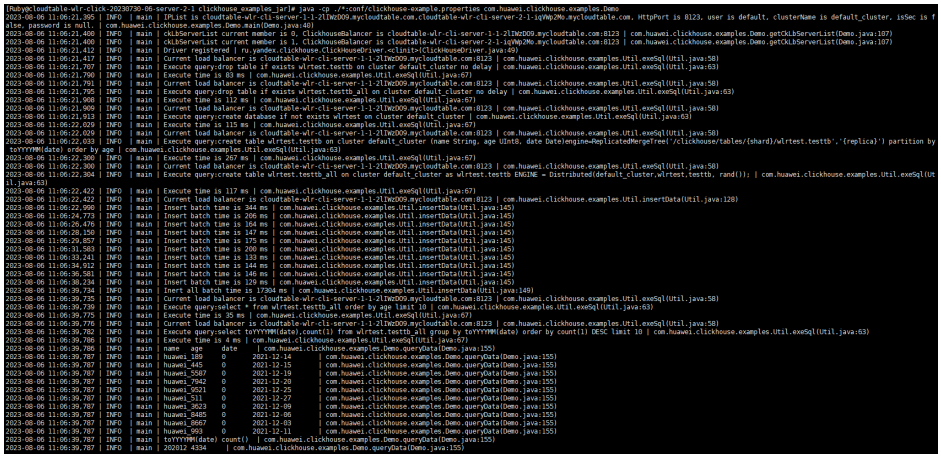
   chmod -R 700 clickhouse-examples.jar
4. In the client directory where **clickhouse_examples.jar** is stored, run the following commands to run the JAR file:

   java -cp ./*:conf/clickhouse-example.properties com.xxx.clickhouse.examples.Demo

Viewing Commissioning Results

If no exception or failure information is displayed, the application running is successful.

Figure 3-1 Run log



3.4 ClickHouse Usage Specifications

3.4.1 ClickHouse Table Creation Rules

This topic describes the rules and suggestions for creating ClickHouse tables.

Table Creation Rules

- [Rule] Do not create service tables in the system database. The system database in ClickHouse is the default database that stores system configurations and metadata. Service-related tables should be created in their own service databases to prevent unnecessary impact on the system database.
- [Rule] Do not use SQL reserved words in database and table names. Ensure that the names are case-sensitive. If reserved keywords must be used, enclose them in double quotation marks or backquote for escape.
- [Rule] Do not use the character type to store time or date data, especially when date fields are used for calculation or comparison.
- [Rule] Do not use the character type to store numeric data, especially when numeric fields are calculated or compared.
- [Suggestion] Avoid using the Nullable column in a table. Instead, use the string "NA".

  When a Nullable column is used in query condition judgments, the system checks whether the column is empty, adding extra calculation overhead. Historical data shows that the query performance of the Nullable type is 20% to 30% slower than that of the String type. Therefore, the Nullable type should be used only when necessary.

3.4.2 ClickHouse Data Writing Rules

This topic describes the rules and suggestions for writing ClickHouse data.

Data Write Rules

- [Rule] Use external modules to ensure idempotence of data import.

  ClickHouse does not support transactional guarantees for data writes. Use external import modules to ensure data idempotence. For example, if a batch fails to import, drop the data of the corresponding partition or clear the imported data, and re-import the data of the partition or batch.

- [Rule] Write data with large batches and a low frequency.

  Each time data is inserted into ClickHouse, one or more part files are generated. If there are too many data parts, the merge pressure increases, potentially causing various exceptions and affecting data insertion. It is recommended that 5,000 to 100,000 rows be written per batch. Adjust the number of rows based on the number of fields to reduce memory and CPU pressure on the node where data is written. Ensure no more than one insertion per second.

- [Suggestion] Insert data into only one partition at a time.

  If data belongs to different partitions, a part file is generated each time data from different partitions is inserted, increasing the total number of parts. It is recommended that data inserted in a batch belong to the same partition.

- [Suggestion] Exercise caution when inserting data into distributed tables in batches.
  - When data is written to a distributed table, it is distributed across all local tables in the cluster. Each local table receives 1/N of the total inserted data. Small batch sizes can lead to numerous data parts, increasing merge pressure and potentially affecting data insertion.
  - Data Consistency: Data is initially written to the host of the distributed table and then asynchronously sent to the host of the local table for storage. There is no consistency check, so if the host of the distributed table fails, data may be lost.
  - Writing data to a distributed table is slower than writing to a local table, as the disk and network I/O of the node handling the distributed table can become a performance bottleneck.
  - The client inserting data cannot detect if the distributed table successfully forwards data to each shard. If forwarding fails, the client continuously retries, consuming CPU resources.
  - Distributed table insertion is suitable only for data deduplication scenarios, where the sharding key forwards data to the same shard for subsequent deduplication queries.

