Data Lake Insight

Developer Guide

Issue	01
Date	2021-03-24

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks 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 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 Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Contents

1 DLI Client Operation Guide

1.1 Submitting a Job Using Beeline

Introduction to DLI Beeline

DLI Beeline is a command line tool used to connect to DLI. Powered on DLI JDBC, DLI Beeline provides the functions of SQL command interaction and batch SQL script execution.

Initial Preparations

Before using DLI Beeline, perform the following operations:

1. Getting authorized.

   DLI uses the Identity and Access Management (IAM) to implement fine-grained permissions for your enterprise-level tenants. IAM provides identity authentication, permissions management, and access control, helping you securely access to your public cloud resources.

   With IAM, you can use your public cloud account to create IAM users for your employees, and assign permissions to the users to control their access to specific resource types.

   Currently, roles (coarse-grained authorization) and policies (fine-grained authorization) are supported. For details about permissions and authorization operations, see the **Data Lake Insight User Guide**.

2. Create a queue. Set **Queue Type** to **SQL Queue**, which is the computing resource of the SQL job.

   You can create a queue on the **Overview**, **SQL Editor**, or **Queue Management** page.

   - In the upper right corner of the Dashboard page, click 

     to create a queue.
   - To create a queue on the **Queue Management** page:
     i. In the navigation pane of the DLI management console, choose **Queue Management**.

ii. In the upper right corner of the **Queue Management** page, click 

to create a queue.

- To create a queue on the **SQL Editor** page:

  i. On the top menu bar of the DLI management console, click **SQL Editor**.

  ii. On the left pane of the displayed **Job Editor** page, click  . Click  to the right of **Queues**.

NOTE

If the user who creates the queue is not an administrator, the queue can be used only after being authorized by the administrator. For details about how to assign permissions, see the **Data Lake Insight User Guide**.

Downloading the DLI Client Tool

You can download the DLI client tool from the DLI management console.

Step 1 Log in to the DLI management console.

Step 2 Click **SDK Download** in the **Common Links** area on the right of the **Overview** page.

Step 3 On the **DLI SDK DOWNLOAD** page, click **huaweicloud-dli-clientkit-<version>** to download the DLI client tool.

NOTE

The Beeline client is named **huaweicloud-dli-clientkit-<version>-bin.tar.gz**, which can be used in Linux and depends on JDK 1.8 or later.

----End

Connecting to the DLI Server Using DLI Beeline

Ensure that you have installed JDK of 1.8 or a later version and configured environment variables on the computer where DLI Beeline is installed. You are advised to use DLI Beeline on the computer running the Linux OS.

Step 1 Download and decompress **huaweicloud-dli-clientkit-<version>-bin.tar.gz**. In this step, set **version** to the actual version.

Step 2 Go to the directory where **dli-beeline-<version>-bin.tar.gz** is decompressed. In the directory, there are three subdirectories **bin**, **conf**, and **lib**, which respectively store the execution scripts, configuration files, and dependency packages related to DLI Beeline.

Step 3 Go to the **conf** directory, rename **connection.properties.template** as **connection.properties**, and set connection parameters.

NOTE

For details about the connection parameters, see *Table 2-Database connection parameters* and *Table 3-Attribute items* in **Submitting a Job Using JDBC**.

Step 4 Enter the **bin** directory, start the beeline script to execute the SQL statement, as shown in the following:

```
%bin/beeline
Start Beeline
Welcome to DLI service !
beeline> !connect
Connecting from the default connection.properties
Connecting to jdbc:dli://dli.cn-north-1.myhuaweicloud.com/8fc20d97a4444cafba3c3a8639380003
Connected to: DLI service
jdbc:dli://dli.cn-north-1.myhuaweicloud... (not set)> show databases;
+--------------------+
|    databaseName    |
+--------------------+
| bjhk               |
| db_xd              |
| dimensions_adgame  |
| odbc_db            |
| sdk_db             |
| tpch_csv_1024      |
| tpch_csv_4g_new    |
| tpchnewtest        |
| tpchtest           |
| xunjian            |
+--------------------+
10 rows selected (0.338 seconds)
```

You can also set connection parameters by using the CLI when starting the DLI Beeline script. If the connection parameters are incomplete, DLI Beeline will prompt you to supplement related information.

```
%bin/beeline -u 'jdbc:dli://dli.cn-north-1.myhuaweicloud.com/8fc20d97a4444cafba3c3a8639380003?
authenticationmode=aksk'
Start Beeline
Connecting to jdbc:dli://dli.cn-north-1.myhuaweicloud.com/8fc20d97a4444cafba3c3a8639380003?
usehttpproxy=true;proxyhost=10.186.60.154;proxyport=3128;authenticationmode=aksk
Enter region name: cn-north-1
Enter service name: DLI
Enter access key(AK): <real access key>
Enter secret key(SK): ****************************************
Enter queue name: default
Connected to: DLI service
Welcome to DLI service !
jdbc:dli://dli.cn-north-1.myhuaweicloud... (not set)>
```

----End

Commands Supported by DLI Beeline

DLI Beeline supports a series of commands. Each command starts with an exclamation mark (!). for example,**!. connect**. For details, see **Table 1-1**.

Table 1-1 Commands Supported by DLI Beeline

Command	Description
!connect	This command is used to connect to DLI by setting connection parameters. If no parameters are specified, the default **connection.properties** file is loaded.
!help	This command is used to print the help document of the command line.

Command	Description
!history	This command is used to display the command execution history.
!outputformat	This command is used to set the output format of the query result. The available output formats include table, vertical, csv2, dsv, tsv2, xmlattr, and xmlelements. For details about each format, see **Output Formats of Query Result**.
!properties	This command is used to connect to DLI by loading the **connection.properties** file. This command delivers the same function as the **!connect** command, but allows you to specify the configuration file.
!quit	This command is used to exit the DLI Beeline session.
!run	This command is used to run a SQL statement. The method is as follows: ***!run <scriptfile>***
!save	This command is used to save the current session attributes to the **beeline.properties** file. These attributes will be automatically loaded when DLI Beeline is enabled next time.
!script	This command is used to save the executed commands to a file. Sample code is provided as follows: ***!script /tmp/mysession.script*** After this statement is executed, subsequent commands will be saved to **/tmp/mysession.script**. Run the **!script** command again to end script recording. The recorded commands will be executed again if you run the following command: ***!run /tmp/mysession.script***
!set	This command is used to set the DLI Beeline variables. A command example is provided as follows: ***!set color true*** If no parameter is specified after **!set**, all variable values will be returned.
!sh	This command is used to run a Shell script. Sample code is provided as follows: ***!sh <shellscript>***
!sql	This command is used to run an SQL statement explicitly. In DLI Beeline, a statement without commands is converted into the **!sql** command by default. The SQL statement must end with a semicolon (;). Sample code is provided as follows: ***!sql <sql>***
!dliconf	This command is used to view the user-defined configurations of DLI.

Command Line Options Supported by DLI Beeline

Table 1-2 lists the startup command line options supported by DLI Beeline.

Table 1-2 Command line options supported by DLI Beeline

Command Line Option	Description
-u <database URL>	Indicates the URL for connecting to DLI JDBC. The URL must be enclosed in single quotation marks ('). An example is provided as follows: **beeline –u db_URL**
-e <query>	Indicates the SQL statements to be executed. Multiple statements can be entered, separated by semicolons (;). Each statement must be enclosed in single quotation marks (').
-f <file>	Indicates the script file to be executed.
--dliconf property=value	DLI property to be set.
--property-file=<property-file>	Indicates to obtain the attribute file in a specified mode and connect to DLI.
--help	Indicates to display help information about command line options.

Output Formats of Query Result

DLI Beeline supports multiple output formats for the query result. The output format can be specified by running the **!outputformat** command. DLI Beeline supports the following output formats: table, vertical, csv2, dsv, tsv2, xmlattr, and xmlelements.

- **table**

  The result in the table format is displayed in a table. For example:

  !outputformat table

  select id, value, comment from test_table;

```
+-----+---------+-----------------+
| id  |  value  |     comment     |
+-----+---------+-----------------+
| 1   | Value1  | Test comment 1  |
| 2   | Value2  | Test comment 2  |
| 3   | Value3  | Test comment 3  |
+-----+---------+-----------------+
```

- **vertical**

  In the vertical format, the result data is organized by rows and each attribute is displayed in key-value format. For example:

!outputformat vertical

select id, value, comment from test_table;

```
id       1
value    Value1
comment  Test comment 1
id       2
value    Value2
comment  Test comment 2
id       3
value    Value3
comment  Test comment 3
```

- **csv2**

  Store table data (digits and texts) in plain text and use commas (,) as separators. For example:

  !outputformat csv2

  select id, value, comment from test_table;

  ```
  id,value,comment
  1,Value1,Test comment 1
  2,Value2,Test comment 2
  3,Value3,Test comment 3
  ```

- **dsv**

  Each line stores a record. Fields in each record are separated by tabs. For example:

  !outputformat dsv

  select id, value, comment from test_table;

  ```
  id|value |comment
  1 |Value1|Test comment 1
  2 |Value2|Test comment 2
  3 |Value3|Test comment 3
  ```

- **tsv2**

  Each line stores a record. Fields in each record are separated by spaces. For example:

  !outputformat tsv2

  select id, value, comment from test_table;

  ```
  id value      comment
  1  Value1Test comment 1
  2  Value2Test comment 2
  3  Value3Test comment 3
  ```

- **xmlattr**

  Function used to set attributes in the XML element returned by the SQL query. For example:

  !outputformat xmlattr

  select id, value, comment from test_table;

  ```
  <resultset>
    <result id="1" value="Value1" comment="Test comment 1"/>
    <result id="2" value="Value2" comment="Test comment 2"/>
    <result id="3" value="Value3" comment="Test comment 3"/>
  </resultset>
  ```

- **xmlelements**

  Function for converting a relationship value to an XML element. The format is **<elementName>value</elementName>**. For example:

  !outputformat xmlelements

select id, value, comment from test_table;

```
<resultset>
 <result>
  <id>1</id>
  <value>Value1</value>
  <comment>Test comment 1</comment>
 </result>
 <result>
  <id>2</id>
  <value>Value2</value>
  <comment>Test comment 2</comment>
 </result>
 <result>
  <id>3</id>
  <value>Value3</value>
  <comment>Test comment 3</comment>
 </result>
</resultset>
```

1.2 Using Spark-submit to Submit a Job

Introduction to DLI Spark-submit

DLI Spark-submit is a command line tool used to submit Spark jobs to the DLI server. This tool provides command lines compatible with open-source Spark.

Initial Preparations

Before using DLI Spark-submit, perform the following operations:

1. Getting authorized.

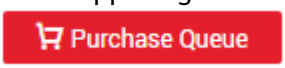
   DLI uses the Identity and Access Management (IAM) to implement fine-grained permissions for your enterprise-level tenants. IAM provides identity authentication, permissions management, and access control, helping you securely access to your public cloud resources.

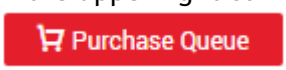
   With IAM, you can use your public cloud account to create IAM users for your employees, and assign permissions to the users to control their access to specific resource types.

   Currently, roles (coarse-grained authorization) and policies (fine-grained authorization) are supported. For details about permissions and authorization operations, see the **Data Lake Insight User Guide**.

2. Create a queue. Set **Queue Type** to **For General Purpose**, that is, the computing resources of the Spark job.

   You can create a queue on the **Overview**, **SQL Editor**, or **Queue Management** page.

   - In the upper right corner of the Dashboard page, click 

     to create a queue.
   - To create a queue on the **Queue Management** page:
     i. In the navigation pane of the DLI management console, choose **Queue Management**.
     ii. In the upper right corner of the **Queue Management** page, click 

     to create a queue.

- To create a queue on the **SQL Editor** page:
  i. On the top menu bar of the DLI management console, click **SQL Editor**.
  ii. On the left pane of the displayed **Job Editor** page, click . Click to the right of **Queues**.

NOTE

If the user who creates the queue is not an administrator, the queue can be used only after being authorized by the administrator. For details about how to assign permissions, see the **Data Lake Insight User Guide**.

Downloading the DLI Client Tool

You can download the DLI client tool from the DLI management console.

Step 1 Log in to the DLI management console.

Step 2 Click **SDK Download** in the **Common Links** area on the right of the **Overview** page.

Step 3 On the **DLI SDK DOWNLOAD** page, click **huaweicloud-dli-clientkit-<version>** to download the DLI client tool.

NOTE

The Beeline client is named **huaweicloud-dli-clientkit-<version>-bin.tar.gz**, which can be used in Linux and depends on JDK 1.8 or later.

----End

Configuring DLI Spark-submit

Ensure that you have installed JDK of 1.8 or a later version and configured environment variables on the computer where spark-submit is installed. You are advised to use spark-submit on the computer running the Linux OS.

Step 1 Download and decompress **huaweicloud-dli-clientkit-<version>-bin.tar.gz**. In this step, set **version** to the actual version.

Step 2 Go to the directory where **dli-clientkit-<version>-bin.tar.gz** is decompressed. In the directory, there are three subdirectories **bin**, **conf**, and **lib**, which respectively store the execution scripts, configuration files, and dependency packages related to **Spark-submit**.

Step 3 Go to the **conf** directory and modify the configuration items in the **client.properties** file. For details about the configuration items, see **Table 1-3**.

Table 1-3 DLI client parameters

Item	Mandatory	Default Value	Description
dliEndPont	No	-	Domain name of DLI. To obtain the endpoint of the region corresponding to DLI, see **Regions and Endpoints**. If this parameter is not set, the program determines the domain name of the region corresponding to HUAWEI CLOUD based on the region parameter.
obsEndPoint	Yes	obs.cn-north-1.myhuaweicloud.com	OBS service domain name. To obtain the endpoint of the region corresponding to DLI, see **Regions and Endpoints**.
bucketName	Yes	-	Name of a bucket on OBS. This bucket is used to store JAR packages, Python program files, and configuration files used in Spark programs.
obsPath	Yes	dli-spark-submit-resources	Directory for storing JAR packages, Python program files, and configuration files on OBS. The directory is in the bucket specified by **Bucket Name**. If the directory does not exist, the program automatically creates it.
localFilePath	Yes	-	The local directory for storing JAR packages, Python program files, and configuration files used in Spark programs. The program automatically uploads the files on which Spark depends to the OBS path and loads them to the resource package on the DLI server.
ak	Yes	-	User's Access Key (AK)
sk	Yes	-	User's Secret Key (SK)
projectId	Yes	-	Project ID used by a user to access DLI.
region	Yes	-	Region of interconnected DLI, for example: **cn-north-1**.

Modify the configuration items in the **spark-defaults.conf** file based on the Spark application requirements. The configuration items are compatible with the open-source Spark configuration items. For details, see the open-source Spark configuration item description.

----End

Using Spark-submit to Submit a Spark Job

Step 1 Go to the **bin** directory of the tool file, run the **spark-submit** command, and carry related parameters.

The command format is as follows:

```
spark-submit [options] <app jar | python file> [app arguments]
```

Table 1-4 DLI Spark-submit parameters

Parameter	Value	Description
--class	<CLASS_NAME>	Name of the main class of the submitted Java or Scala application.
--conf	<PROP=VALUE>	Spark program parameters can be configured in the **spark-defaults.conf** file in the **conf** directory. If both the command and the configuration file are configured, the parameter value specified in the command is preferentially used. **NOTE** If there are multiple **conf** files, the format is **--conf key1=value1 --conf key2=value2**.
--jars	<JARS>	Name of the JAR package on which the Spark application depends. Use commas (,) to separate multiple names. The JAR package must be stored in the local path specified by **localFilePath** in the **client.properties** file in advance.
--name	<NAME>	Name of a Spark application.
--queue	<QUEUE_NAME>	Name of the Spark queue on the DLI server. Jobs are submitted to the queue for execution.
--py-files	<PY_FILES>	Name of the Python program file on which the Spark application depends. Use commas (,) to separate multiple file names. The Python program file must be saved in the local path specified by **localFilePath** in the **client.properties** file in advance.

Parameter	Value	Description
-s,--skip-upload-resources	<all | app | deps>	Specifies whether to skip. Upload the JAR package, Python program file, and configuration file to OBS and load them to the resource list on the DLI server. If related resource files have been loaded to the DLI resource list, skip this step. If this parameter is not specified, all resource files in the command are uploaded and loaded to DLI by default. • **all**: Skips the upload and loading all resource files. • **app**: Skips the upload and loading of Spark application files. • **deps**: skips the upload and loading of all dependent files.
-h,--help	-	Displays command help information.

Command example:

```
./spark-submit --name <name> --queue <queue_name> --class org.apache.spark.examples.SparkPi spark-examples_2.11-2.1.0.luxor.jar 10
./spark-submit --name <name> --queue <queue_name> word_count.py
```

NOTE

To use the DLI queue rather than the existing Spark environment, use **./spark-submit** instead of **spark-submit**.

----End

1.3 Submitting a Job Using JDBC or ODBC

1.3.1 Obtaining the Server Connection Address

Scenario

On DLI, you can connect to the server for data query in the Internet environment. In this case, you need to first obtain the connection information, including the endpoint and project ID by following the following procedure.

Procedure

The format of the address for connecting to DLI is jdbc:dli://<endPoint>/<projectId>. Therefore, you need to obtain the endpoint and project ID.

Obtain the DLI endpoint from **Regions and Endpoints**. Specifically, log in to the public cloud, click your username, and choose **My Credentials** from the short-cut menu. You can obtain the project ID on the **My Credentials** page.

Example: jdbc:dli://dli.cn-north-1.myhuaweicloud.com/96a17d961b84434baec6a58b9e567908

1.3.2 Downloading the JDBC or ODBC Driver Package

Scenario

To connect to DLI using JDBC or ODBC, download the specific driver file on the DLI management console.

Procedure

Step 1 Log in to the DLI management console.

Step 2 Click **SDK Download** in the **Common Links** area on the right of the **Overview** page.

Step 3 On the **DLI SDK DOWNLOAD** page, select a driver and download it.

- JDBC driver package

  For example, click **huaweicloud-dli-jdbc-1.1.2** to download the JDBC driver package of version 1.1.2.

  NOTE

  The JDBC driver package is named **huaweicloud-dli-jdbc-<version>.zip**. It can be used in all versions of all platforms (such as Linux and Windows) and depends on JDK 1.7 or later versions.

  If Maven is used, add the following Maven configuration items on which **huaweicloud-dli-jdbc** depends:

```
<dependency>
  <groupId>com.huawei.dli</groupId>
  <artifactId>huaweicloud-dli-jdbc</artifactId>
  <version>x.x.x</version>
</dependency>
```

  NOTE

  For details about how to configure the Huawei Maven image source, visit **Huawei Open Source Image Site**, select Huawei SDK, and click HuaweiCloud SDK.

- ODBC driver package

  For example, click **huaweicloud-dli-odbc-1.1.2** to download the ODBC driver package of version 1.1.2.

  NOTE

  The ODBC driver package is named **huaweicloud-dli-odbc-<version>.zip**. Currently, the ODBC driver package can be used only in Windows 2012 or later versions.

----End

1.3.3 Performing Authentication

Scenario

Before a user uses JDBC or ODBC to set up a DLI link, the user must be authenticated.

Procedure

Currently, the JDBC supports authentication using the Access Key/Secret Key (AK/SK) or token. The ODBC supports authentication using the AK/SK or public cloud account. You can log in to the ODBC as a public cloud account or IAM user in a similar way of logging in to the public cloud console.

- (Recommended) Generating the AK/SK
  a. Log in to the DLI management console.
  b. Click the username in the upper right corner and select **My Credentials** from the drop-down list.
  c. On the displayed **My Credentials** page, click **Access Keys**. By default, the **Project List** page is displayed.
  d. Click **Add Access Key**. In the displayed **Add Access Key** dialog box, specify **Login Password** and **SMS Verification Code**.
  e. Click **OK** to download the certificate.
  f. After the certificate is downloaded, you can obtain the AK and SK information in the **credentials** file.
- Obtain the token

  When using token authentication, you need to obtain the user token and configure the token information in the JDBC connection parameters. You can obtain the token as follows:

  a. Send ***POST https://<IAM_Endpoint>/v3/auth/tokens***. To obtain the IAM endpoint and region name in the message body, see **Regions and Endpoints**.

     An example request message is as follows:

     NOTE

     The italic items in the following example need to be replaced with the actual content. For details, see the *Identity and Access Management API Reference*.

```
{
  "auth": {
    "identity": {
      "methods": [
        "password"
      ],
      "password": {
        "user": {
          "name": "username",
          "password": "password",
          "domain": {
            "name": "domainname"
          }
        }
      }
    },
    "scope": {
      "project": {
        "id": "0aa253a31a2f4cfda30eaa073fee6477" //Assume that project_id is 0aa253a31a2f4cfda30eaa073fee6477.
      }
    }
  }
}
```

  b. After the request is processed, the value of **X-Subject-Token** in the response header is the token value.

1.3.4 Submitting a Job Using JDBC

Scenario

In Linux or Windows, you can connect to the DLI server using JDBC.

NOTE

The job submitted by using JDBC to connect to DLI runs on the Spark engine.

DLI supports 13 data types. Each type can be mapped to a JDBC type. If JDBC is used to connect to the server, you must use the mapped Java type. **Table 1-5** describes the mapping relationships.

Table 1-5 Data type mapping

DLI Data Type	JDBC Type	Java Type
INT	INTEGER	java.lang.Integer
STRING	VARCHAR	java.lang.String
FLOAT	FLOAT	java.lang.Float
DOUBLE	DOUBLE	java.lang.Double
DECIMAL	DECIMAL	java.math.BigDecimal
BOOLEAN	BOOLEAN	java.lang.Boolean
SMALLINT/SHORT	SMALLINT	java.lang.Short
TINYINT	TINYINT	java.lang.Short
BIGINT/LONG	BIGINT	java.lang.Long
TIMESTAMP	TIMESTAMP	java.sql.Timestamp
CHAR	CHAR	Java.lang.Character
VARCHAR	VARCHAR	java.lang.String
DATE	DATE	java.sql.Date

Prerequisites

Before using JDBC, perform the following operations:

1. Getting authorized.

   DLI uses the Identity and Access Management (IAM) to implement fine-grained permissions for your enterprise-level tenants. IAM provides identity authentication, permissions management, and access control, helping you securely access to your public cloud resources.

   With IAM, you can use your public cloud account to create IAM users for your employees, and assign permissions to the users to control their access to specific resource types.

Currently, roles (coarse-grained authorization) and policies (fine-grained authorization) are supported. For details about permissions and authorization operations, see the **Data Lake Insight User Guide**.

2. Create a queue. Set **Queue Type** to **SQL Queue**, which is the computing resource of the SQL job.

   You can create a queue on the **Overview**, **SQL Editor**, or **Queue Management** page.

   - In the upper right corner of the Dashboard page, click 

     to create a queue.
   - To create a queue on the **Queue Management** page:
     i. In the navigation pane of the DLI management console, choose **Queue Management**.
     ii. In the upper right corner of the **Queue Management** page, click 

     to create a queue.
   - To create a queue on the **SQL Editor** page:
     i. On the top menu bar of the DLI management console, click **SQL Editor**.
     ii. On the left pane of the displayed **Job Editor** page, click  . Click  to the right of **Queues**.

   NOTE

   If the user who creates the queue is not an administrator, the queue can be used only after being authorized by the administrator. For details about how to assign permissions, see the **Data Lake Insight User Guide**.

Procedure

Step 1 Install JDK 1.7 or later on the computer where JDBC is installed, and configure environment variables.

Step 2 Obtain the DLI JDBC driver package **huaweicloud-dli-jdbc-<version>.zip** by referring to **Downloading the JDBC or ODBC Driver Package**. Decompress the package to obtain **huaweicloud-dli-jdbc-<version>-jar-with-dependencies.jar**.

Step 3 On the computer using JDBC, add **huaweicloud-dli-jdbc-1.1.1-jar-with-dependencies.jar** to the **classpath** path of the Java project.

Step 4 DLI JDBC provides two authentication modes, namely, token and AK/SK, to connect to DLI. For details about how to obtain the token and AK/SK, see **Performing Authentication**.

Step 5 Run the **Class.forName()** command to load the DLI JDBC driver.

```
Class.forName("com.huawei.dli.jdbc.DliDriver");
```

Step 6 Call the GetConnection method of DriverManager to create a connection.

```
Connection conn = DriverManager.getConnection(String url, Properties info);
```


JDBC configuration items are passed using the URL. For details, see **Table 1-6**. JDBC configuration items can be separated by semicolons (;) in the URL, or you can dynamically set the attribute items using the Info object. For details, see **Table 1-7**.

Table 1-6 Database connection parameters

Parameter	Description
url	The URL format is as follows: jdbc:dli://<endPoint>/projectId? <key1>=<val1>;<key2>=<val2>... • **EndPoint** indicates the DLI domain name. **ProjectId** indicates the project ID. To obtain the endpoint corresponding to DLI, see **Regions and Endpoints**. To obtain the project ID, log in to the public cloud, move the mouse on the account, and click **My Credentials** from the shortcut menu. • Other configuration items are listed after **?** in the form of **key=value**. The configuration items are separated by semicolons (;). They can also be passed using the Info object.
Info	The Info object passes user-defined configuration items. If Info does not pass any attribute item, you can set it to null. The format is as follows: info.setProperty ("Attribute item", "Attribute value").

Table 1-7 Attribute items

Item	Mandatory	Default Value	Description
queuename	Yes	-	Queue name of DLI.
databasename	No	-	Name of a database.
authenticationmode	No	token	Authentication mode. Currently, token- and AK/SK-based authentication modes are supported.
accesskey	Yes	-	AK/SK authentication key. For details about how to obtain the key, see **Performing Authentication**.
secretkey	Yes	-	AK/SK authentication key. For details about how to obtain the key, see **Performing Authentication**.

Item	Mandatory	Default Value	Description
regionname	This parameter must be configured if **authenticationmode** is set to **aksk**.	-	Region name. For details, see **Regions and Endpoints**.
servicename	This parameter must be configured if **authenticationmode** is set to **aksk**.	-	Indicates the service name, that is, **dli**.
token	This parameter must be configured if **authenticationmode** is set to **token**.	-	Token authentication. For details about the authentication mode, see **Performing Authentication**.
charset	No	UTF-8	JDBC encoding mode.
usehttpproxy	No	false	Whether to use the access proxy.
proxyhost	This parameter must be configured if **usehttpproxy** is set to **true**.	-	Access proxy host.
proxyport	This parameter must be configured if **usehttpproxy** is set to **true**.	-	Access proxy port.

Item	Mandatory	Default Value	Description
dli.sql.checkNoResultQuery	No	false	Whether to allow invoking the executeQuery API to execute statements (for example, DDL) that do not return results. • Value **false** indicates that invoking of the executeQuery API is allowed. • Value **true** indicates that invoking of the executeQuery API is not allowed.
jobtimeout	No	300	End time of the job submission. Unit: second
iam.endpoint	No. By default, the value is automatically combined based on **regionName**.	-	Endpoint. For details about regions, see **Regions and Endpoints**.
obs.endpoint	No. By default, the value is automatically combined based on **regionName**.	-	Endpoint. For details about regions, see **Regions and Endpoints**.
directfetchthreshold	No	1000	If the query result is less than or equal to the preset value, the getJobResult function is called to obtain the result. Otherwise, the exported result is the returned query result.

Step 7 Create a Statement object, set related parameters, and submit Spark SQL to DLI.

Statement statement = conn.createStatement();

statement.execute("select * from tb1");

Step 8 Obtain the result.

ResultSet rs = statement.getResultSet();

Step 9 Display the result.

```
while (rs.next()) {
int a = rs.getInt(1);
```



```
int b = rs.getInt(2);
}
```

Step 10 Close the connection.

conn.close();

----End

Example

```
import java.sql.*;
import java.util.Properties;

public class DLIJdbcDriverExample {

    public static void main(String[] args) throws ClassNotFoundException, SQLException {
        Connection conn = null;
        try {
            Class.forName("com.huawei.dli.jdbc.DliDriver");
            String url = "jdbc:dli://<endpoint>/<projectId>?databasename=db1;queuename=testqueue";
            Properties info = new Properties();
            info.setProperty("authenticationmode", "aksk");
            info.setProperty("regionname", "<real region name>");
            info.setProperty("accesskey", "<real ak>");
            info.setProperty("secretkey", "<real sk>")
            conn = DriverManager.getConnection(url, info);
            Statement statement = conn.createStatement();
            statement.execute("select * from tb1");
            ResultSet rs = statement.getResultSet();
            int line = 0;
            while (rs.next()) {
                line ++;
                int a = rs.getInt(1);
                int b = rs.getInt(2);
                System.out.println("Line:" + line + ":" + a + "," + b);
            }
            statement.execute("describe tb1");
            ResultSet rs1 = statement.getResultSet();
            line = 0;
            while (rs1.next()) {
                line ++;
                String a = rs1.getString(1);
                String b = rs1.getString(2);
                System.out.println("Line:" + line + ":" + a + "," + b);
            }
        } catch (SQLException ex) {
        } finally {
            if (conn != null) {
                conn.close();
            }
        }
    }
}
```

Enabling JDBC Requery

If the JDBC requery function is enabled, the system automatically requeries when the query operation fails.

NOTE

- To avoid repeated data insertion, non-query statements do not support requery.
- This function is available in the JDBC driver package of 1.1.5 or later. To use this function, obtain the latest JDBC driver package.

To enable the requery function, add the attributes listed in **Table 1-8** to the **Info** parameter.

Table 1-8 Requery parameter description

Item	Mandatory	Default Value	Description
USE_RETRY_KEY	Yes	false	Whether to enable the requery function. If this parameter is set to **True**, the requery function is enabled.
RETRY_TIMES_KEY	Yes	3000	Requery times. You are advised to set this parameter to 3 to 5 times.
RETRY_INTERVALS_KEY	Yes	3	Requery interval (milliseconds). You are advised to set this parameter to **30000 ms**.

Set JDBC parameters, enable the requery function, and create a link. The following is an example:

```
import java.sql.*;
import java.util.Properties;

public class DLIJdbcDriverExample {

    private static final String X_AUTH_TOKEN_VALUE = "<realtoken>";
    public static void main(String[] args) throws ClassNotFoundException, SQLException {
        Connection conn = null;
        try {
            Class.forName("com.huawei.dli.jdbc.DliDriver");
            String url = "jdbc:dli://<endpoint>/<projectId>?databasename=db1;queuename=testqueue";
            Properties info = new Properties();
            info.setProperty("token", X_AUTH_TOKEN_VALUE);
info.setProperty(ConnectionResource.USE_RETRY_KEY, "true"); // Enable the requery function.
info.setProperty(ConnectionResource.RETRY_TIMES_KEY, "30000");// Requery interval (ms)
info.setProperty(ConnectionResource.RETRY_INTERVALS_KEY, "5");// Requery Times
            conn = DriverManager.getConnection(url, info);
            Statement statement = conn.createStatement();
            statement.execute("select * from tb1");
            ResultSet rs = statement.getResultSet();
            int line = 0;
            while (rs.next()) {
                line ++;
                int a = rs.getInt(1);
                int b = rs.getInt(2);
                System.out.println("Line:" + line + ":" + a + "," + b);
            }
            statement.execute("describe tb1");
            ResultSet rs1 = statement.getResultSet();
            line = 0;
            while (rs1.next()) {
                line ++;
                String a = rs1.getString(1);
                String b = rs1.getString(2);
                System.out.println("Line:" + line + ":" + a + "," + b);
            }
        } catch (SQLException ex) {
        } finally {
            if (conn != null) {
                conn.close();
```



```
        }
      }
    }
}
```

1.3.5 Submitting a Job Using ODBC

Scenario

In Windows, you can use an ODBC application to connect to the DLI server and submit jobs.

Precautions

OS requirement: Windows 2012 or later

Prerequisites

Before using ODBC, perform the following operations:

1. Getting authorized.

   DLI uses the Identity and Access Management (IAM) to implement fine-grained permissions for your enterprise-level tenants. IAM provides identity authentication, permissions management, and access control, helping you securely access to your public cloud resources.

   With IAM, you can use your public cloud account to create IAM users for your employees, and assign permissions to the users to control their access to specific resource types.

   Currently, roles (coarse-grained authorization) and policies (fine-grained authorization) are supported. For details about permissions and authorization operations, see the **Data Lake Insight User Guide**.

2. Create a queue. Set **Queue Type** to **SQL Queue**, which is the computing resource of the SQL job.

   You can create a queue on the **Overview**, **SQL Editor**, or **Queue Management** page.

   - In the upper right corner of the Dashboard page, click 

     to create a queue.
   - To create a queue on the **Queue Management** page:
     i. In the navigation pane of the DLI management console, choose **Queue Management**.
     ii. In the upper right corner of the **Queue Management** page, click 

     to create a queue.
   - To create a queue on the **SQL Editor** page:
     i. On the top menu bar of the DLI management console, click **SQL Editor**.
     ii. On the left pane of the displayed **Job Editor** page, click  . Click  to the right of **Queues**.

NOTE

If the user who creates the queue is not an administrator, the queue can be used only after being authorized by the administrator. For details about how to assign permissions, see the **Data Lake Insight User Guide**.

Procedure

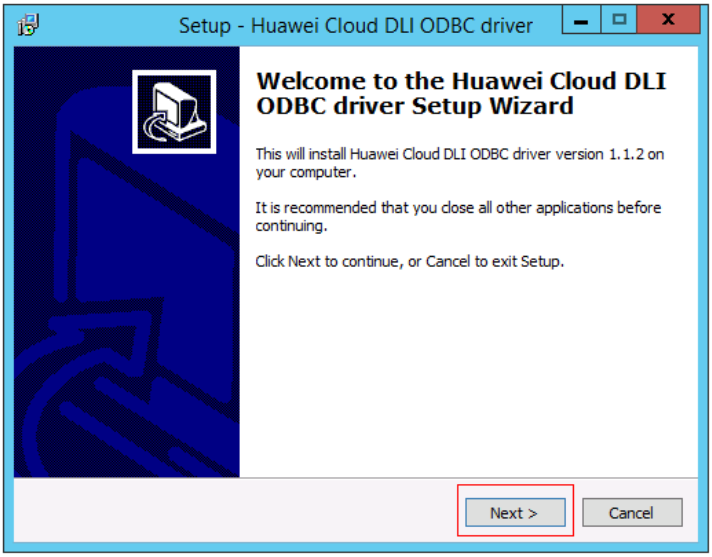
Step 1 Decompress the installation package.

Decompress **huaweicloud-dli-odbc-<version>.zip** to any directory of the system.

Step 2 Install the ODBC driver.

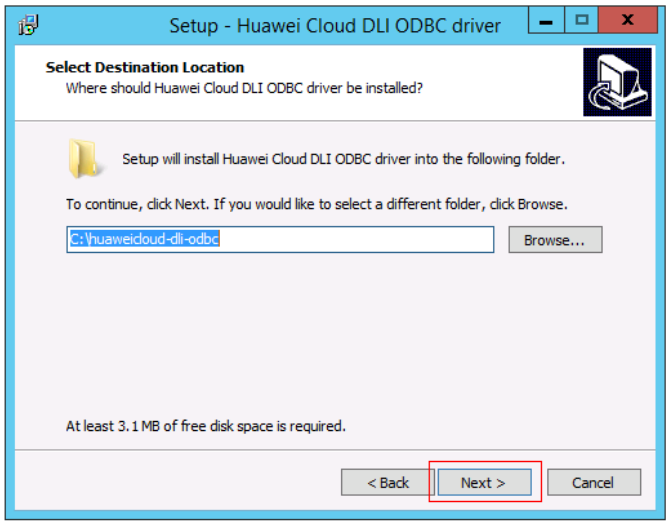
1. Double-click the **huaweicloud-dli-odbc-<version>-setup.exe** file in the decompression directory.
2. Click **Next** to install the ODBC driver.

Figure 1-1 ODBC driver installation wizard



a. Specify the installation path, which is **C:\huaweicloud-dli-odbc** by default. Then, click **Next**.

Figure 1-2 Selecting the installation path

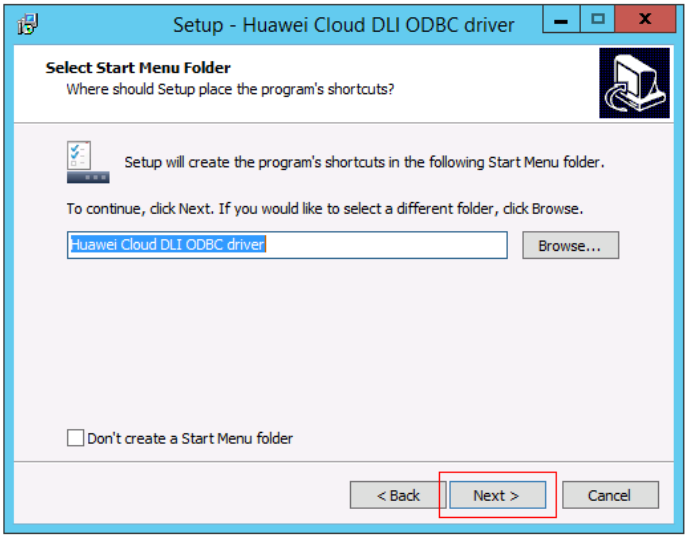


NOTE

You are advised to use the default installation path. If you change the installation path to another one, you need to manually modify the log configuration file.

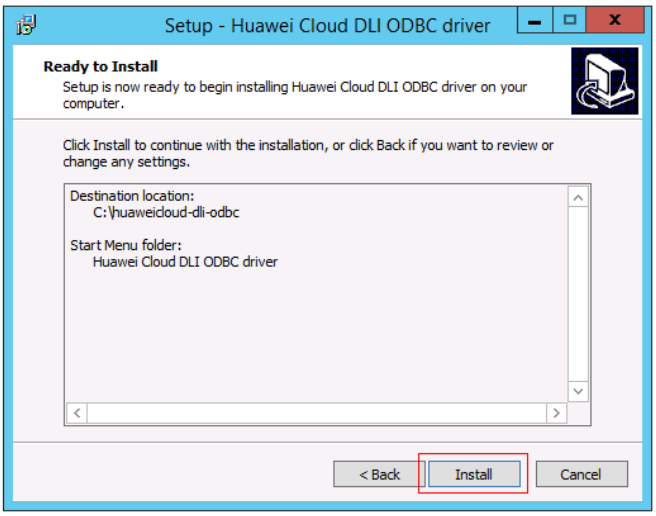
b. Specify the start menu folder and click **Next**.

Figure 1-3 Selecting the start menu folder



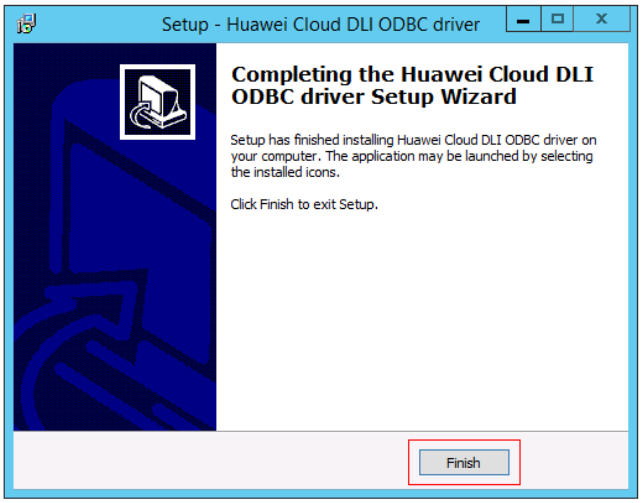
c. After confirming the installation path and start menu folder, click **Install**.

Figure 1-4 Confirming the installation path and start menu folder



d. Click **Finish**.

Figure 1-5 Installation completion



Step 3 (Optional) Configure log parameters.

NOTE

This step is required if the ODBC driver installation directory is not **C:\huaweicloud-dli-odbc**.

1. Open the **log.properties** file in the driver installation directory, for example: **C:\xxx\windows\log.properties**.
2. Change the log output path. For example: **log4cplus.appender.DLILog.File=C:\xxx\windows\log\dli_odbc.log**.

NOTE

Set the log output path based on the actual installation path. The **log** directory in the installation path is recommended.

Step 4 Configure a data source.

1. Open **Control Panel**. Click **Administrative Tools**. Double-click **ODBC Data Sources (32-bit)** to start the 32-bit ODBC data source manager.

   NOTE

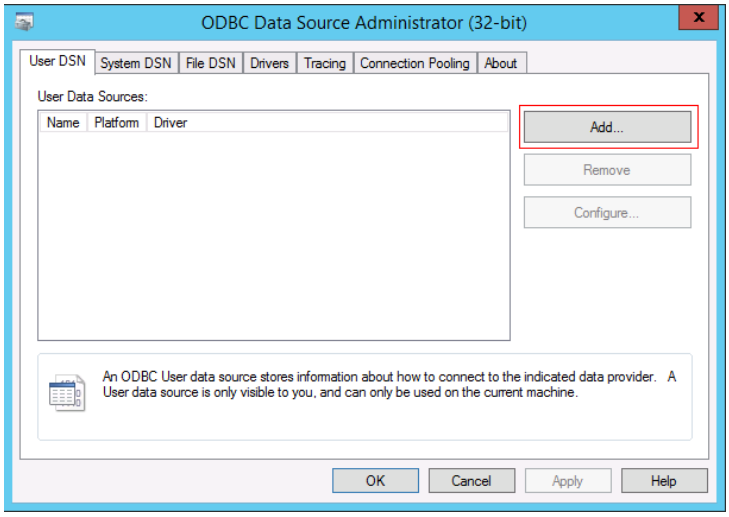
   You can start the 32-bit ODBC data source manager using the following programs:

   - For a 64-bit Windows OS, use **C:\Windows\SysWOW64\odbcad32.exe**.
   - For a 32-bit Windows OS, use **C:\Windows\System32\odbcad32.exe**.

2. Add a data source.

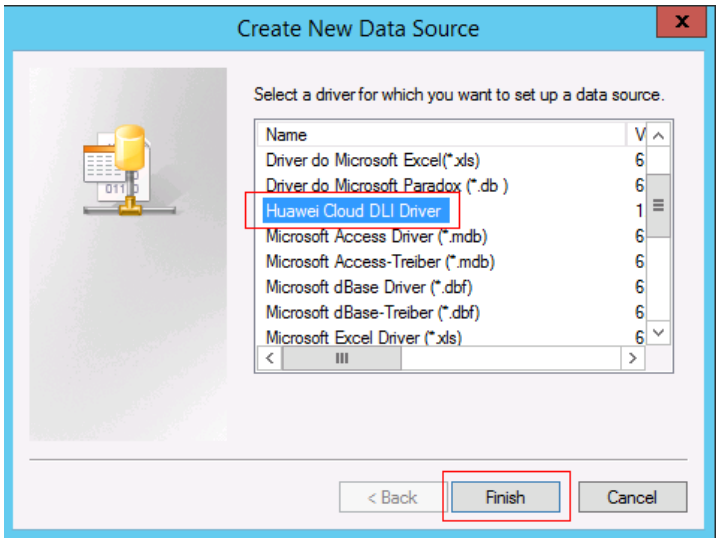
   To add a user data source, click **Add** on the **User DSN** page. To add a system data source, click **Add** on the **System DSN** page.

   Figure 1-6 ODBC data source management



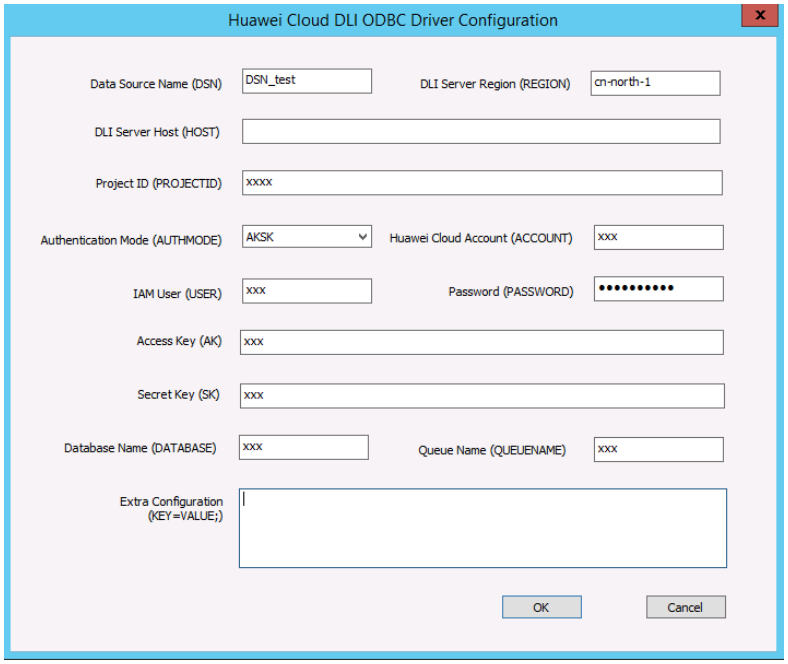
3. Select **Huawei Cloud DLI Driver** and click **Finish**.