- [Suggestion] Exercise caution when performing **delete** and **update** operations.

  Standard SQL **update** and **delete** operations are synchronous, meaning the server returns the execution result immediately. However, ClickHouse performs these operations asynchronously. When an **update** statement is executed, the server responds immediately, but the data change is queued, potentially leading to overwrites and compromising atomicity. For service scenarios requiring **update** and **delete** operations, use the ReplacingMergeTree, CollapsingMergeTree, and VersionedCollapsingMergeTree engines.

- [Suggestion] Exercise caution when performing the **optimize** operation.

Generally, **optimize** rewrites tables. Perform this operation during off-peak hours to avoid consuming excessive I/O, memory, and CPU resources, which can slow down or disrupt service queries.

- [Suggestion] Run the ALTER TABLE statement for each column individually.
  - Exercise caution when doing delete, update, and mutation operations.

    The update and delete of standard SQL statements are synchronous operations. That is, the client needs to wait for the server to return the execution results (usually an **int** value). In contrast, the update and delete of ClickHouse are asynchronous operations. When an update statement is processed, the server immediately returns the request status: success or fail, while the operation is not complete. At that time, the update request is accepted and queued in the background. As a result, the operation may be overwritten, and atomicity of operations cannot be ensured.

    For scenarios that involve update and delete operations, you are advised to use the ReplacingMergeTree, CollapsingMergeTree, and VersionedCollapsingMergeTree engines. For details, see **Table Engines**.
  - Try to avoid adding or deleting data columns.

    Plan the number of columns for future use, reserve enough columns to avoid a large number of **alter table modify** operations during service running in the production system. Otherwise, unpredictable performance problem and data inconsistency may occur.

3.4.3 ClickHouse Data Query Rules

This topic describes the rules and suggestions for querying ClickHouse data.

Data Query Rules

- [Rule] Avoid using SELECT *; query only the required fields to reduce machine load and improve query performance.

  In OLAP analysis scenarios, wide tables can contain hundreds or thousands of columns, but typically only a few are needed for dimension and metric calculations. Since ClickHouse stores data by column, using SELECT * increases system pressure.
- [Rule] Limit the amount of data returned by queries to save computing resources and reduce network overhead.

  Returning large volumes of data can cause client-side issues like memory overflow. If ClickHouse is used at the frontend and large data queries are necessary, paginate the queries to reduce network bandwidth and computing resource usage.
- [Rule] Join large tables with small tables for associated queries.

  ClickHouse requires multi-table join models to be pre-processed into wide table models. If tables and dimension tables change frequently, use a join model to query the latest data in real-time. Always join a large table with a small table using a join condition. Small tables should range from millions to tens of millions of rows and be filtered based on conditions before joining.
- [Suggestion] Use GLOBAL JOIN/IN instead of common JOIN.

  ClickHouse converts distributed table queries into local table operations across all shards and then summarizes the results. The execution logic of JOIN

and GLOBAL JOIN differs significantly, so use GLOBAL JOIN for querying distributed tables.

- [Rule] Properly use partition fields and index fields in data tables.
  - The MergeTree engine organizes data in partition directories, allowing partitions to skip irrelevant data files during queries, reducing data reading.
  - It sorts data based on the index field and generates sparse indexes based on the **index_granularity** configuration, enabling quick data filtering and improving query performance.
- [Suggestion] Specify the data query scope.

  Use filter criteria and data query periods to narrow the query scope. For example, when querying a specified partition, specify the partition field to reduce the number of files scanned and improve query performance. For a large table with thousands of columns in 700 partitions, querying 70 million records in one partition takes hundreds of milliseconds, while scanning all partitions takes 1-2 seconds. The performance of the former is 20 times higher.
- [Suggestion] Exercise caution when using the final query.

  The FINAL keyword is used at the end of a query statement. For the ReplacingMergeTree engine, if data cannot be completely deduplicated, developers may use FINAL for real-time merge-on-read to ensure no duplicate data exists. Consider using the argMax function or other methods to avoid this issue.
- [Suggestion] Use materialized views to accelerate queries.

  In scenarios with fixed query modes, materialized views can aggregate data in advance, significantly improving performance compared to querying detail tables.
- [Suggestion] Syntax verification is not performed when creating a materialized view. Errors occur only during data insertion or querying. Fully verify materialized views before bringing them online.

3.4.4 Rules for Importing ClickHouse Data to the Database

This topic describes the rules and suggestions for importing ClickHouse data to the database.

Importing Data to the Database

[Suggestion] Avoid creating a ClickHouse Kafka table engine to synchronize data to the ClickHouse. Using the Kafka engine of the ClickHouse may present various performance issues during data import. Experience shows that the application side needs to use Kafka data consumption and write a large amount of data to the ClickHouse, thereby improving the data import performance of the ClickHouse.