   Figure 1-7 Creating a data source



4. Set the ODBC driver parameters and click **OK**. For details about the ODBC driver parameters, see **Table 1-9**.

Figure 1-8 DLI ODBC driver configuration



NOTE

- All parameters, except for **DSN**, can be configured through the ODBC connection parameters in the application. If a parameter is configured during both the ODBC connection parameter configuration and the data source configuration, the ODBC driver preferentially uses the configuration for this parameter in the connection parameter configuration.
- If there are multiple configurations in the **Extra Configuration** area, use the format of KEY1=VALUE1; KEY2=VALUE2.

Table 1-9 DLI ODBC driver configuration parameters

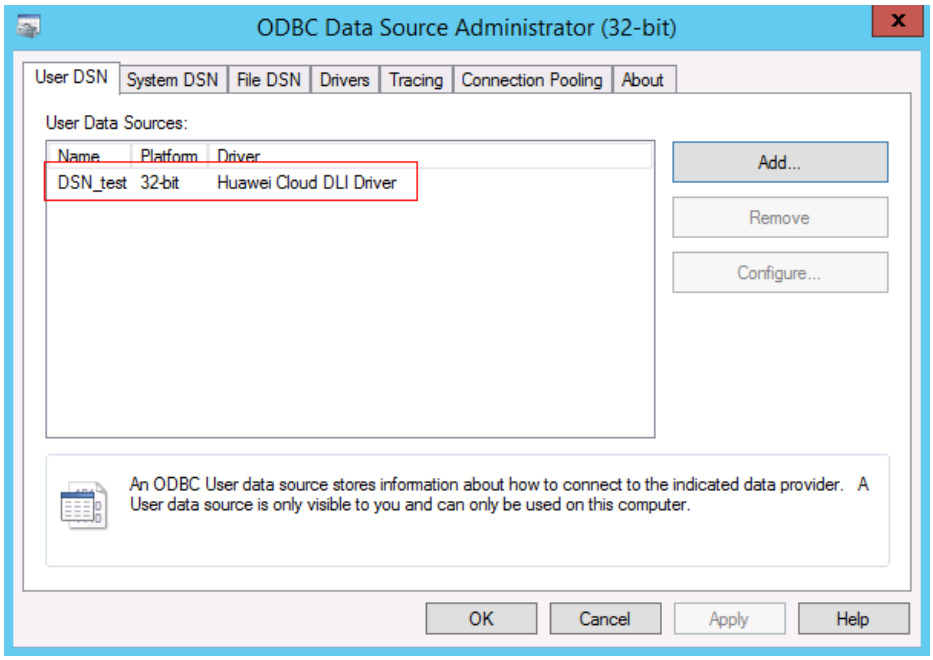
Item	Mandatory	Default Value	Description
DSN	Yes	-	Name of the ODBC data source, which is used in the connection string of the application.
REGION	Yes	-	Region of interconnected DLI, for example: **cn-north-1**.
HOST	No	-	Domain name of interconnected DLI.
PROJECTID	Yes	-	Project ID used by a user to access DLI.

Item	Mandatory	Default Value	Description
AUTHMODE	Yes	-	Authentication mode used to connect to DLI. Currently, token authentication and AK/SK authentication are supported. For details, see **Performing Authentication**. Value **0** indicates that token authentication is used. Value **1** indicates that AK/SK authentication is used.
ACCOUNT	No	-	Account name for logging in to the public cloud. This parameter is mandatory when **AUTH MODE** is set to **TOKEN**.
USER	No	-	IAM usb-user name for logging in to the public cloud. This parameter is mandatory when **AUTH MODE** is set to **TOKEN**. If a tenant account is used for logging in to the public cloud, the value of this parameter is the same as that of **ACCOUNT**.
PASSWORD	No	-	User password for logging in to the public cloud. This parameter is mandatory when **AUTH MODE** is set to **TOKEN**.
AK	No	-	Access key of a user. This parameter is mandatory when **AUTH MODE** is set to **AK/SK**.
SK	No	-	Secret key of a user. This parameter is mandatory when **AUTH MODE** is set to **AK/SK**.
DATABASE	No	default	Name of the database used by a user.
QUEUENAME	No	default	Name of the queue used by a user.
USEPROXY	No	0	Whether to use an agency to access DLI. Value **0** indicates that no agency is used. Value **1** indicates that an agency is used. This parameter is configured in **Extra Configuration**.
PROXYHOST	No	-	Proxy server address. This parameter is configured in **Extra Configuration**.
PROXYPORT	No	-	Proxy server port. This parameter is configured in **Extra Configuration**.

Item	Mandatory	Default Value	Description
JOBTIMEOUT	No	300	Timeout interval for submitting a DLI job, in the unit of seconds. This parameter is configured in **Extra Configuration**.

5. After the configuration is completed, view the configured data source in the driver manager.

Figure 1-9 Viewing the configured data source



----End

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.Odbc;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString =
"DSN=DLIODBCSysDS;AUTHMODE=0;ACCOUNT=xxx;USER=xxx;PASSWORD=xxx";

            string queryString = "show tables;";
            OdbcCommand command = new OdbcCommand(queryString);

            using (OdbcConnection connection = new OdbcConnection(connectionString))
```



```
        {

            OdbcDataReader reader;
            try
            {
                command.Connection = connection;

                connection.Open();

                reader = command.ExecuteReader();

                while (reader.Read())
                {
                    Console.WriteLine("\t{0}\t{1}\t{2}",
reader[0], reader[1], reader[2]);
                }

                reader.Dispose();
                reader.Close();

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        command.Dispose();
    }
  }
}
```

NOTE

1. In this example, C# sample code is used.
2. In actual conditions, replace the DSN specified by **string connectionString** in the example code with the data source name configured in **Step 4**.

1.3.6 Introduction to RDS

Basic Concepts

Relational Database Service (RDS) is a cloud-based web service that is reliable, scalable, easy to manage, and immediately ready for use. It can be deployed in single-node, active/standby, or cluster mode.

Features

- Hosting: RDS provides upgrade, patch, and backup hosting services.
- Compatibility: RDS is compatible with MySQL, SQL Server, and PostgreSQL.
- Scalability: RDS provides elastic capacity expansion to deliver one-click CPU, memory, and disk capacity expansion
- Security: RDS supports creation of VPCs, subnets, and security groups.
- Stability: RDS supports automatic or manual backup.
- High availability: RDS can work in active/standby mode.

2 Using UDFs

Scenario

DLI allows you to query data by using the user-defined functions (Hive UDF).

NOTE

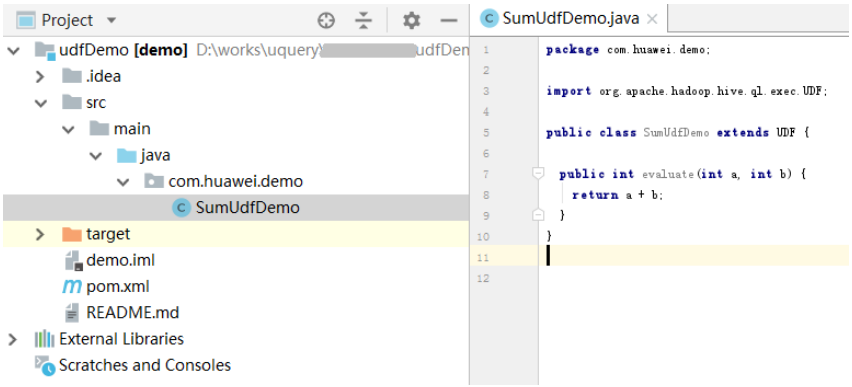
- When performing UDF-related operations on the DLI console, you need to use a self-created queue.
- When a UDF is used across accounts, it can be used only after being authorized. Only the user who creates the UDF does not need authorization. On the DLI management console, you can choose **Data Management > Package Management**, select the corresponding UDF JAR package, and click **Manage Permissions** in the **Operation** column. On the displayed page, click **Grant Permission** in the upper right corner and select the corresponding permission.

Procedure

1. Compile a UDF.

   Create or modify the content of **SumUdfDemo.java** in the sample code based on service requirements. For details, see **Example Code**.

   Figure 2-1 Creating or modifying **SumUdfDemo.java**



2. Generate a JAR package, set the output JAR package to **TestUDF.jar**, and run the **Build Artifacts** command.

Figure 2-2 Selecting **Artifacts**

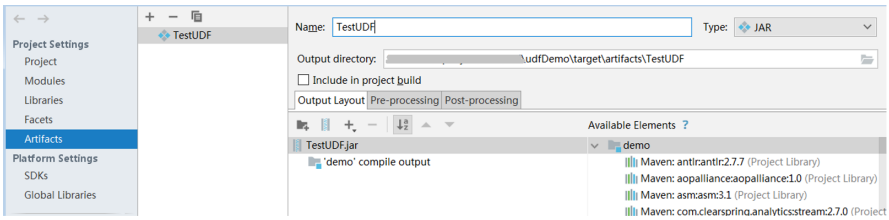
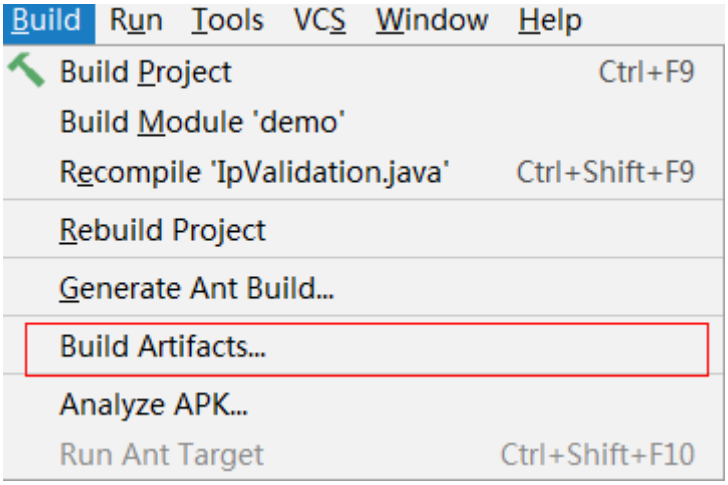


Figure 2-3 Running **Build Artifacts**



Once the artifacts are successfully built, a **TestUDF.jar** file is generated in the corresponding path. As shown in **Figure 2-2**, the path is **udfDemo\target\artifacts\TestUDF**.

3. Upload **TestUDF.jar** to OBS. For details about how to upload data to OBS, see *Step 2: Upload Data to OBS* in **Submitting a SQL Job**.
4. Create a function.

   Run the following command on the management console to create a function:

   ```
   CREATE FUNCTION fun1 AS 'com.huawei.demo.SumUdfDemo' using jar 'obs://udf/TestUDF.jar';
   ```

   NOTE

   - If you use an existing class name to create a function, you must restart the original queue on the **Queue Management** page. Otherwise, the function may not take effect.
   - For details about the SQL syntax of user-defined functions, see the *Data Lake Insight SQL Syntax Reference*.

5. Use the created function.

   Run the following statement to query using the function created in **4**.

   ```
   select fun1(ip) from ip_tables;
   ```

6. Delete the created function.

   If this function is no longer used, run the following statement to delete the function:

   ```
   Drop FUNCTION fun1;
   ```


Example Code

The sample code in **SumUdfDemo.java** is as follows:

```
package com.huawei.demo;
  import org.apache.hadoop.hive.ql.exec.UDF;
  public class SumUdfDemo extends UDF {
    public int evaluate(int a, int b) {
     return a + b;
  }
 }
```


3 Connecting the Third-party BI Tool to the DLI

3.1 Interconnection Between Yonghong BI System and DLI

3.1.1 Preparing for Yonghong BI Interconnection

Scenario

Prepare for the interconnection between Yonghong BI system and DLI.

Procedure

Step 1 (Optional) From the top navigation menu of the HUAWEI CLOUD management console, choose **Service List > Enterprise Intelligence > Data Lake Insight**. On the displayed page, click **Common Links** and download the DLI JDBC driver, for example, **dli-jdbc-1.1.0-jar-with-dependencies-jdk1.7.jar**. For details, see **Downloading the JDBC or ODBC Driver Package**.

NOTE

You can skip this step because the DLI JDBC driver has been uploaded to the Yonghong SaaS production environment.

Step 2 The AK/SK and token authentication modes can be used for JDBC authentication. The AK/SK mode is recommended. For details, see **Performing Authentication**.

Step 3 Contact Yonghong customer service personnel to obtain the username and password of the Yonghong SaaS production environment.

Step 4 Log in to **https://henter.yonghongcloud.com/enter/** and enter the username and password.

----End

3.1.2 Adding Yonghong BI Data Source

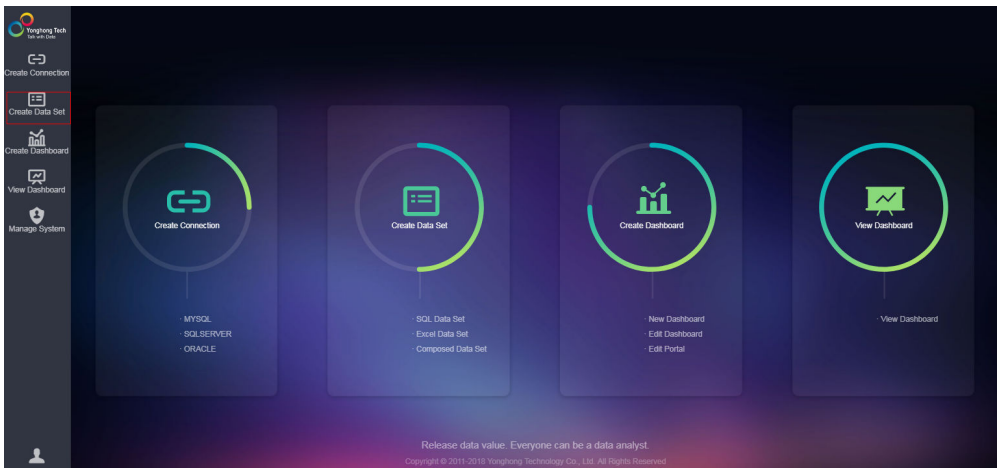
Scenario

Add the DLI data source to the Yonghong SaaS production environment.

Procedure

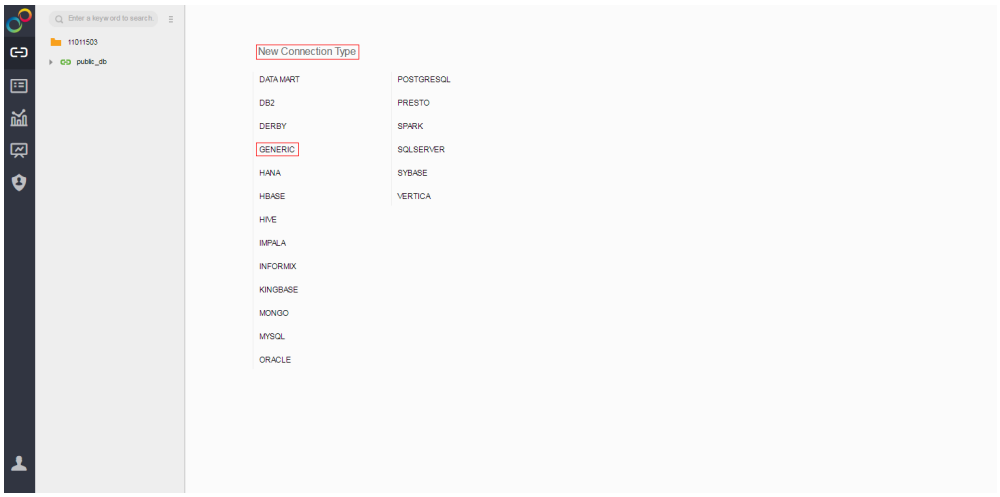
Step 1 On the homepage of the Yonghong SaaS production environment, click **Create Connection** from the left navigation tree. See **Figure 3-1**.

Figure 3-1 Adding a connection



Step 2 On the **New Connection Type** page, choose **GENERIC** for the type of the new connection. See **Figure 3-2**.

Figure 3-2 Choosing a new connection type



Step 3 Configure the new connection. See **Figure 3-3**.

In **Driver**, enter **com.huawei.dli.jdbc.DliDriver**.

In **URL**, select **Self-defined Protocol**. Enter the URL of the DLI JDBC driver. For details about the URL format and the attributes, see **Table 3-1** and **Table 3-2**, respectively.

NOTE

- In **Schema**, you can optionally enter the name of the database to be accessed. If you enter the name, only tables in the database are displayed during data set creation. If you do not enter the name, tables in all databases are displayed during data set creation. For details about how to create a data set, see **Creating Yonghong BI Data Set**.
- Retain default values of other parameters. You do not need to select **Request Login**.

Figure 3-3 Configuring the new connection

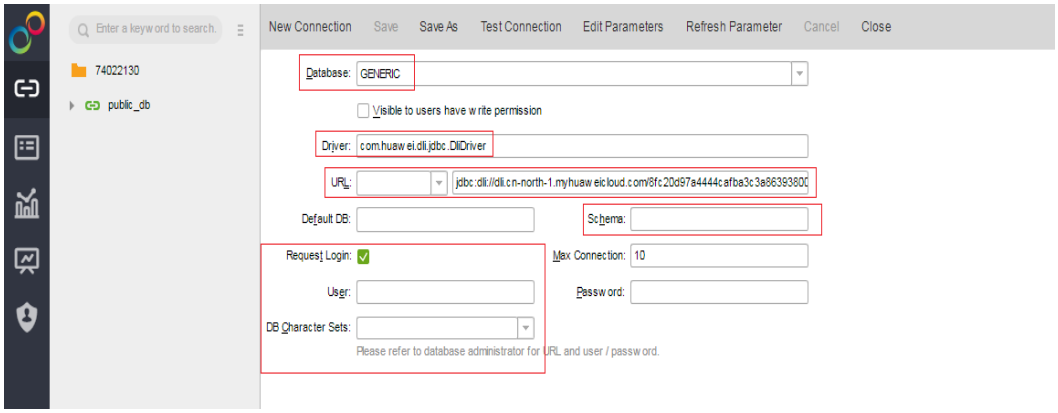


Table 3-1 Database connection parameters

Parameter	Description
URL	The URL format is as follows: ***jdbc:dli://<endPoint>/<projectId>?<key1>=<val1>;<key2>=<val2>...*** **NOTE** • **endpoint** indicates the domain name of DLI. For details, see **Regions and Endpoints**. • **projectId** indicates the project ID, which can be obtained from the **My Credentials** page of the public cloud platform. • The question mark (?) is followed by other configuration items. Each configuration item is listed in the "key=value" format. Semicolons (;) are used to separate configuration items. For details, see **Table 3-2**.

Table 3-2 Attribute-related configuration items

Attribute (key)	Mandatory	Default Value (value)	Description
queuename	Yes	-	Queue name of DLI.
databasename	No	-	Default database to be accessed. If this parameter is not specified in the URL, you need to use **db.table** (for example, **select * from dbother.tabletest**) to access tables in the database.
authentication mode	Yes	token	Authentication method, which can be **token** or **aksk**. Value **aksk** is recommended during the interconnection with Yonghong BI system.
accesskey	This parameter must be configured if **authentication mode** is set to **aksk**.	-	For details, see **Preparing for Yonghong BI Interconnection**.
secretkey	This parameter must be configured if **authentication mode** is set to **aksk**.	-	For details, see **Preparing for Yonghong BI Interconnection**.
regionname	This parameter must be configured if **authentication mode** is set to **aksk**.	-	For details, see **Regions and Endpoints**.
servicename	This parameter must be configured if **authentication mode** is set to **aksk**.	-	**servicename**=**dli**

Attribute (key)	Mandatory	Default Value (value)	Description
dli.sql.checkNoResultQuery	No	false	Whether to allow invoking the executeQuery API to execute statements (for example, DDL) that do not return results. • Value **false** indicates that invoking of the executeQuery API is allowed. • Value **true** indicates that invoking of the executeQuery API is not allowed. **NOTE** If **dli.sql.checkNoResultQuery** is set to **false**, non-query statements will be executed twice.

Step 4 On the tool bar of the displayed page, click **Test Connection**. After the test is complete, click **Save**. Enter the data source name, and save the data source.

NOTE

Currently, you are not allowed to save the data source to the root directory. Therefore, you can only save the data source to an existing folder.

----End

3.1.3 Creating Yonghong BI Data Set

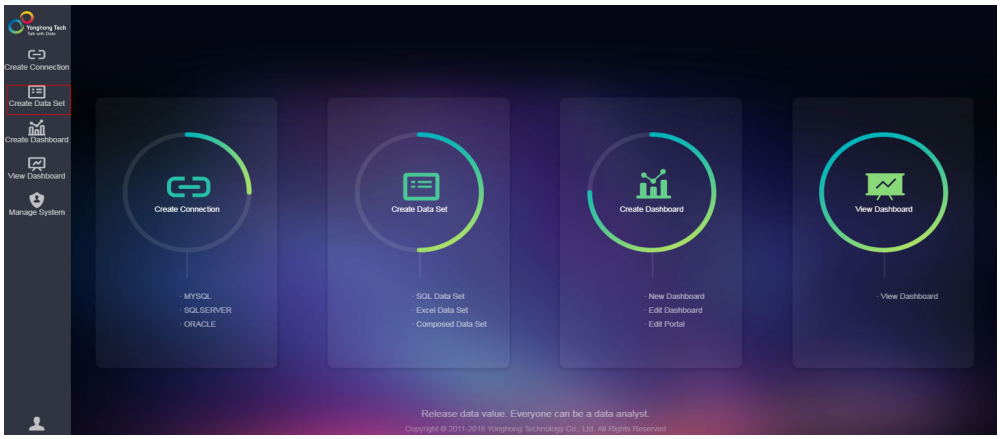
Scenario

Create a DLI data set in the Yonghong SaaS production environment.

Procedure

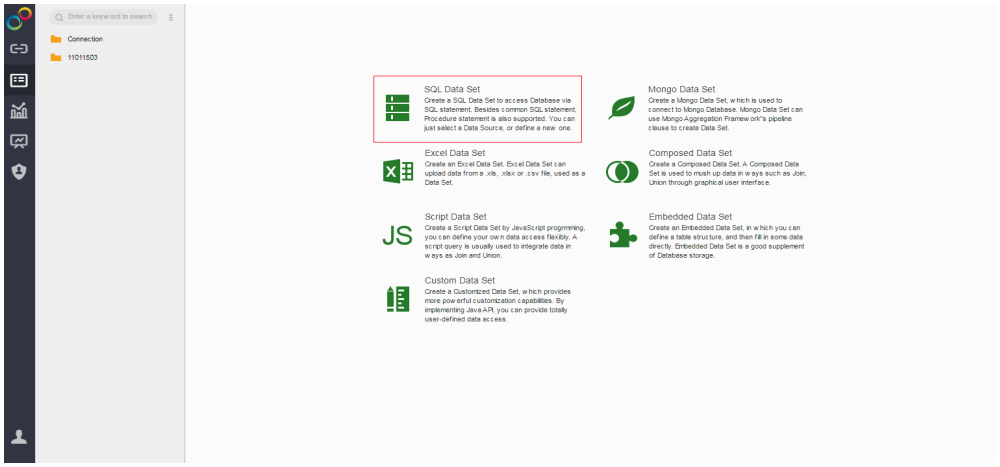
Step 1 On the home page of the Yonghong SaaS production environment, click **Create Data Set** in the left navigation tree. See **Figure 3-4**.

Figure 3-4 Creating a data set



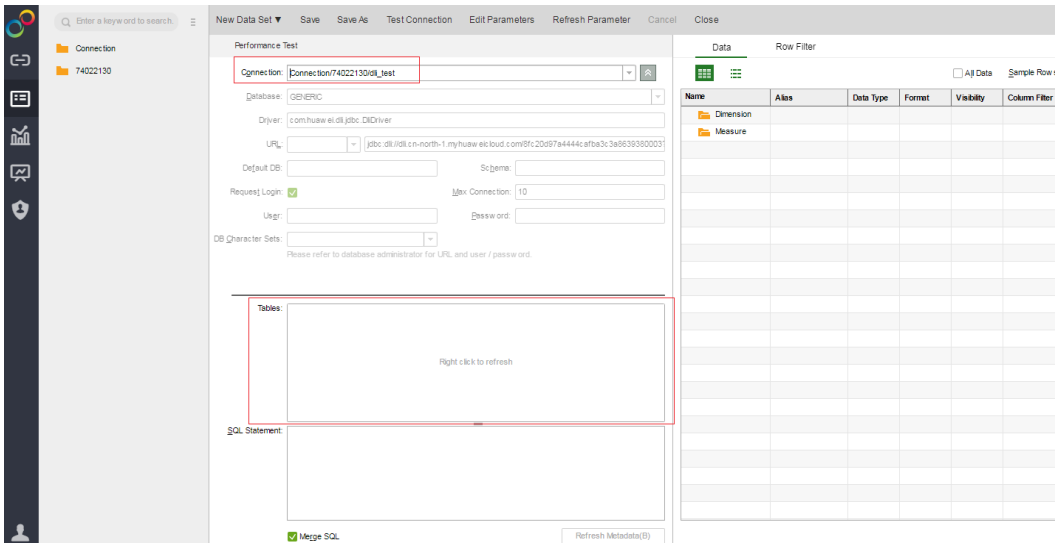
Step 2 On the displayed page, click **SQL Data Set**. See **Figure 3-5**.

Figure 3-5 Creating a SQL data set



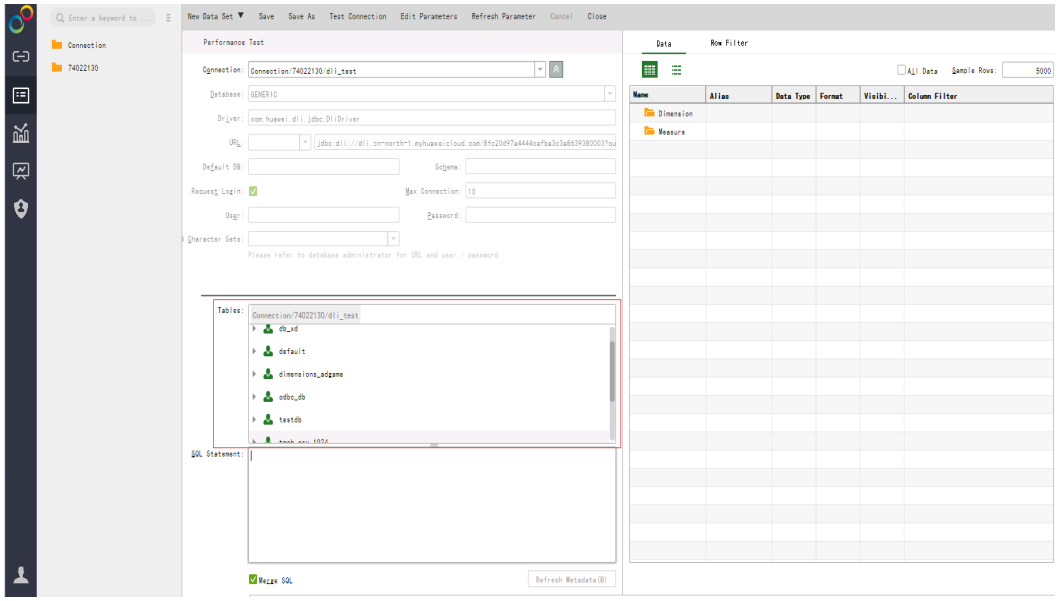
Step 3 On the displayed page, select the added DLI data source from the **Connection** drop-down list box. See **Figure 3-6**.

Figure 3-6 Selecting a data source



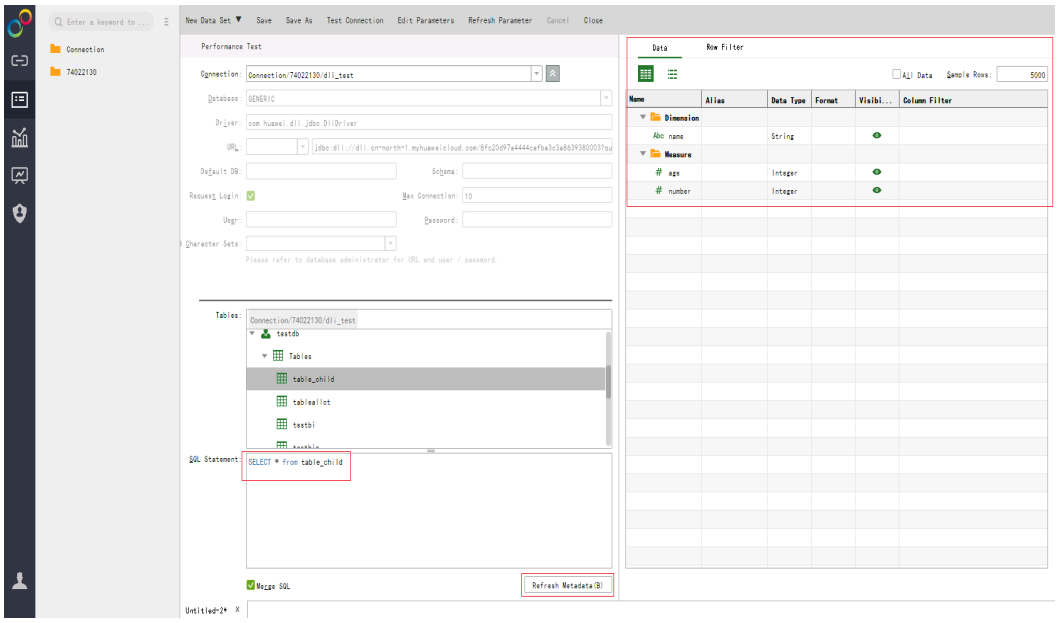
Step 4 In the **Table** area on the left pane, right-click and choose **Update** to update tables. All databases and their tables are listed in the area. **Figure 3-7** shows the page displayed when **Table Structure** is not configured during connection creation.

Figure 3-7 Updating tables



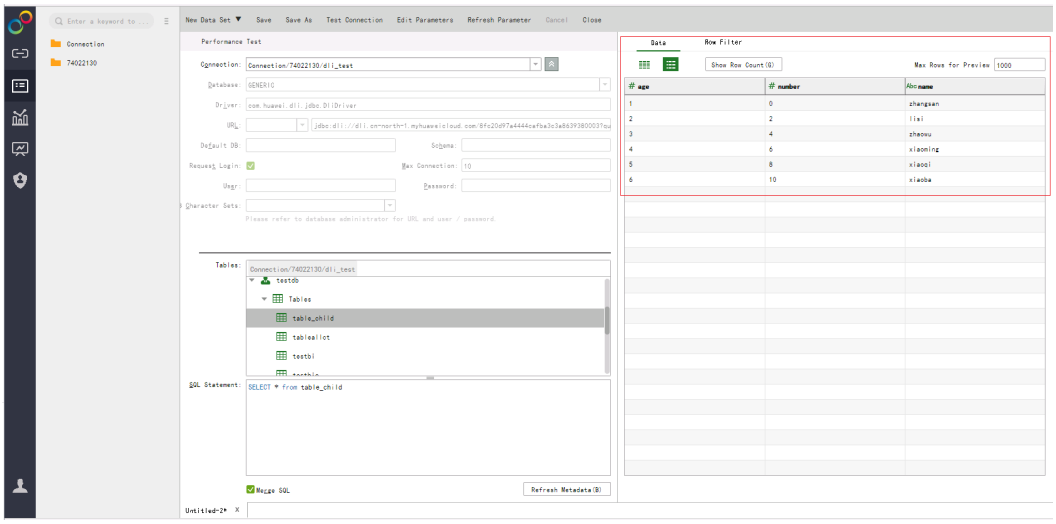
Step 5 In the **SQL Statement** area on the left pane, enter the **select * from table_name** command to query tables. On the **Preview Data** page on the right pane, click  . Metadata of the table, including fields and field types, is displayed. See **Figure 3-8**.

Figure 3-8 Querying the table



Step 6 Click  on the right pane to query data details. See **Figure 3-9**.

Figure 3-9 Querying data of the table



Step 7 On the tool bar of the displayed page, click **Save**.

----End

3.1.4 Creating a Chart in Yonghong BI

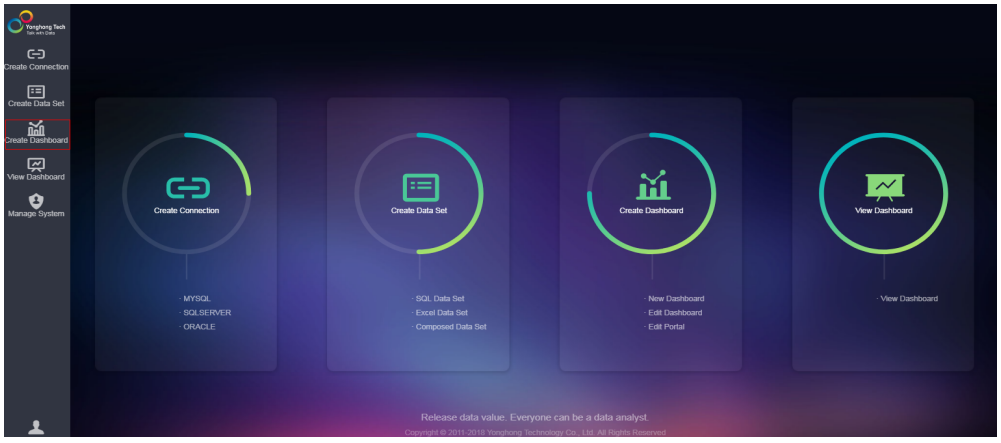
Scenario

Create a chart in the Yonghong SaaS production environment.

Procedure

Step 1 On the home page of the Yonghong SaaS production environment, click **Create Dashboard** in the left navigation tree. See **Figure 3-10**.

Figure 3-10 Creating a dashboard

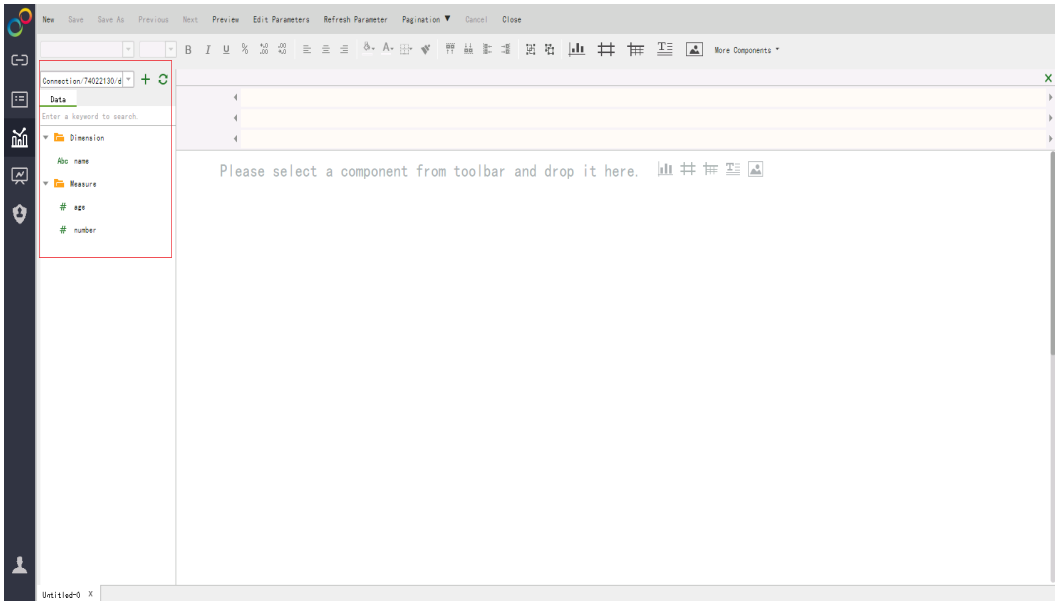


Step 2 Select a theme. See **Figure 3-11**.

Figure 3-11 Selecting a theme

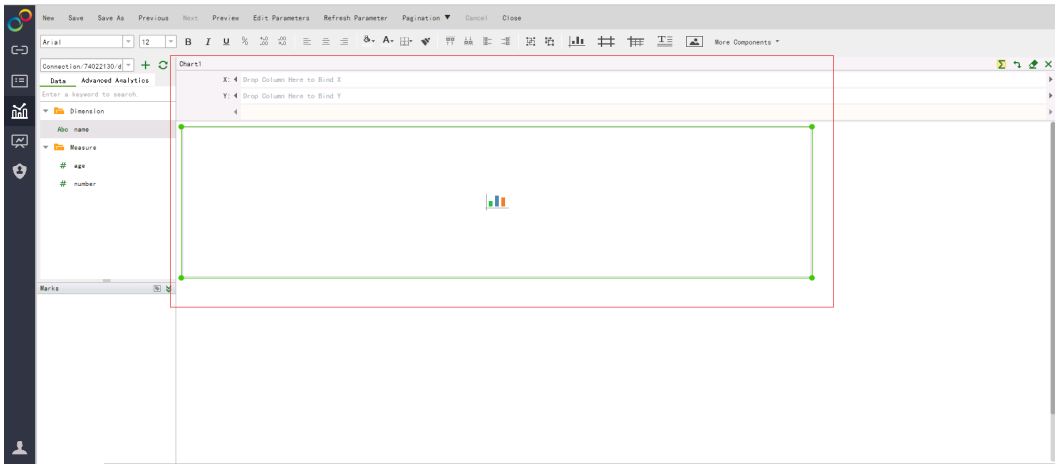
Step 3 In this example, the Refreshing Green theme is selected. On the left pane, select the created data set from the drop-down list box and choose a table as the data source (for example, **table_child**). Metadata (including fields and field types) of the table is displayed in the lower part of the **Data** column. See **Figure 3-12**.

Figure 3-12 Selecting a connection for the table



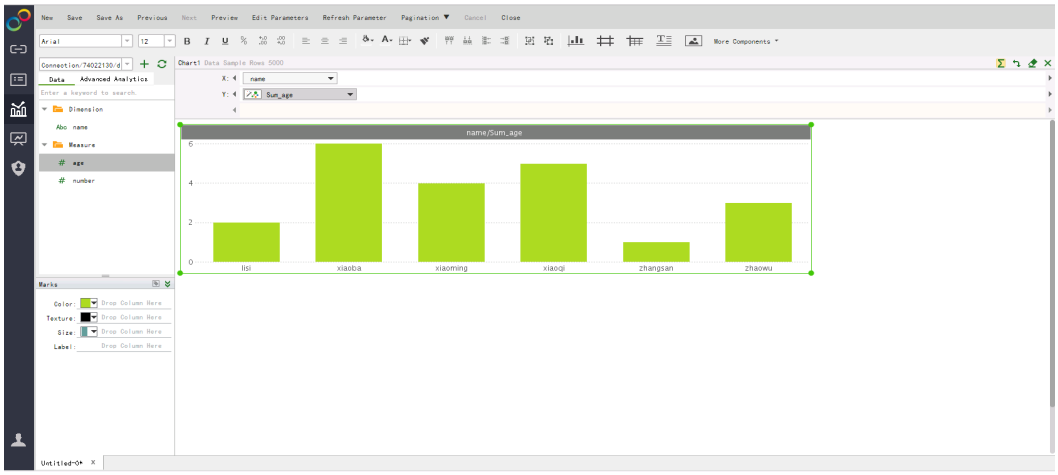
Step 4 On the report creation page, chart, table, matrix, and list filtering components are available. For example, if you want to create a chart, click  and drag it to the editing area. See **Figure 3-13**.

Figure 3-13 Creating a chart



Step 5 In **X**, choose **name**. In **Y**, choose **age**. Drag them to the corresponding area, and the system automatically generates a bar chart. See **Figure 3-14**.

Figure 3-14 Generating a chart



Step 6 On the tool bar of the displayed page, click **Save**.

----End

3.2 Interconnection Between Tableau Desktop and DLI

3.2.1 Preparing for Tableau Desktop Interconnection

Scenario

Prepare for the interconnection between Tableau Desktop and DLI.

Procedure

Step 1 Visit the Tableau official website **https://www.tableau.com/products/desktop** to obtain Tableau Desktop, and install it according to the official guide.

Step 2 Install the DLI ODBC driver and configure the data source. For details, see **Submitting a Job Using ODBC**.

NOTE

- The DLI ODBC driver is required to implement the interconnection between Tableau Desktop and DLI.

----End

3.2.2 Adding Tableau Desktop Data Source

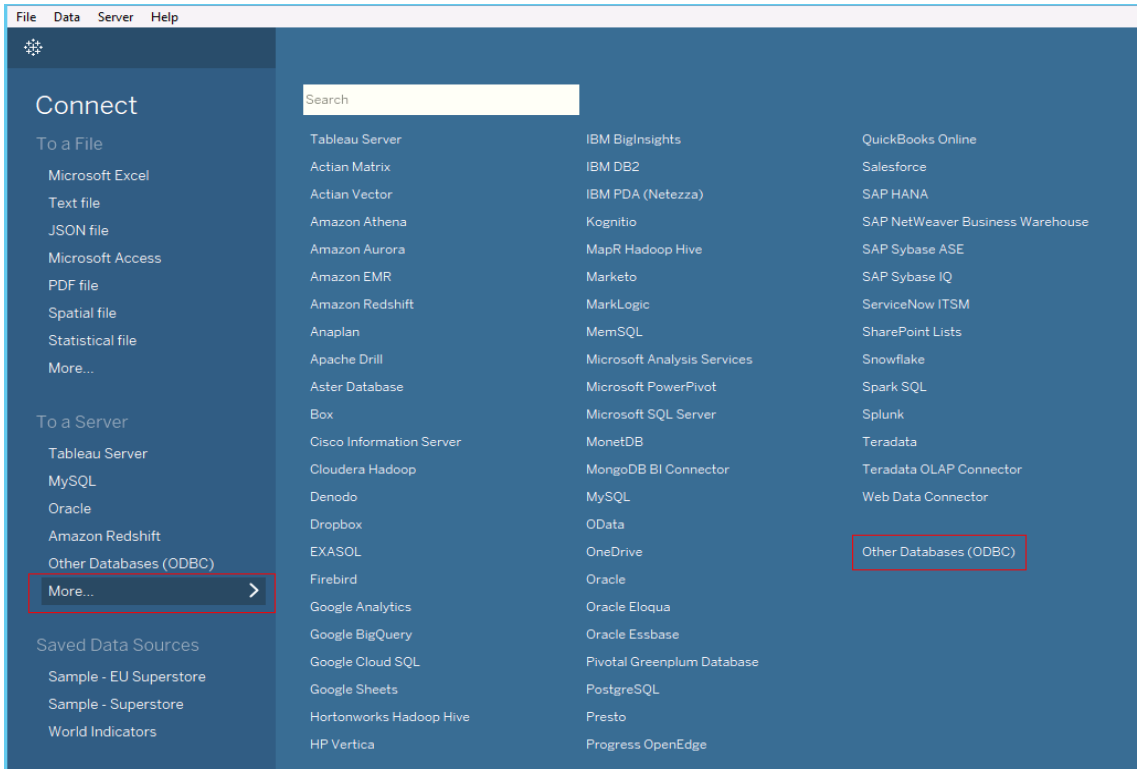
Scenario

This section describes how to add a DLI connection in Tableau Desktop.

Procedure

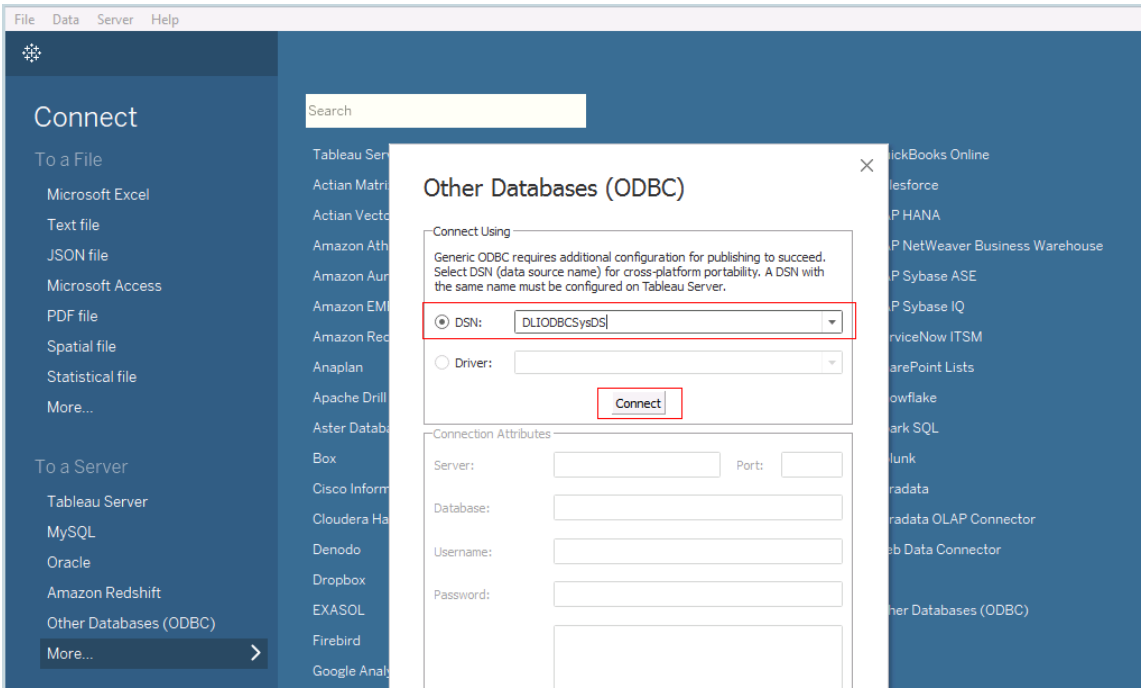
Step 1 Open Tableau Desktop and choose **Other Databases (ODBC)** in the navigation pane. See **Figure 3-15**.

Figure 3-15 Selecting ODBC



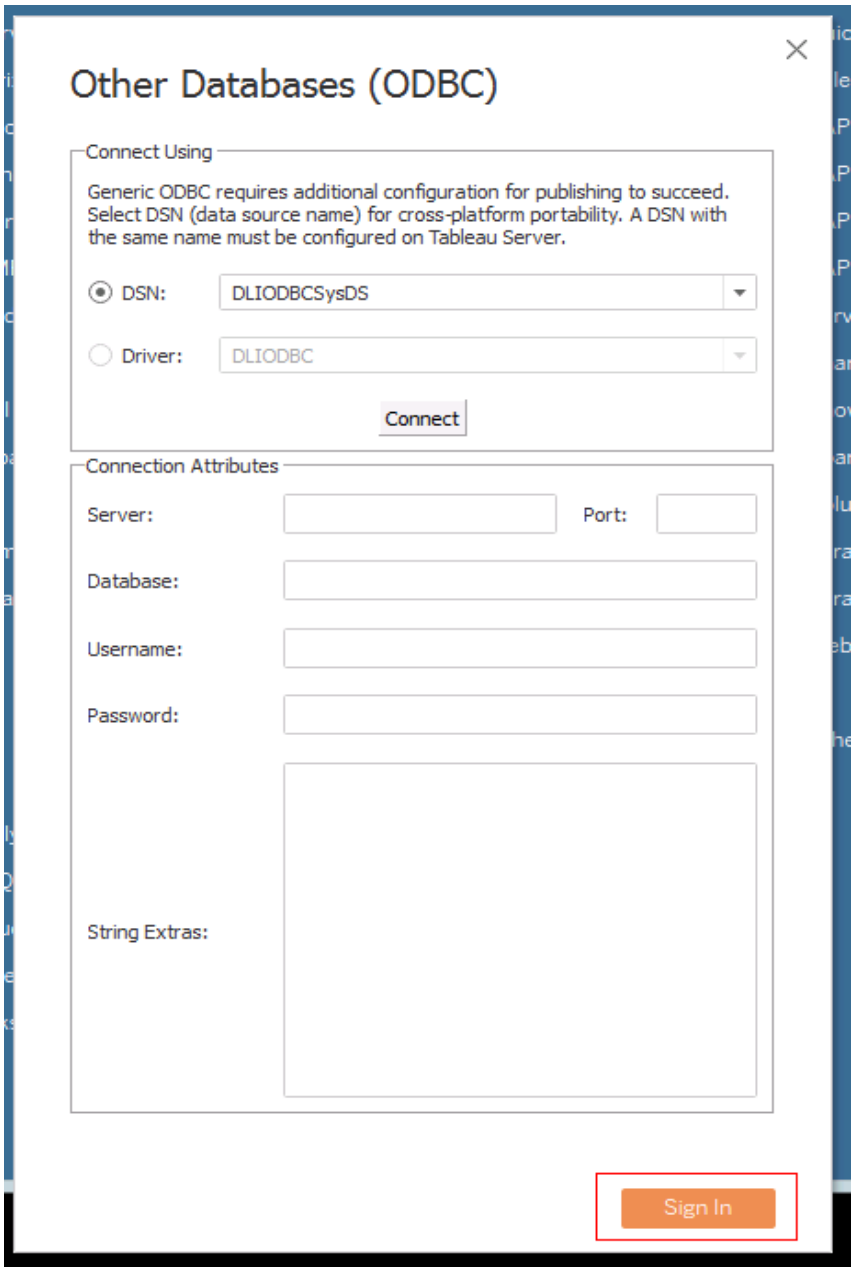
Step 2 In the displayed **Other Database (ODBC)** dialog box, select **DLIODBCSysDS** for **DSN** and click **Connect**. In this step, name **DLIODBCSysDS** is only for reference. Use the actually registered name during DLI ODBC installation. See **Figure 3-16**.

Figure 3-16 Selecting the DSN for connection



Step 3 After the connection is complete, click **Sign In** to log in to the database. See **Figure 3-17**.

Figure 3-17 Logging in to the database



----End

3.2.3 Creating Tableau Desktop Data Set

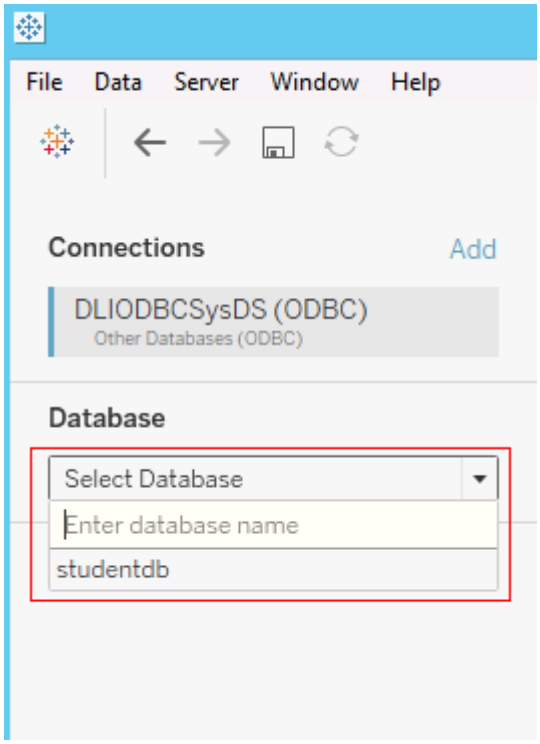
Scenario

This section describes how to create a DLI data set in Tableau Desktop.

Procedure

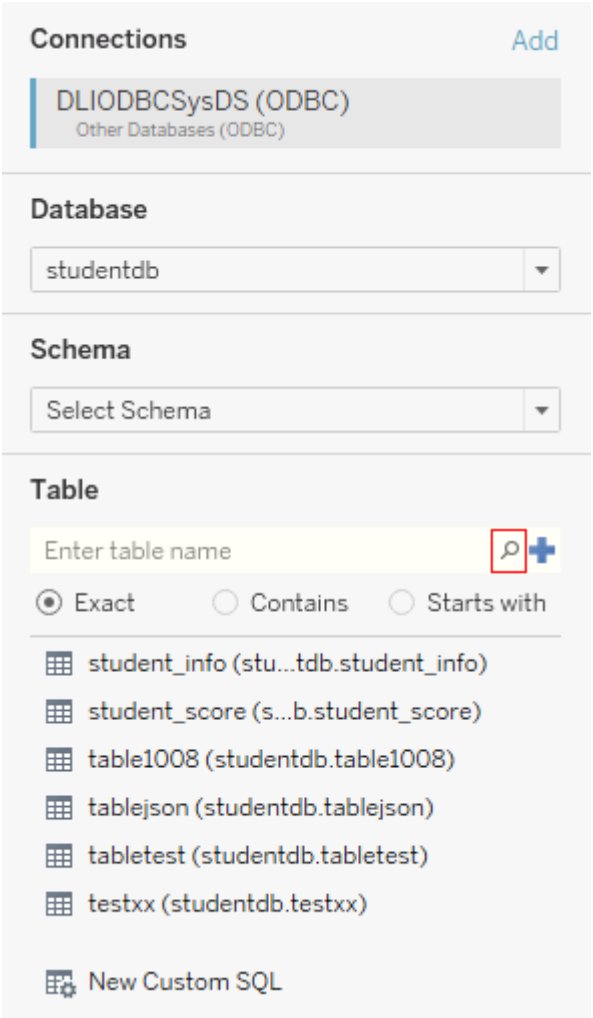
Step 1 After logging in to the Tableau Desktop, choose a database from the **Database** drop-down list box in the left navigation pane. See **Figure 3-18**.

Figure 3-18 Selecting a database



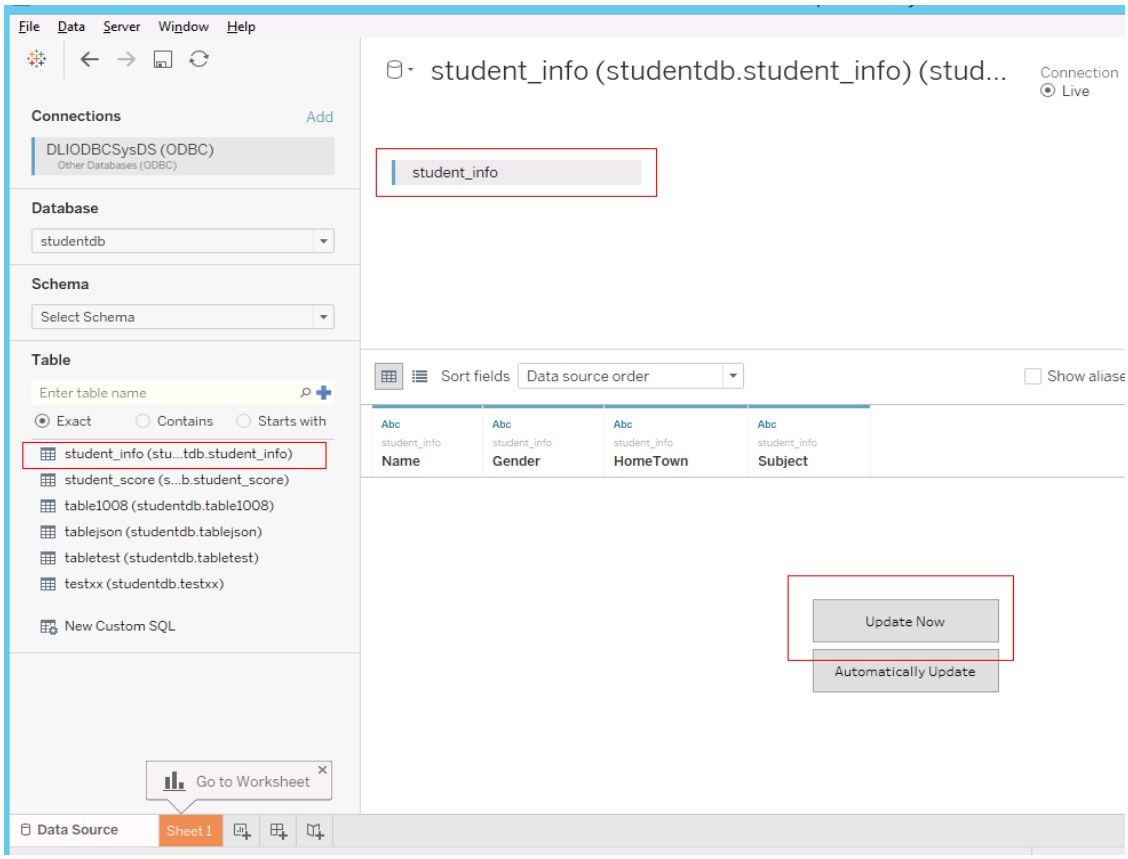
Step 2 In the left navigation pane, click under **Table** to query all tables. See **Figure 3-19**.

Figure 3-19 Querying tables



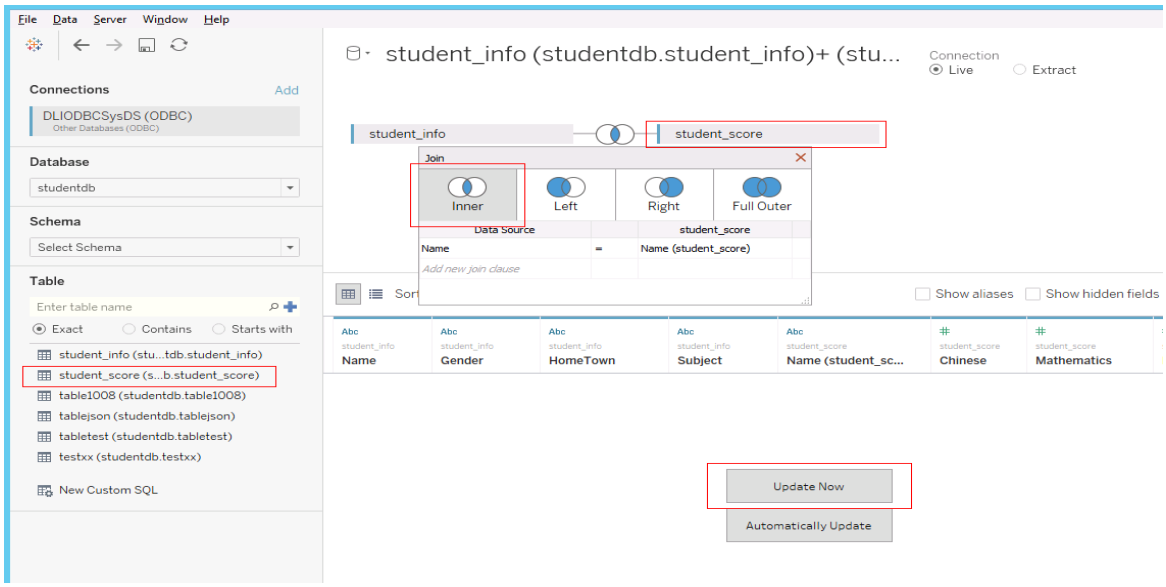
Step 3 Drag the **student_info** table to be analyzed to the upper area in the right pane and click **Update Now** to view the table data. See **Figure 3-20**.

Figure 3-20 Querying data in a table



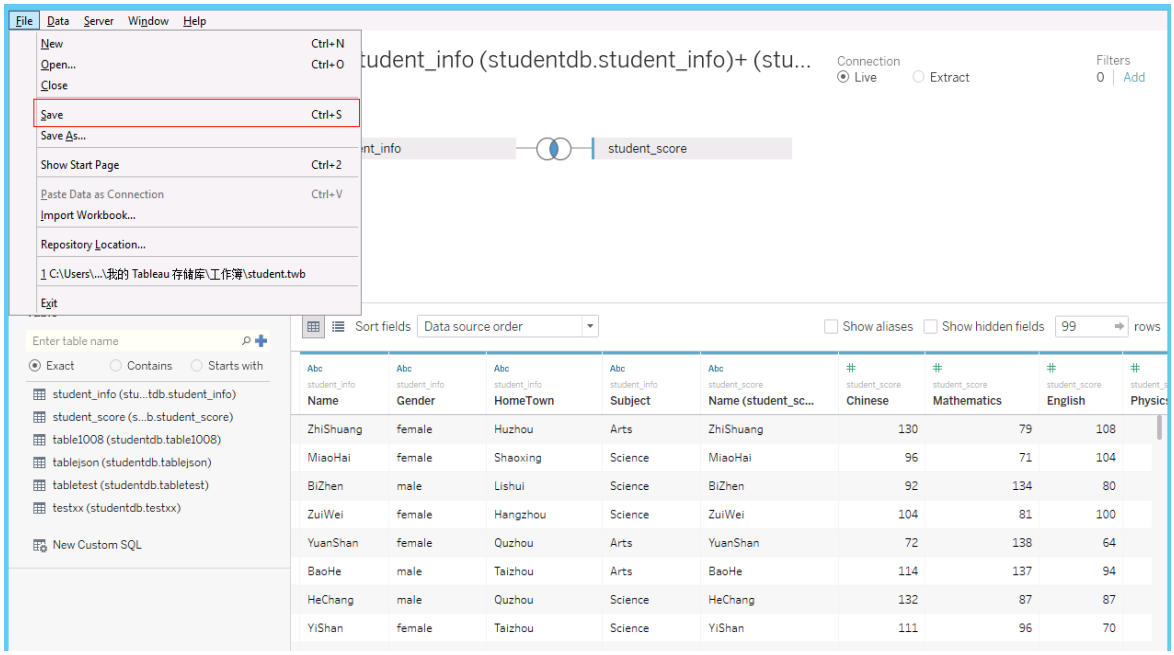
Step 4 Drag the **student_score** table to be analyzed to the upper area in the right pane and associate it with **student_info**. Click **Update Now** to view the association result. See **Figure 3-21**.

Figure 3-21 Viewing the table association result



Step 5 Choose **File** > **Save** to save the result in a Tableau Workbook. See **Figure 3-22**.

Figure 3-22 Saving the result in a Tableau Workbook



----End

3.2.4 Creating a Graph in Tableau Desktop

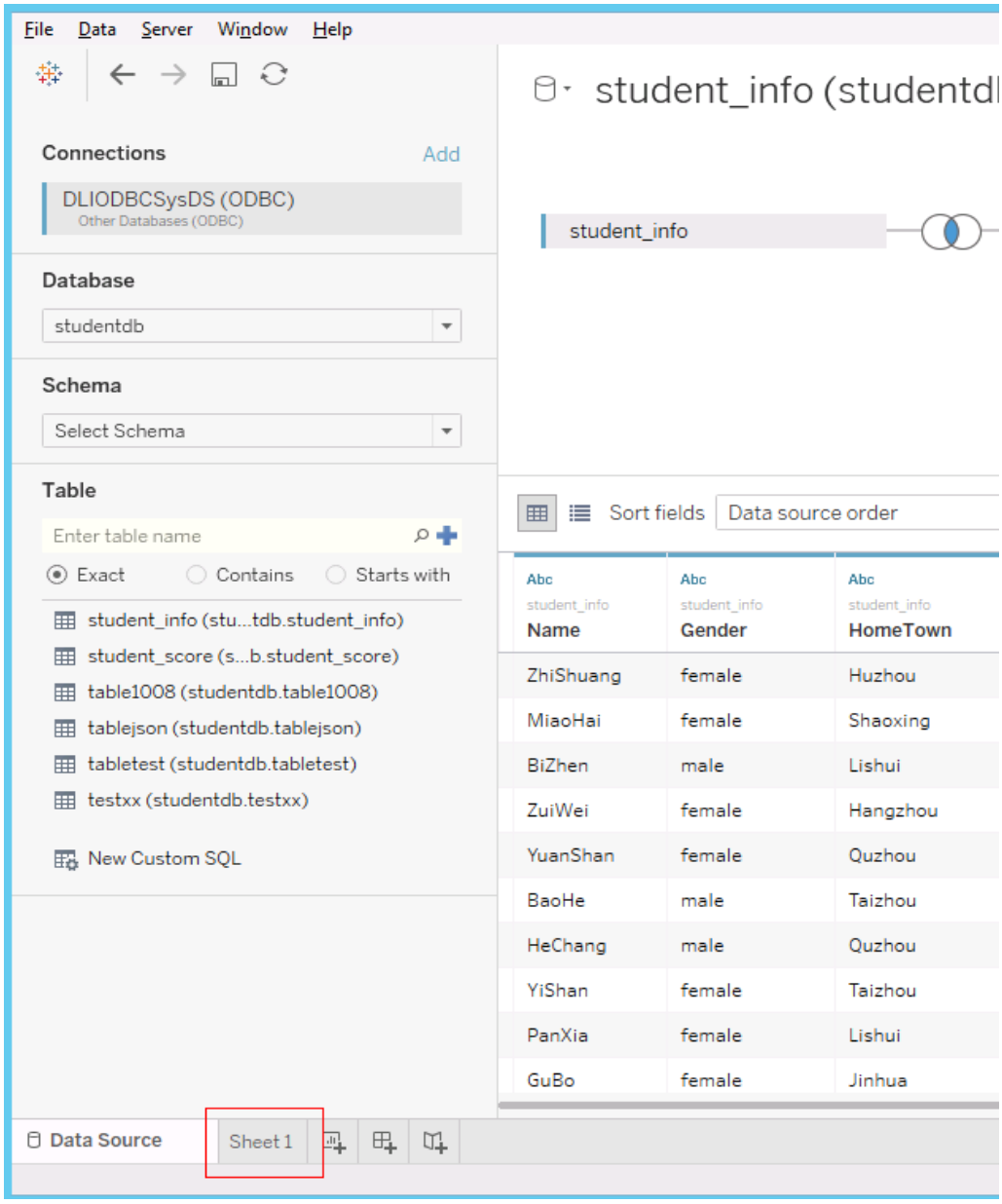
Scenario

This section describes how to create a DLI measurement graph in Tableau Desktop.

Procedure

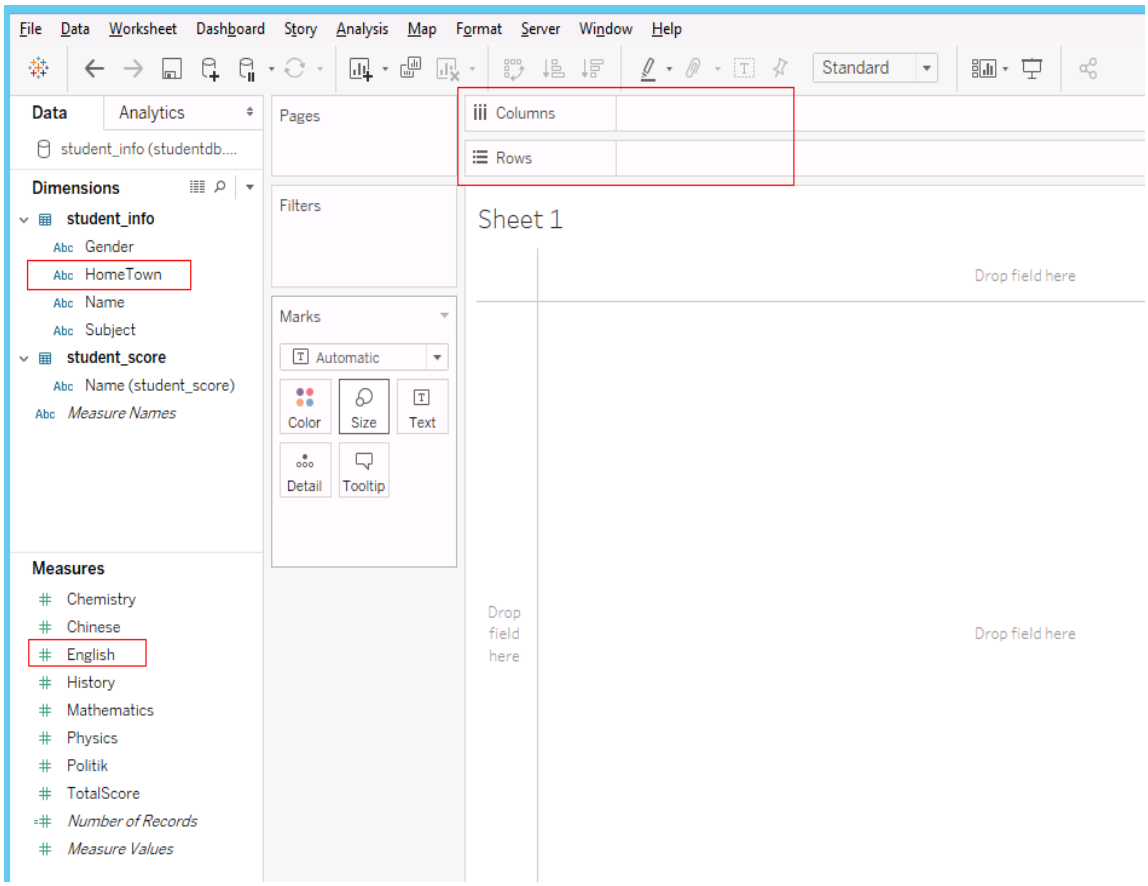
Step 1 After creating the data set worksheet, click **Worksheet** in the lower part of the window. See **Figure 3-23**.

Figure 3-23 Making a measurement graph



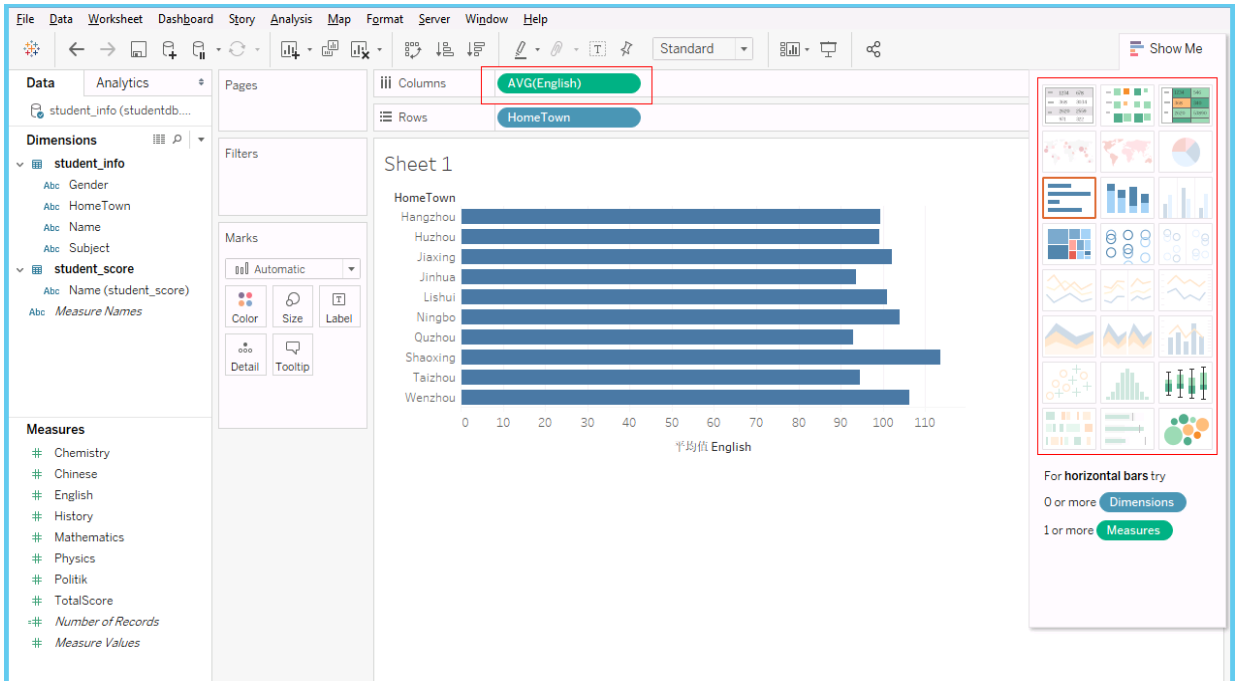
Step 2 After entering the worksheet window, drag **HomeTown** under **Dimensions** to the area next to **Rows** and **English** under **Measures** to the area next to **Columns**. See **Figure 3-24**.

Figure 3-24 Selecting the dimension and measure



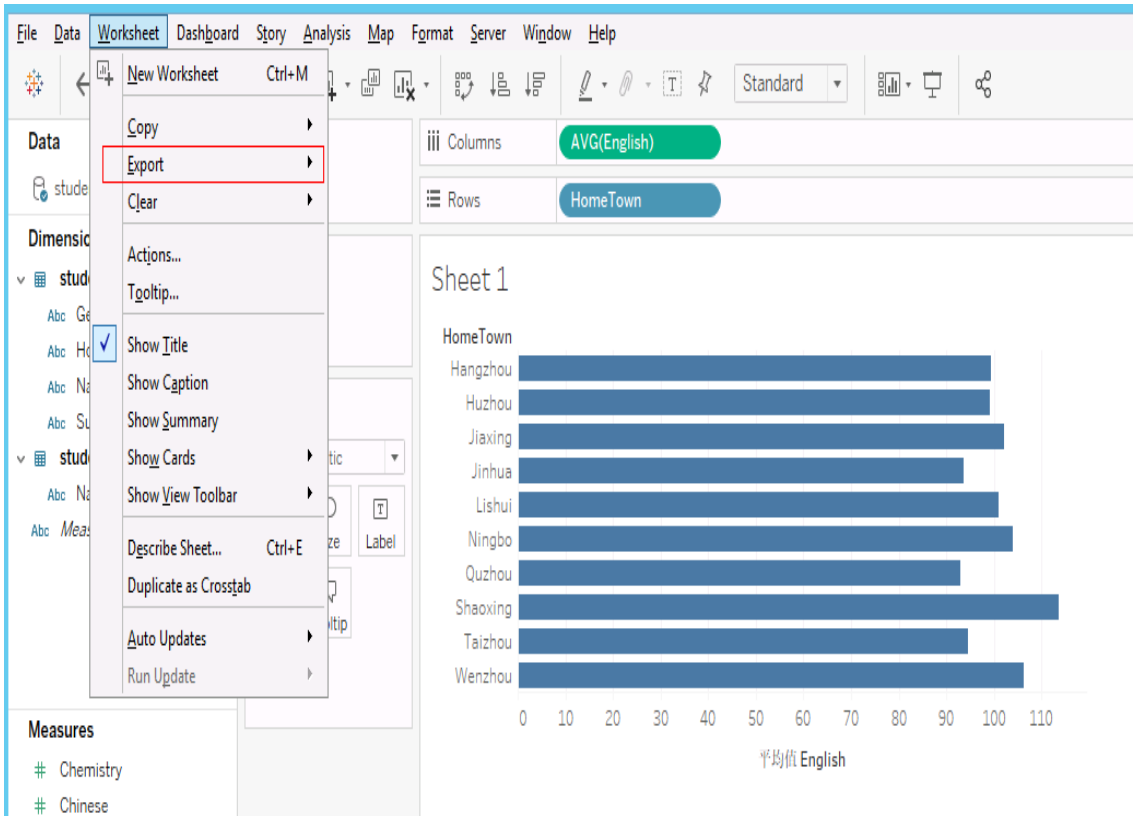
Step 3 Set the **English** score as the average value and select the graph type on the right to generate the average English score graph for each area. See **Figure 3-25**.

Figure 3-25 Generating a measurement graph



Step 4 In the main menu, choose **Worksheet** > **Export** to export the measurement graph as an image or an EXCEL file. See **Figure 3-26**.

Figure 3-26 Saving the measurement graph



----End

4 Flink Job Sample

Overview

You can perform secondary development based on Flink and Spark APIs to build your own JAR packages and submit them to the DLI queues to implement interaction with MRS Kafka, HBase, Hive, HDFS, DWS, and DCS.

This section describes the interaction between user-defined job and MRS. For more sample code, see the **DLI sample code**.

Environment Preparations

1. Log in to the **MRS management console**, create an MRS cluster, choose **enable kerberos**, and select **kafka**, **hbase**, **hdfs**. Enable the UDP/TCP port for the **Security Group Rules**.
2. Enter **MRS Manager** page.
   a. Create a machine-machine account. Ensure that you have the **hdfs_admin** and **hbase_admin** permissions. Download the user authentication credentials, including the **user.keytab** and **krb5.conf** files.
   b. Click **Services**, download the client, and click **OK**.
   c. Download the configuration files from the MRS node, including **hbase-site.xml** and **hiveclient.properties**.
3. Create an exclusive DLI queue.

   To create a dedicated DLI queue, select **Pay-per-use** for **Billing Mode** and click **Dedicated Resource Mode** for **Queue Type** when purchasing a queue. For details, see **Creating a Queue** in the *Data Lake Insight User Guide*.
4. Ensure that a datasource connection has been set up between the DLI dedicated queue and the MRS cluster, and security group rules have been configured based on the site requirements.

   For details about how to create an enhanced datasource connection, see **Enhanced Datasource Connections** in the *Data Lake Insight User Guide*.

   For details about how to configure security group rules, see **Security Group** in the **Virtual Private Cloud User Guide**.
5. Obtain the IP address and domain name mapping of all nodes in the MRS cluster, and configure the host mapping in the host information of the DLI cross-source connection.

For details about how to add an IP-domain mapping, see **Modifying the Host Information** in the *Data Lake Insight User Guide*.

NOTE

If the Kafka server listens on the port using **hostname**, you need to add the mapping between the hostname and IP address of the Kafka Broker node to the DLI queue. Contact the Kafka service deployment personnel to obtain the hostname and IP address of the Kafka Broker node.

Prerequisites

- Ensure that a dedicated queue has been created.
- When running a Flink Jar job, you need to build the secondary development application code into a Jar package and upload the JAR package to the created OBS bucket. In addition, create a package on the **Data Management** > **Package Management** page of DLI. For details, see **Creating a Package**.

  NOTE

  DLI does not support the download function. If you need to modify the uploaded data file, edit the local file and upload it again.
- Flink dependencies have been built in the DLI server and security hardening has been performed based on the open-source community version. To prevent dependency compatibility issues or log output and dump issues, exclude the following files during packaging:
  - Built-in dependencies (or set the package dependency scope to **provided** in Maven or sbt)
  - Log configuration files (for example, **log4j.properties** or **logback.xml**)
  - JAR packages for log output implementation (for example, **log4j**)

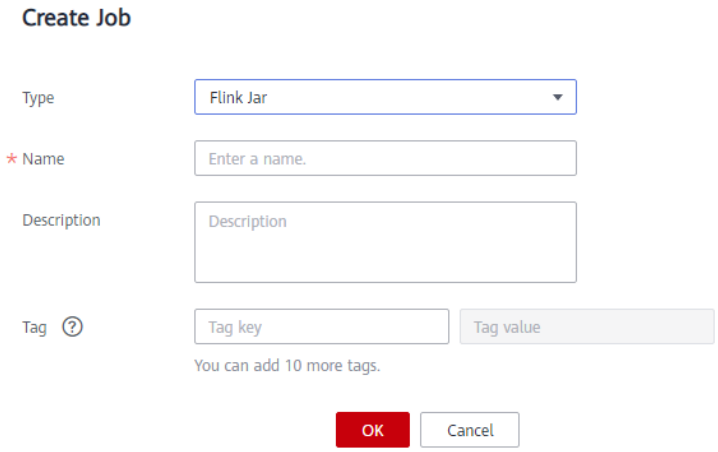
How to Use

Create and submit a Flink Jar job. For details, see **Creating a Flink Jar Job** in the *Data Lake Insight User Guide*.

Step 1 In the left navigation pane of the DLI management console, choose **Job Management** > **Flink Jobs**. The **Flink Jobs** page is displayed.

Step 2 In the upper right corner of the **Flink Jobs** page, click **Create Job**.

Figure 4-1 Creating a Flink Jar job



Step 3 Configure job parameters.

Table 4-1 Job parameters

Parameter	Description
Type	Select **Flink Jar**.
Name	Job name, which contains 1 to 57 characters and consists of only letters, digits, hyphens (-), and underscores (_). **NOTE** The job name must be globally unique.
Description	Description of the job, which contains 0 to 512 characters.
Tags	Tags are used to identify cloud resources. A tag pair includes **Tag key** and **Tag value**. If you want to use the same tag to identify multiple cloud resources, that is, to select the same tag from the drop-down list box for all services, you are advised to create predefined tags on the Tag Management Service (TMS). For details, see the **Tag Management Service User Guide**. **NOTE** • A maximum of 10 tags can be added. • Only one tag value can be added to a tag key. • **Tag key**: Enter a tag key name in the text box. **NOTE** – A tag key contains a maximum of 36 characters. The first and last characters cannot be spaces. The following characters are not allowed: =*,<>|/ – If there are predefined tags, you can select a tag from the drop-down list box. • **Tag value**: Enter a tag value in the text box. **NOTE** – A tag value contains a maximum of 43 characters. The first and last characters cannot be spaces. The following characters are not allowed: =*,<>|/ – If there are predefined tags, you can select a tag from the drop-down list box.

Step 4 Click **OK** to enter the **Edit** page.

Step 5 Select a queue. A Flink Jar job can run only on general queues.

NOTE

- A Flink Jar job can run only on a pre-created dedicated queue.
- If no dedicated queue is available in the **Queue** drop-down list, create a dedicated queue and bind it to the current user.

Figure 4-2 Selecting a queue

Step 6 Upload the JAR package.

Figure 4-3 Uploading the JAR package

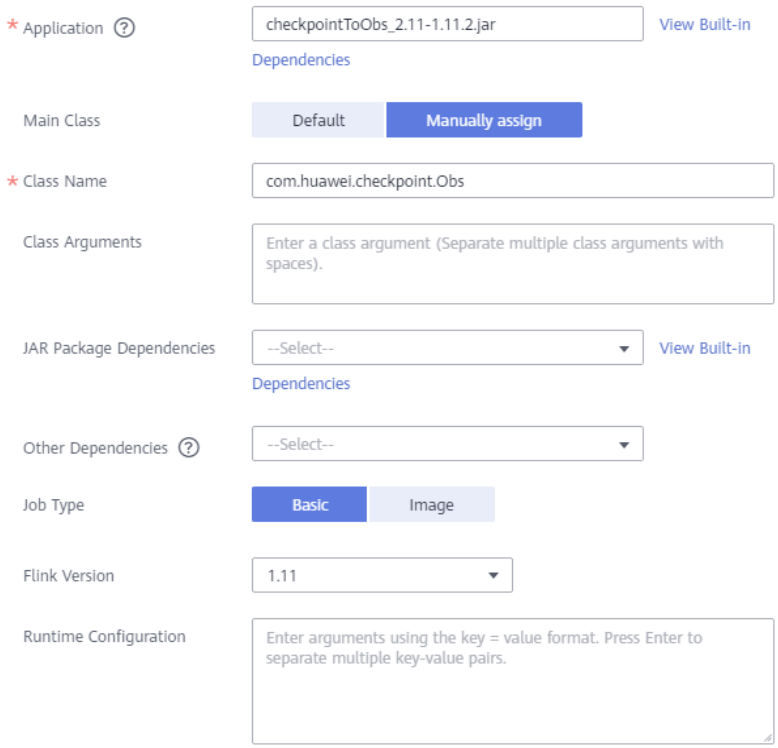


Table 4-2 Parameter description

Name	Description
Application	User-defined package. Before selecting a package, upload the corresponding JAR package to the OBS bucket and create a package on the **Data Management** > **Package Management** page. For details, see **Creating a Package**.
Main Class	Name of the main class of the JAR package to be loaded, for example, **KafkaMessageStreaming**. • **Default**: The value is specified based on the **Manifest** file in the JAR package. • **Manually assign**: You must enter the class name and confirm the class arguments (separate arguments with spaces). **NOTE** When a class belongs to a package, the package path must be carried, for example, **packagePath.KafkaMessageStreaming**.

Name	Description
Class Arguments	List of arguments of a specified class. The arguments are separated by spaces.
JAR Package Dependencies	User-defined dependencies. Before selecting a package, upload the corresponding JAR package to the OBS bucket and create a JAR package on the **Data Management** > **Package Management** page. For details, see **Creating a Package**.
Other Dependencies	User-defined dependency files. Before selecting a file, upload the corresponding file to the OBS bucket and create a package of any type on the **Data Management** > **Package Management** page. For details, see **Creating a Package**. You can add the following content to the application to access the corresponding dependency file: **fileName** indicates the name of the file to be accessed, and **ClassName** indicates the name of the class that needs to access the file. `ClassName.class.getClassLoader().getResource("userData/fileName")`
Job Type	This parameter is displayed when the queue type is CCE. • **Basic** • **Image**: Select the image name and image version. Images are set on the Software Repository for Container (SWR) console. For details, see the **SoftWare Repository for Container User Guide**.
Flink Version	Before selecting a Flink version, you need to select the queue to which the Flink version belongs. Currently, the following versions are supported: 1.10 and 1.11.

Step 7 Configure job parameters.

Figure 4-4 Configuring parameters

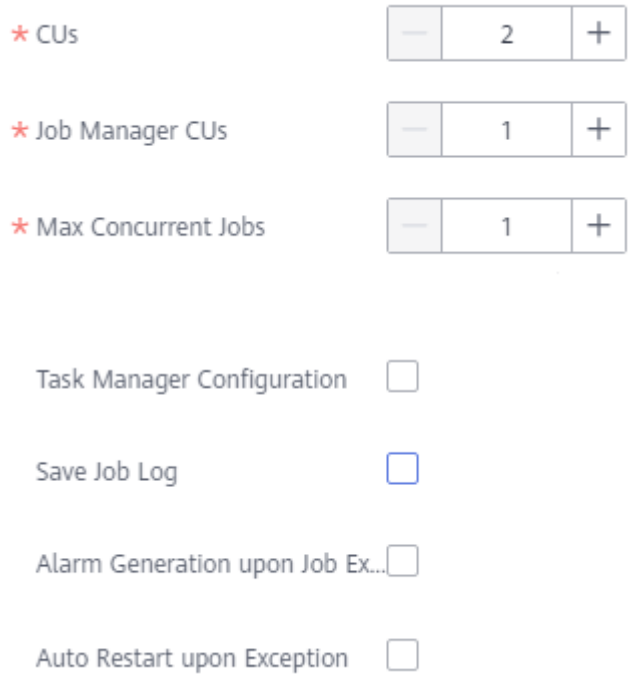


Table 4-3 Parameter description

Parameter	Description
CUs	One CU has one vCPU and 4 GB memory. The number of CUs ranges from 2 to 400.
Job Manager CUs	Set the number of CUs on a management unit. The value ranges from 1 to 4. The default value is **1**.
Max Concurrent Jobs	Maximum number of parallel operators in a job. **NOTE** • The value must be less than or equal to four times the number of compute units (CUs minus the number of job manager CUs). • You are advised to set this parameter to a value greater than that configured in the code. Otherwise, job submission may fail.
Task Manager Configuration	Whether to set Task Manager resource parameters. If this option is selected, you need to set the following parameters: • **CU(s) per TM**: Number of resources occupied by each Task Manager. • **Slot(s) per TM**: Number of slots contained in each Task Manager.

Parameter	Description
Save Job Log	Whether to save the job running logs to OBS. If this option is selected, you need to set the following parameters: **OBS Bucket**: Select an OBS bucket to store user job logs. If the selected OBS bucket is not authorized, click **Authorize**.
Alarm Generation upon Job Exception	Whether to report job exceptions, for example, abnormal job running or exceptions due to an insufficient balance, to users via SMS or email. If this option is selected, you need to set the following parameters: **SMN Topic** Select a user-defined SMN topic. For details about how to customize SMN topics, see "Creating a Topic" in the **Simple Message Notification User Guide**.
Auto Restart upon Exception	Whether to enable automatic restart. If this function is enabled, jobs will be automatically restarted and restored when exceptions occur. If this option is selected, you need to set the following parameters: • **Max. Retry Attempts**: maximum number of retry times upon an exception. The unit is **Times/hour**. – **Unlimited**: The number of retries is unlimited. – **Limited**: The number of retries is user-defined. • **Restore Job from Checkpoint**: Restore the job from the saved checkpoint. If you select this parameter, you also need to set **Checkpoint Path**. **Checkpoint Path**: Select the checkpoint saving path. The value must be the same as the checkpoint path you set in the application package. Note that the checkpoint path for each job must be unique. Otherwise, the checkpoint cannot be obtained.

Step 8 Click **Save** on the upper right of the page.

Step 9 Click **Start** on the upper right side of the page. On the displayed **Start Flink Job** page, confirm the job specifications and the price, and click **Start Now** to start the job.

After the job is started, the system automatically switches to the **Flink Jobs** page, and the created job is displayed in the job list. You can view the job status in the **Status** column. After a job is successfully submitted, the job status will change from **Submitting** to **Running**. After the execution is complete, the message **Completed** is displayed.

If the job status is **Submission failed** or **Running exception**, the job submission failed or the job did not execute successfully. In this case, you can move the cursor over the status icon in the **Status** column of the job list to view the error details. You can click to copy these details. After handling the fault based on the provided information, resubmit the job.

NOTE

Other buttons are as follows:

Save As: Save the created job as a new job.

----End

5 Stream Ecosystem Development Guide

Overview

Built on Flink and Spark, the stream ecosystem is fully compatible with the open-source Flink, Storm, and Spark APIs. It is enhanced in features and improved in performance to provide easy-to-use DLI with low latency and high throughput.

The stream ecosystem development of DLI includes the cloud service ecosystem, open source ecosystem, and custom stream ecosystem.

- Cloud service ecosystems

  DLI can be interconnected with other services by using Stream SQLs. You can directly use SQL statements to read and write data from various cloud services, such as Data Ingestion Service (DIS), Object Storage Service (OBS), CloudTable Service (CloudTable), MapReduce Service (MRS), Relational Database Service (RDS), Simple Message Notification (SMN), and Distributed Cache Service (DCS).
- Open-source ecosystems

  After connections to other VPCs are established through VPC peering connections, you can access all data sources and output targets (such as Kafka, HBase, and Elasticsearch) supported by Flink and Spark in your dedicated DLI queues.
- Custom stream ecosystems

  You can compile code to obtain data from the desired cloud ecosystem or open-source ecosystem as the input data of Flink jobs.

Cloud Service Ecosystem Development

Table 5-1 Development overview

Data Source	SQL		Custom Stream Job
	Input Stream	Output Stream	
CloudTable	**HBase Source Stream**	• **HBase Sink Stream** • **OpenTSDB Sink Stream**	-
CSS	-	**Elasticsearch Sink Stream**	-

DCS	-	DCS Sink Stream	Connecting to Other Services Through Custom Stream Jobs
DDS	-	DDS Sink Stream	-
DIS	DIS Source Stream	DIS Sink Stream	-
DMS	DMS Source Stream	DMS Sink Stream	-
DWS	-	• DWS Sink Stream (JDBC Mode) • DWS Sink Stream (OBS Mode)	Connecting to Other Services Through Custom Stream Jobs
MRS	MRS Kafka Source Stream	• MRS Kafka Sink Stream • MRS HBase Sink Stream	Connecting to Other Services Through Custom Stream Jobs
OBS	OBS Source Stream	OBS Sink Stream	-
RDS	-	RDS Sink Stream	-
SMN	-	SMN Sink Stream	-

Open-source Ecosystem Development

Table 5-2 Development overview

Data Source	SQL		Custom Stream Job
	Input Stream	Output Stream	
Apache Kafka	Open-Source Kafka Source Stream	Open-Source Kafka Sink Stream	-

Custom Stream Ecosystem Development

Table 5-3 Development overview

Data Source	SQL		Custom Stream Job
	Input Stream	Output Stream	
Custom	**Custom Source Stream**	**Custom Sink Stream**	-

Supported Data Formats

DLI Flink jobs support the following data formats:

Avro, Avro_merge, BLOB, CarbonData, CSV, EMAIL, JSON, ORC, Parquet, and XML.

Table 5-4 Supported source and sink streams of corresponding data formats

Data Format	Supported Source Stream	Supported Sink Stream
Avro	-	**OBS Sink Stream**
Avro_merge	-	**OBS Sink Stream**
BLOB	• **DIS Source Stream** • **MRS Kafka Source Stream** • **Open-Source Kafka Source Stream**	-
CarbonData	-	**OBS Sink Stream**
CSV	• **DIS Source Stream** • **OBS Source Stream** • **Open-Source Kafka Source Stream**	• **DIS Sink Stream** • **OBS Sink Stream** • **DWS Sink Stream (OBS Mode)** • **Open-Source Kafka Sink Stream** • **File System Sink Stream**
EMAIL	**DIS Source Stream**	-
JSON	• **DIS Source Stream** • **OBS Source Stream** • **MRS Kafka Source Stream** • **Open-Source Kafka Source Stream**	• **DIS Sink Stream** • **OBS Sink Stream** • **MRS Kafka Sink Stream** • **Open-Source Kafka Sink Stream**
ORC	-	• **OBS Sink Stream** • **DWS Sink Stream (OBS Mode)**

Data Format	Supported Source Stream	Supported Sink Stream
Parquet	-	• **OBS Sink Stream** • **File System Sink Stream**
XML	**DIS Source Stream**	-

6 Using the Spark Job to Access Data Sources of Datasource Connections

6.1 Overview

DLI supports the native Spark Data Source APIs. Expansions are also made to query and analysis data from other services through SQL syntax and Spark jobs. Currently, the DLI datasource connection supports the following services: CloudTable, CSS, DCS, DDS, DWS, MRS, and RDS. To use the datasource connection capability of DLI, you need to create a datasource connection first. For details about how to create a datasource connection on the management console, see the **Data Lake Insight User Guide**. You can use Scala, PySpark, and Java to develop Spark jobs for data sources in datasource connections.

6.2 Connecting to CSS

6.2.1 Scala Example Code

Prerequisites

A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

CSS Non-security Cluster

- Development description
  - Construct dependency information and create a Spark session.
    i. Import dependencies

       Involved Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```


Dependencies related to **import**

```
import org.apache.spark.sql.{Row, SaveMode, SparkSession}
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
```

ii. Create a session

```
val sparkSession = SparkSession.builder().getOrCreate()
```

- Connecting to datasources through SQL APIs

i. Create a table to connect to CSS datasource

```
sparkSession.sql("create table css_table(id int, name string) using css options(
  'nodes' 'to-css-1174404221-Y2bKVIqY.datasource.com:9200',
  'nodes.wan.only'='true',
  'resource' '/mytest/css')")
```

Table 6-1 Parameters for creating a table

Parameter	Description
es.nodes	CSS connection address. You need to create a datasource connection first. For details, see **Data Lake Insight User Guide**. After a basic datasource connection is created, the returned IP address is used. After an enhanced datasource connection is created, use the intranet IP address provided by CSS. The address format is ***IP1*:*PORT1*,*IP2*:*PORT2***.
resource	The **resource** is used to specify the CSS datasource connection name. You can use **/index/type** to specify the resource location (for easier understanding, the **index** can be seen as **database** and **type** as **table**). **NOTE** • In Elasticsearch 6.X, a single index supports only one type, and the type name can be customized. • In Elasticsearch 7.X, a single index uses **_doc** as the type name and cannot be customized. To access Elasticsearch 7.X, set this parameter to **index**.
pushdown	Indicates whether the press function of CSS is enabled. The default value is set to **true**. If there are a large number of I/O transfer tables, the **pushdown** can be enabled to reduce I/Os when the **where** filtering conditions are met.
strict	Indicates whether the CSS **pushdown** is strict. The default value is set to **false**. In exact match scenarios, more I/Os are reduced than **pushdown**.
batch.size.entries	Maximum number of entries that can be inserted to a batch processing. The default value is **1000**. If the size of a single data record is so large that the number of data records in the bulk storage reaches the upper limit of the data amount of a single batch processing, the system stops storing data and submits the data based on the **batch.size.bytes**.

Parameter	Description
batch.size.bytes	Maximum amount of data in a single batch processing. The default value is 1 MB. If the size of a single data record is so small that the number of data records in the bulk storage reaches the upper limit of the data amount of a single batch processing, the system stops storing data and submits the data based on the **batch.size.entries**.
es.nodes.wan.only	Indicates whether to access the Elasticsearch node using only the domain name. The default value is **false**. If a basic datasource connection address is used as the **es.nodes**, set this parameter to **true**. If the original internal IP address provided by CSS is used as the **es.nodes**, you do not need to set this parameter or set it to **false**.
es.mapping.id	Specifies a field whose value is used as the document ID in the Elasticsearch node. **NOTE** • The document ID in the same **/index/type** is unique. If a field that functions as a document ID has duplicate values, the document with the duplicate ID will be overwritten when the ES is inserted. • This feature can be used as a fault tolerance solution. When data is being inserted, the DLI job fails and some data has been inserted into Elasticsearch. The data is redundant. If Document id is set, the last redundant data will be overwritten when the DLI job is executed again.

NOTE

batch.size.entries and **batch.size.bytes** limit the number of data records and data volume respectively.

ii. Insert data

```
sparkSession.sql("insert into css_table values(13, 'John'),(22, 'Bob')")
```

iii. Query data

```
val dataFrame = sparkSession.sql("select * from css_table")
dataFrame.show()
```

Before data is inserted:

```
+---+----+\n
| id|name|\n
+---+----+\n
|  1|John|\n
|  2| Bob|\n
+---+----+\n
```

After data is inserted:


```
+---+----+\n
| id|name|\n
+---+----+\n
|  1|John|\n
|  2| Bob|\n
| 13|John|\n
| 22| Bob|\n
+---+----+\n
```

iv. Delete the datasource connection table

```
sparkSession.sql("drop table css_table")
```

- Connecting to datasources through DataFrame APIs

i. Configure datasource connection

```
val resource = "/mytest/css"
val nodes = "to-css-1174405013-Ht7O1tYf.datasource.com:9200"
```

ii. Create a schema and add data to the schema

```
val schema = StructType(Seq(StructField("id", IntegerType, false), StructField("name", StringType, false)))
val rdd = sparkSession.sparkContext.parallelize(Seq(Row(12, "John"),Row(21,"Bob")))
```

iii. Import data to CSS

```
val dataFrame_1 = sparkSession.createDataFrame(rdd, schema)
dataFrame_1.write
 .format("css")
 .option("resource", resource)
 .option("nodes", nodes)
 .mode(SaveMode.Append)
 .save()
```

NOTE

The value of **SaveMode** can be one of the following:

- ErrorIfExis: If the data already exists, the system throws an exception.
- Overwrite: If the data already exists, the original data will be overwritten.
- Append: If the data already exists, the system saves the new data.
- Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

iv. Read data from CSS

```
val dataFrameR =
sparkSession.read.format("css").option("resource",resource).option("nodes", nodes).load()
dataFrameR.show()
```

Before data is inserted:

```
+---+----+\n
| id|name|\n
+---+----+\n
|  1|John|\n
| 22| Bob|\n
+---+----+\n
```

After data is inserted:


```
+---+----+\n
| id|name|\n
+---+----+\n
|  1|John|\n
| 12|John|\n
| 21| Bob|\n
| 22| Bob|\n
+---+----+\n
```

- Submitting a Spark job
  i. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.
  ii. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.css**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

- Complete example code
  - Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

  - Connecting to datasources through SQL APIs

```
import org.apache.spark.sql.SparkSession

object Test_SQL_CSS {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    // Create a DLI data table for DLI-associated CSS
    sparkSession.sql("create table css_table(id long, name string) using css options(
    'nodes' = 'to-css-1174404217-QG2SwbVV.datasource.com:9200',
    'nodes.wan.only' = 'true',
    'resource' = '/mytest/css')")

    //*****************************SQL model***********************************
    // Insert data into the DLI data table
    sparkSession.sql("insert into css_table values(13, 'John'),(22, 'Bob')")

    // Read data from DLI data table
    val dataFrame = sparkSession.sql("select * from css_table")
    dataFrame.show()

    // drop table
    sparkSession.sql("drop table css_table")
```



```
    sparkSession.close()
  }
}
```

- Connecting to datasources through DataFrame APIs

```
import org.apache.spark.sql.{Row, SaveMode, SparkSession};
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType};

object Test_SQL_CSS {
  def main(args: Array[String]): Unit = {
    //Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    //****************************DataFrame model***********************************
    // Setting the /index/type of CSS
    val resource = "/mytest/css"

    // Define the cross-origin connection address of the CSS cluster
    val nodes = "to-css-1174405013-Ht7O1tYf.datasource.com:9200"

    //Setting schema
    val schema = StructType(Seq(StructField("id", IntegerType, false), StructField("name",
StringType, false)))

    // Construction data
    val rdd = sparkSession.sparkContext.parallelize(Seq(Row(12, "John"),Row(21,"Bob")))

    // Create a DataFrame from RDD and schema
    val dataFrame_1 = sparkSession.createDataFrame(rdd, schema)

   //Write data to the CSS
   dataFrame_1.write .format("css")
    .option("resource", resource)
    .option("nodes", nodes)
    .mode(SaveMode.Append)
    .save()

    //Read data
    val dataFrameR = sparkSession.read.format("css").option("resource",
resource).option("nodes", nodes).load()
    dataFrameR.show()

    spardSession.close()
  }
}
```

CSS Security Cluster

- Preparations
  - The Elasticsearch 6.5.4 provided by CSS provides the security settings. Once the function is enabled, CSS provides identity authentication, authorization, and encryption for users. Before connecting DLI to the CSS security cluster, you need to perform certain preparations.
    i. Select CSS Elasticsearch 6.5.4 or a later cluster version, create a CSS security cluster, and download the security cluster certificate (**CloudSearchService.cer**).
    ii. Create a datasource connection.
    iii. Use the **keytools** tool to generate the **keystore** and **truststore** files.
      1) While generating the required files, the security certificate (**CloudSearchService.cer**) of the security cluster is required. Run the following commands to generate the files. Other parameters of the **keytools** tool can be set as required.


```
keytool -genkeypair -alias certificatekey -keyalg RSA -keystore transport-keystore.jks
keytool -list -v -keystore transport-keystore.jks
keytool -import -alias certificatekey -file CloudSearchService.cer  -keystore
truststore.jks
keytool -list -v -keystore truststore
```

2) Upload the generated **keystore** and **truststore** files to an OBS bucket.

- Development description
  - Construct dependency information and create a Spark session.
    i. Import dependencies

    Involved Maven dependency

    ```
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_2.11</artifactId>
      <version>2.3.2</version>
    </dependency>
    ```

    Dependencies related to **import**

    ```
    import org.apache.spark.sql.{Row, SaveMode, SparkSession}
    import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
    ```

    ii. Create a session and set the access keys.

    ```
    val sparkSession = SparkSession.builder().getOrCreate()
    sparkSession.conf.set("fs.obs.access.key", ak)
    sparkSession.conf.set("fs.obs.secret.key", sk)
    sparkSession.conf.set("fs.obs.endpoint", enpoint)
    sparkSession.conf.set("fs.obs.connecton.ssl.enabled", "false")
    ```

  - Connecting to datasources through DataFrame APIs
    i. Configure datasource connection

    ```
    val resource = "/mytest/css"
    val nodes = "to-css-1174405013-Ht7O1tYf.datasource.com:9200"
    ```

    ii. Create a schema and add data to the schema

    ```
    val schema = StructType(Seq(StructField("id", IntegerType, false), StructField("name",
    StringType, false)))
    val rdd = sparkSession.sparkContext.parallelize(Seq(Row(12, "John"),Row(21,"Bob")))
    ```

    iii. Import data to CSS

    ```
    val dataFrame_1 = sparkSession.createDataFrame(rdd, schema)
    dataFrame_1.write
      .format("css")
      .option("resource", resource)
      .option("nodes", nodes)
      .option("es.net.ssl", "true")
      .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-
    keystore.jks")
      .option("es.net.ssl.keystore.pass", "***")
      .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
      .option("es.net.ssl.truststore.pass", "***")
      .option("es.net.http.auth.user", "admin")
      .option("es.net.http.auth.pass", "***")
      .mode(SaveMode.Append)
      .save()
    ```

    NOTE

    The value of **SaveMode** can be one of the following:

    - ErrorIfExis: If the data already exists, the system throws an exception.
    - Overwrite: If the data already exists, the original data will be overwritten.
    - Append: If the data already exists, the system saves the new data.
    - Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

iv. Read data from CSS

```
val dataFrameR = sparkSession.read.format("css")
      .option("resource",resource)
      .option("nodes", nodes)
      .option("es.net.ssl", "true")
      .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-keystore.jks")
      .option("es.net.ssl.keystore.pass", "***")
      .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
      .option("es.net.ssl.truststore.pass", "***")
      .option("es.net.http.auth.user", "admin")
      .option("es.net.http.auth.pass", "***")
      .load()
dataFrameR.show()
```

Before data is inserted:

```
+---+----+\n
| id|name|\n
+---+----+\n
|  1|John|\n
| 22| Bob|\n
+---+----+\n
```

After data is inserted:

```
+---+----+\n
| id|name|\n
+---+----+\n
|  1|John|\n
| 12|John|\n
| 21| Bob|\n
| 22| Bob|\n
+---+----+\n
```

– Submitting a Spark job

i. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

ii. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

NOTE

- When submitting a job, you need to specify a dependency module named **sys.datasource.css**.
- For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
- For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

- Complete example code
  – Maven dependency

```
<dependency>
 <groupId>org.apache.spark</groupId>
```



```
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

- Connecting to datasources through DataFrame APIs

```
import org.apache.spark.sql.{Row, SaveMode, SparkSession};
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType};

object Test_SQL_CSS {
  def main(args: Array[String]): Unit = {
    //Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()
    sparkSession.conf.set("fs.obs.access.key", ak)
    sparkSession.conf.set("fs.obs.secret.key", sk)

    //****************************DataFrame model***********************************
    // Setting the /index/type of CSS
    val resource = "/mytest/css"

    // Define the cross-origin connection address of the CSS cluster
    val nodes = "to-css-1174405013-Ht7O1tYf.datasource.com:9200"

    //Setting schema
    val schema = StructType(Seq(StructField("id", IntegerType, false), StructField("name",
StringType, false)))

    // Construction data
    val rdd = sparkSession.sparkContext.parallelize(Seq(Row(12, "John"),Row(21,"Bob")))

    // Create a DataFrame from RDD and schema
    val dataFrame_1 = sparkSession.createDataFrame(rdd, schema)

   //Write data to the CSS
   dataFrame_1.write .format("css")
    .option("resource", resource)
    .option("nodes", nodes)
    .option("es.net.ssl", "true")
    .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-keystore.jks")
    .option("es.net.ssl.keystore.pass", "***")
    .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
    .option("es.net.ssl.truststore.pass", "***")
    .option("es.net.http.auth.user", "admin")
    .option("es.net.http.auth.pass", "***")
    .mode(SaveMode.Append)
    .save();

    //Read data
    val dataFrameR = sparkSession.read.format("css")
    .option("resource", resource)
    .option("nodes", nodes)
    .option("es.net.ssl", "true")
    .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-keystore.jks")
    .option("es.net.ssl.keystore.pass", "***")
    .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
    .option("es.net.ssl.truststore.pass", "***")
    .option("es.net.http.auth.user", "admin")
    .option("es.net.http.auth.pass", "***")
    .load()
    dataFrameR.show()

    spardSession.close()
  }
}
```


6.2.2 PySpark Example Code

Prerequisites

A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

CSS Non-security Cluster

- Development description
  - Code implementation
    i. Dependencies related to **import**
    ```
    from __future__ import print_function
    from pyspark.sql.types import StructType, StructField, IntegerType, StringType
    from pyspark.sql import SparkSession
    ```
    ii. Create a session
    ```
    sparkSession = SparkSession.builder.appName("datasource-css").getOrCreate()
    ```
  - Connecting to datasources through DataFrame APIs
    i. Configure datasource connection
    ```
    resource = "/mytest/css"
    nodes = "to-css-1174404953-hDTx3UPK.datasource.com:9200"
    ```
    ii. Create a schema and add data to the schema
    ```
    schema = StructType([StructField("id", IntegerType(), False),
                         StructField("name", StringType(), False)])
    rdd = sparkSession.sparkContext.parallelize([Row(1, "John"), Row(2, "Bob")])
    ```
    iii. Construct a DataFrame
    ```
    dataFrame = sparkSession.createDataFrame(rdd, schema)
    ```
    iv. Save the data to CSS
    ```
    dataFrame.write.format("css").option("resource", resource).option("nodes",
    nodes).mode("Overwrite").save()
    ```

    NOTE

    The value of **mode** can be one of the following:
    - ErrorIfExis: If the data already exists, the system throws an exception.
    - Overwrite: If the data already exists, the original data will be overwritten.
    - Append: If the data already exists, the system saves the new data.
    - Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

    v. Read data from CSS
    ```
    jdbcDF = sparkSession.read.format("css").option("resource", resource).option("nodes",
    nodes).load()
    jdbcDF.show()
    ```
    vi. Operation result
    ```
    +---+----+
    | id|name|
    +---+----+
    |  2| Bob|
    |  1|John|
    +---+----+
    ```
  - Connecting to datasources through SQL APIs

i. Create a table to connect to CSS datasource

```
sparkSession.sql(
  "create table css_table(id long, name string) using css options(
    'nodes'='to-css-1174404953-hDTx3UPK.datasource.com:9200',
    'nodes.wan.only'='true',
    'resource'='/mytest/css')")
```

NOTE

For details about the parameters for creating a CSS datasource connection table, see **Table 6-1**.

ii. Insert data

```
sparkSession.sql("insert into css_table values(3,'tom')")
```

iii. Query data

```
jdbcDF = sparkSession.sql("select * from css_table")
jdbcDF.show()
```

iv. Operation result

```
+---+----+
| id|name|
+---+----+
|  3| tom|
|  2| Bob|
|  1|John|
+---+----+
```

– Submitting a Spark job

i. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

ii. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

NOTE

- When submitting a job, you need to specify a dependency module named **sys.datasource.css**.
- For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
- For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

- Complete example code
  – Connecting to datasources through DataFrame APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql.types import Row, StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-css").getOrCreate()

  # Setting cross-source connection parameters
  resource = "/mytest/css"
```



```
nodes = "to-css-1174404953-hDTx3UPK.datasource.com:9200"

# Setting schema
schema = StructType([StructField("id", IntegerType(), False),
                     StructField("name", StringType(), False)])

# Construction data
rdd = sparkSession.sparkContext.parallelize([Row(1, "John"), Row(2, "Bob")])

# Create a DataFrame from RDD and schema
dataFrame = sparkSession.createDataFrame(rdd, schema)

# Write data to the CSS
dataFrame.write.format("css").option("resource", resource).option("nodes",
nodes).mode("Overwrite").save()

# Read data
jdbcDF = sparkSession.read.format("css").option("resource", resource).option("nodes",
nodes).load()
jdbcDF.show()

# close session
sparkSession.stop()
```

- Connecting to datasources through SQL APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-css").getOrCreate()

  # Create a DLI data table for DLI-associated CSS
  sparkSession.sql(
    "create table css_table(id long, name string) using css options(
      'nodes'='to-css-1174404953-hDTx3UPK.datasource.com:9200',
      'nodes.wan.only'='true',
      'resource'='/mytest/css')")

  # Insert data into the DLI data table
  sparkSession.sql("insert into css_table values(3,'tom')")

  # Read data from DLI data table
  jdbcDF = sparkSession.sql("select * from css_table")
  jdbcDF.show()

  # close session
  sparkSession.stop()
```

CSS Security Cluster

- Preparations
  - The Elasticsearch 6.5.4 provided by CSS provides the security settings. Once the function is enabled, CSS provides identity authentication, authorization, and encryption for users. Before connecting DLI to the CSS security cluster, you need to perform certain preparations.
    i. Select CSS Elasticsearch 6.5.4 or a later cluster version, create a CSS security cluster, and download the security cluster certificate (**CloudSearchService.cer**).
    ii. Create a datasource connection.
    iii. Use the **keytools** tool to generate the **keystore** and **truststore** files.
      1) While generating the required files, the security certificate (**CloudSearchService.cer**) of the security cluster is required. Run

the following commands to generate the files. Other parameters of the **keytools** tool can be set as required.

```
keytool -genkeypair -alias certificatekey -keyalg RSA -keystore transport-keystore.jks
keytool -list -v -keystore transport-keystore.jks
keytool -import -alias certificatekey -file CloudSearchService.cer  -keystore truststore.jks
keytool -list -v -keystore truststore
```

2) Upload the generated **keystore** and **truststore** files to an OBS bucket.

- Development description
  - Code implementation
    i. Dependencies related to **import**

```
from __future__ import print_function
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession
```

    ii. Create a session and set the access keys.

```
sparkSession = SparkSession.builder.appName("datasource-css").getOrCreate()
sparkSession.conf.set("fs.obs.access.key", ak)
sparkSession.conf.set("fs.obs.secret.key", sk)
sparkSession.conf.set("fs.obs.endpoint", enpoint)
sparkSession.conf.set("fs.obs.connecton.ssl.enabled", "false")
```

  - Connecting to datasources through DataFrame APIs
    i. Configure datasource connection

```
resource = "/mytest/css";
nodes = "to-css-1174404953-hDTx3UPK.datasource.com:9200"
```

    ii. Create a schema and add data to the schema

```
schema = StructType([StructField("id", IntegerType(), False),
                     StructField("name", StringType(), False)])
rdd = sparkSession.sparkContext.parallelize([Row(1, "John"), Row(2, "Bob")])
```

    iii. Construct a DataFrame

```
dataFrame = sparkSession.createDataFrame(rdd, schema)
```

    iv. Save the data to CSS

```
dataFrame.write.format("css")
  .option("resource", resource)
  .option("nodes", nodes)
  .option("es.net.ssl", "true")
  .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-keystore.jks")
  .option("es.net.ssl.keystore.pass", "***")
  .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
  .option("es.net.ssl.truststore.pass", "***")
  .option("es.net.http.auth.user", "admin")
  .option("es.net.http.auth.pass", "***")
  .mode("Overwrite")
  .save()
```

    NOTE

    The value of **mode** can be one of the following:

    - ErrorIfExis: If the data already exists, the system throws an exception.
    - Overwrite: If the data already exists, the original data will be overwritten.
    - Append: If the data already exists, the system saves the new data.
    - Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

    v. Read data from CSS

```
jdbcDF = sparkSession.read.format("css")
  .option("resource", resource)
```



```
  .option("nodes", nodes)
  .option("es.net.ssl", "true")
  .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-
keystore.jks")
  .option("es.net.ssl.keystore.pass", "***")
  .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
  .option("es.net.ssl.truststore.pass", "***")
  .option("es.net.http.auth.user", "admin")
  .option("es.net.http.auth.pass", "***")
  .load()
jdbcDF.show()
```

vi. Operation result

```
+---+----+
| id|name|
+---+----+
|  2| Bob|
|  1|John|
+---+----+
```

– Submitting a Spark job

i. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

ii. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

NOTE

- When submitting a job, you need to specify a dependency module named **sys.datasource.css**.
- For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
- For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

- Complete example code

– Connecting to datasources through DataFrame APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql.types import Row, StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-css").getOrCreate()
  sparkSession.conf.set("fs.obs.access.key", ak)
  sparkSession.conf.set("fs.obs.secret.key", sk)
  sparkSession.conf.set("fs.obs.endpoint", enpoint)
  sparkSession.conf.set("fs.obs.connecton.ssl.enabled", "false")

  # Setting cross-source connection parameters
  resource = "/mytest/css";
  nodes = "to-css-1174404953-hDTx3UPK.datasource.com:9200"

  # Setting schema
  schema = StructType([StructField("id", IntegerType(), False),
```



```
                    StructField("name", StringType(), False)])

# Construction data
rdd = sparkSession.sparkContext.parallelize([Row(1, "John"), Row(2, "Bob")])

# Create a DataFrame from RDD and schema
dataFrame = sparkSession.createDataFrame(rdd, schema)

# Write data to the CSS
dataFrame.write.format("css")
  .option("resource", resource)
  .option("nodes", nodes)
  .option("es.net.ssl", "true")
  .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-keystore.jks")
  .option("es.net.ssl.keystore.pass", "***")
  .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
  .option("es.net.ssl.truststore.pass", "***")
  .option("es.net.http.auth.user", "admin")
  .option("es.net.http.auth.pass", "***")
  .mode("Overwrite")
  .save()

# Read data
jdbcDF = sparkSession.read.format("css")
  .option("resource", resource)
  .option("nodes", nodes)
  .option("es.net.ssl", "true")
  .option("es.net.ssl.keystore.location", "obs://AK:SK@bucket name/path/transport-keystore.jks")
  .option("es.net.ssl.keystore.pass", "***")
  .option("es.net.ssl.truststore.location", "obs://AK:SK@bucket name/path/truststore.jks")
  .option("es.net.ssl.truststore.pass", "***")
  .option("es.net.http.auth.user", "admin")
  .option("es.net.http.auth.pass", "***")
  .load()
jdbcDF.show()

# close session
sparkSession.stop()
```

6.3 Connecting to DWS

6.3.1 Scala Example Code

Development description

- Prerequisites

  A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Construct dependency information and create a Spark session.

  a. Import dependencies

  Involved Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

  Dependencies related to **import**

```
import java.util.Properties
import org.apache.spark.sql.{Row,SparkSession}
import org.apache.spark.sql.SaveMode
```


b. Create a session

```
val sparkSession = SparkSession.builder().getOrCreate()
```

- Connecting to datasources through SQL APIs

a. Create a table to connect to DWS datasource

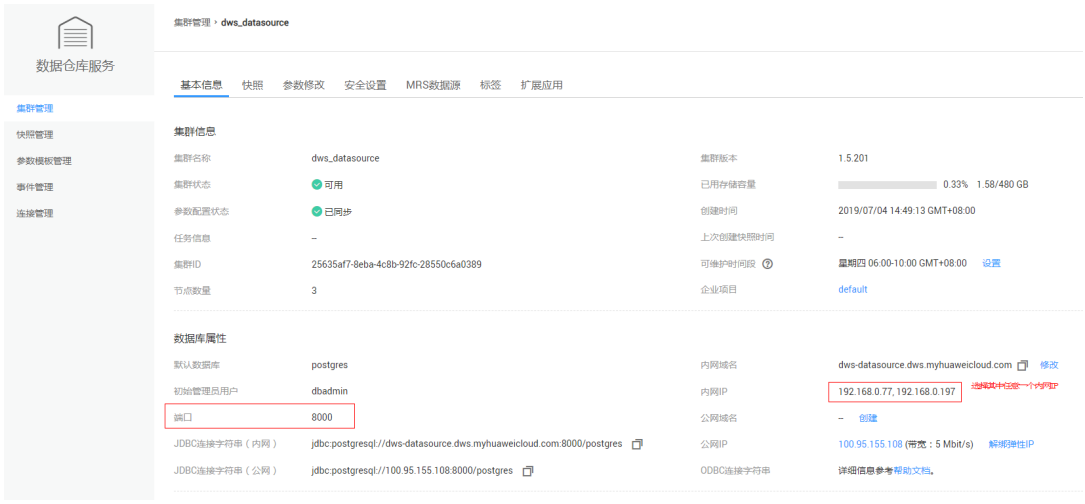
```
sparkSession.sql(
  "CREATE TABLE IF NOT EXISTS dli_to_dws USING JDBC OPTIONS (
    'url'='jdbc:postgresql://to-dws-1174404209-cA37siB6.datasource.com:8000/postgres',
    'dbtable'='customer',
    'user'='dbadmin',
    'password'='######')")
```

Table 6-2 Parameters for creating a table

Parameter	Description
url	To obtain a DWS IP address, you need to create a datasource connection first. Refer to the *Data Lake Insight User Guide* for more information. After a basic datasource connection is created, the returned IP address is used. After an enhanced datasource connection is created, you can use the JDBC connection string (intranet) provided by DWS or the intranet IP address and port number to connect to DWS. The format is ***protocol header.//internal IP address:internal network port number/database name***, for example: **jdbc:postgresql://192.168.0.77:8000/postgres**. For details about how to obtain the value, see *DWS cluster information*. **NOTE** The DWS IP address is in the following format: ***protocol header.//IP address:port number/database name*** Example: jdbc:postgresql://to-dws-1174405119-ihlUr78j.datasource.com:8000/postgres If you want to connect to a database created in DWS, change **postgres** to the corresponding database name in this connection.
user	Username of the DWS data warehouse.
password	Password of the DWS data warehouse.
dbtable	Tables in the PostgreSQL database.
partitionColumn	This parameter is used to set the numeric field used concurrently when data is read. **NOTE** • The **partitionColumn**, **lowerBound**, **upperBound**, and **numPartitions** parameters must be set at the same time. • To improve the concurrent read performance, you are advised to use auto-increment columns.

Parameter	Description
lowerBound	Minimum value of a column specified by **partitionColumn**. The value is contained in the returned result.
upperBound	Maximum value of a column specified by **partitionColumn**. The value is not contained in the returned result.
numPartitions	Number of concurrent read operations. **NOTE** When data is read, **lowerBound** and **upperBound** are evenly allocated to each task to obtain data. Example: 'partitionColumn'='id', 'lowerBound'='0', 'upperBound'='100', 'numPartitions'='2' Two concurrent tasks are started in DLI. The execution ID of one task is greater than or equal to **0** and the ID is less than **50**, and the execution ID of the other task is greater than or equal to **50** and the ID is less than **100**.
fetchsize	Number of data records obtained in each batch during data reading. The default value is **1000**. If this parameter is set to a large value, the performance is good but more memory is occupied. If this parameter is set to a large value, memory overflow may occur.
batchsize	Number of data records written in each batch. The default value is **1000**. If this parameter is set to a large value, the performance is good but more memory is occupied. If this parameter is set to a large value, memory overflow may occur.
truncate	Indicates whether to clear the table without deleting the original table when **overwrite** is executed. The options are as follows: ● true ● false The default value is **false**, indicating that the original table is deleted and then a new table is created when the **overwrite** operation is performed.
isolationLevel	Transaction isolation level. The options are as follows: ● NONE ● READ_UNCOMMITTED ● READ_COMMITTED ● REPEATABLE_READ ● SERIALIZABLE The default value is **READ_UNCOMMITTED**.

Figure 6-1 DWS cluster information



b. Insert data

```
sparkSession.sql("insert into dli_to_dws values(1, 'John',24),(2, 'Bob',32)")
```

c. Query data

```
val dataFrame = sparkSession.sql("select * from dli_to_dws")
dataFrame.show()
```

Before data is inserted:

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  4|  kobe| 24|
|  1|   tom| 18|
|  2|  ammy| 18|
|  5|jordan| 22|
|  7|   chm| 13|
|  6|    qz| 13|
|  3|  mark| 20|
+---+------+---+
```

After data is inserted:

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  4|  kobe| 24|
|  6|    qz| 13|
|  7|   chm| 13|
|  3|  mark| 20|
|  1|   tom| 18|
|  2|  ammy| 18|
|  5|jordan| 22|
|  1|  John| 24|
|  2|   Bob| 32|
+---+------+---+
```

d. Delete the datasource connection table

```
sparkSession.sql("drop table dli_to_dws")
```

- Connecting to datasources through DataFrame APIs

  a. Configure datasource connection


```
val url = "jdbc:postgresql://to-dws-1174405057-EA1Kgo8H.datasource.com:8000/postgres"
val username = "dbadmin"
val password = "######"
val dbtable = "customer"
```

b. Create a DataFrame, add data, and rename fields

```
var dataFrame_1 = sparkSession.createDataFrame(List((8, "Jack_1", 18)))
val df = dataFrame_1.withColumnRenamed("_1", "id")
                    .withColumnRenamed("_2", "name")
                    .withColumnRenamed("_3", "age")
```

c. Import data to DWS

```
df.write.format("jdbc")
  .option("url", url)
  .option("dbtable", dbtable)
  .option("user", username)
  .option("password", password)
  .mode(SaveMode.Append)
  .save()
```

NOTE

The value of **SaveMode** can be one of the following:

- ErrorIfExis: If the data already exists, the system throws an exception.
- Overwrite: If the data already exists, the original data will be overwritten.
- Append: If the data already exists, the system saves the new data.
- Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

d. Read data from DWS

- Method 1: read.format()

```
val jdbcDF = sparkSession.read.format("jdbc")
                .option("url", url)
                .option("dbtable", dbtable)
                .option("user", username)
                .option("password", password)
                .load()
```

- Method 2: read.jdbc()

```
val properties = new Properties()
 properties.put("user", username)
 properties.put("password", password)
 val jdbcDF2 = sparkSession.read.jdbc(url, dbtable, properties)
```

Before data is inserted:

```
+---+------+---+\n
| id|  name|age|\n
+---+------+---+\n
|  4|  kobe| 24|\n
|  3|  mark| 20|\n
|  7|   chm| 13|\n
|  6|    qz| 13|\n
|  1|   tom| 18|\n
|  2|  ammy| 18|\n
|  5|jordan| 22|\n
+---+------+---+\n
```

After data is inserted:


```
+---+------+---+\n
| id|  name|age|\n
+---+------+---+\n
|  7|   chm| 13|\n
|  1|   tom| 18|\n
|  2|  ammy| 18|\n
|  5|jordan| 22|\n
|  8|Jack_1| 18|\n
|  3|  mark| 20|\n
|  6|    qz| 13|\n
|  4|  kobe| 24|\n
+---+------+---+\n
```

The DataFrame read by using the **read.format()** or **read.jdbc()** method is registered as a temporary table. Then, you can use SQL statements to query data.

```
jdbcDF.registerTempTable("customer_test")
 sparkSession.sql("select * from customer_test where id = 1").show()
```

Query results

```
+---+----+---+
| id|name|age|
+---+----+---+
|  1| tom| 18|
+---+----+---+
```

- DataFrame-related operations

  The data created by the **createDataFrame()** method and the data queried by the **read.format()** method and the **read.jdbc()** method are all DataFrame objects. You can directly query a single record. (In step **d**, the DataFrame data is registered as a temporary table.)

  – where

  Where conditions can be combined with filter expressions such as AND and OR. The DataFrame object after filtering is returned. The following is an example:

  ```
  jdbcDF.where("id = 1 or age <=10").show()
  ```

  ```
  +---+------+---+
  | id|  name|age|
  +---+------+---+
  |  7|   chm| 13|
  |  1|   tom| 18|
  |  2|  ammy| 18|
  |  5|jordan| 22|
  |  6|    qz| 13|
  |  3|  mark| 20|
  +---+------+---+
  ```

  – filter

  The **Filter** condition can be used in the same way as **Where**. After you input the **Filter** criteria expression, the result after the filter is returned. The following is an example:

  ```
  jdbcDF.filter("id = 1 or age <=10").show()
  ```



```
+---+------+---+
| id|  name|age|
+---+------+---+
|  4|  kobe| 24|
|  7|   chm| 13|
|  5|jordan| 22|
|  6|    qz| 13|
|  3|  mark| 20|
+---+------+---+
```

- select

  Select is used to query the DataFrame object of the specified field. Multiple fields can be queried.

  - Example 1:
    jdbcDF.select(**"id"**).show()

    ```
    +---+
    | id|
    +---+
    |  4|
    |  7|
    |  3|
    |  6|
    |  1|
    |  2|
    |  5|
    +---+
    ```

  - Example 2:
    jdbcDF.select(**"id"**, **"name"**).show()

    ```
    +---+------+
    | id|  name|
    +---+------+
    |  4|  kobe|
    |  7|   chm|
    |  6|    qz|
    |  3|  mark|
    |  1|   tom|
    |  2|  ammy|
    |  5|jordan|
    +---+------+
    ```

  - Example 3:
    jdbcDF.select(**"id"**,**"name"**).where(**"id<4"**).show()

    ```
    +---+----+
    | id|name|
    +---+----+
    |  1| tom|
    |  2|ammy|
    |  3|mark|
    +---+----+
    ```


- selectExpr

  selectExpr is used to perform special processing on a field. For example, the **selectExpr** function can be used to change the field name. The following is an example:

  If you want to name the **name** field as **name_test** and add 1 to the value of **age**, run the following statement.

  ```
  jdbcDF.selectExpr("id", "name as name_test", "age+1").show()
  ```

- col

  Col is used to obtain a specified field. Different from **Select**, **Col** can only be used to query the column type and one field can be returned at a time. The following is an example:

  ```
  val idCol = jdbcDF.col("id")
  ```

- drop

  Drop is used to delete a specified field. Specify a field you need to delete (only one field can be deleted at a time), the DataFrame object that does not contain the field is returned. The following is an example:

  ```
  jdbcDF.drop("id").show()
  ```

  ```
  +----+---+
  |name|age|
  +----+---+
  |  qz| 13|
  | chm| 13|
  | tom| 18|
  |ammy| 18|
  +----+---+
  ```

- Submitting a Spark job

  a. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Maven dependency

  ```
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>2.3.2</version>
  </dependency>
  ```

- Connecting to datasources through SQL APIs

  ```
  import java.util.Properties
  import org.apache.spark.sql.SparkSession

  object Test_SQL_DWS {
    def main(args: Array[String]): Unit = {
      // Create a SparkSession session.
      val sparkSession = SparkSession.builder().getOrCreate()
      // Create a data table for DLI-associated DWS
  ```



```
  sparkSession.sql("CREATE TABLE IF NOT EXISTS dli_to_dws USING JDBC OPTIONS (
    'url'='jdbc:postgresql://to-dws-1174405057-EA1Kgo8H.datasource.com:8000/postgres',
    'dbtable'='customer',
    'user'='dbadmin',
    'password'='######')")

  //*****************************SQL model***********************************
  //Insert data into the DLI data table
  sparkSession.sql("insert into dli_to_dws values(1,'John',24),(2,'Bob',32)")

  //Read data from DLI data table
  val dataFrame = sparkSession.sql("select * from dli_to_dws")
  dataFrame.show()

  //drop table
  sparkSession.sql("drop table dli_to_dws")

  sparkSession.close()
 }
}
```

- Connecting to datasources through DataFrame APIs

```
import java.util.Properties
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SaveMode

object Test_SQL_DWS {
 def main(args: Array[String]): Unit = {
  // Create a SparkSession session.
  val sparkSession = SparkSession.builder().getOrCreate()

  //*****************************DataFrame model***********************************
  // Set the connection configuration parameters. Contains url, username, password, dbtable.
  val url = "jdbc:postgresql://to-dws-1174405057-EA1Kgo8H.datasource.com:8000/postgres"
  val username = "dbadmin"
  val password = "######"
  val dbtable = "customer"

  //Create a DataFrame and initialize the DataFrame data.
  var dataFrame_1 = sparkSession.createDataFrame(List((1, "Jack", 18)))

  //Rename the fields set by the createDataFrame() method.
  val df = dataFrame_1.withColumnRenamed("_1", "id")
            .withColumnRenamed("_2", "name")
            .withColumnRenamed("_3", "age")

  //Write data to the dws_table_1 table
  df.write.format("jdbc")
   .option("url", url)
   .option("dbtable", dbtable)
   .option("user", username)
   .option("password", password)
   .mode(SaveMode.Append)
   .save()

  // DataFrame object for data manipulation
  //Filter users with id=1
  var newDF = df.filter("id!=1")
  newDF.show()

  // Filter the id column data
  var newDF_1 = df.drop("id")
  newDF_1.show()

  // Read the data of the customer table in the RDS database
  //Way one: Read data from DWS using read.format()
  val jdbcDF = sparkSession.read.format("jdbc")
            .option("url", url)
            .option("dbtable", dbtable)
            .option("user", username)
```



```
                .option("password", password)
                .option("driver", "org.postgresql.Driver")
                .load()
    //Way two: Read data from DWS using read.jdbc()
    val properties = new Properties()
    properties.put("user", username)
    properties.put("password", password)
    val jdbcDF2 = sparkSession.read.jdbc(url, dbtable, properties)

    /**
     * Register the dateFrame read by read.format() or read.jdbc() as a temporary table, and query the
data
     * using the sql statement.
     */
    jdbcDF.registerTempTable("customer_test")
    val result = sparkSession.sql("select * from customer_test where id = 1")
    result.show()

    sparkSession.close()
  }
}
```

6.3.2 PySpark Example Code

Development description

- Prerequisites

  A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Code implementation

  a. Dependencies related to **import**

  ```
  from __future__ import print_function
  from pyspark.sql.types import StructType, StructField, IntegerType, StringType
  from pyspark.sql import SparkSession
  ```

  b. Create a session

  ```
  sparkSession = SparkSession.builder.appName("datasource-dws").getOrCreate()
  ```

- Connecting to datasources through DataFrame APIs

  a. Configure datasource connection parameters

  ```
  url = "jdbc:postgresql://to-dws-1174404951-W8W4cW8I.datasource.com:8000/postgres"
  dbtable = "customer"
  user = "dbadmin"
  password = "######"
  driver = "org.postgresql.Driver"
  ```

  b. Configure data

  ```
  dataList = sparkSession.sparkContext.parallelize([(1, "Katie", 19)])
  ```

  c. Configure the schema

  ```
  schema = StructType([StructField("id", IntegerType(), False),
                       StructField("name", StringType(), False),
                       StructField("age", IntegerType(), False)])
  ```

  d. Create a DataFrame

  ```
  dataFrame = sparkSession.createDataFrame(dataList, schema)
  ```

  e. Save the data to DWS

  ```
  dataFrame.write \
      .format("jdbc") \
      .option("url", url) \
      .option("dbtable", dbtable) \
      .option("user", user) \
      .option("password", password) \
      .option("driver", driver) \
      .mode("Overwrite") \
      .save()
  ```


NOTE

The value of **mode** can be one of the following:

- ErrorIfExis: If the data already exists, the system throws an exception.
- Overwrite: If the data already exists, the original data will be overwritten.
- Append: If the data already exists, the system saves the new data.
- Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

f. Read data from DWS

```
jdbcDF = sparkSession.read \
    .format("jdbc") \
    .option("url", url) \
    .option("dbtable", dbtable) \
    .option("user", user) \
    .option("password", password) \
    .option("driver", driver) \
    .load()
jdbcDF.show()
```

g. Operation result

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|  1|Katie| 19|
+---+-----+---+
```

- Connecting to datasources through SQL APIs

a. Create a table to connect to DWS datasource

```
sparkSession.sql(
  "CREATE TABLE IF NOT EXISTS dli_to_dws USING JDBC OPTIONS (
    'url'='jdbc:postgresql://to-dws-1174404951-W8W4cW8I.datasource.com:8000/postgres',
    'dbtable'='customer',
    'user'='dbadmin',
    'password'='######',
    'driver'='org.postgresql.Driver')")
```

NOTE

For details about table creation parameters, see **Table 6-2**.

b. Insert data

```
sparkSession.sql("insert into dli_to_dws values(2,'John',24)")
```

c. Query data

```
jdbcDF = sparkSession.sql("select * from dli_to_dws").show()
```

d. Operation result

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|  1|Katie| 19|
|  2| John| 24|
+---+-----+---+
```

- Submitting a Spark Job

a. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake**

Insight User Guide. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

NOTE

- When submitting a job, you need to specify a dependency module named **sys.datasource.dws**.
- For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
- For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Connecting to datasources through DataFrame APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-dws").getOrCreate()

  # Set cross-source connection parameters
  url = "jdbc:postgresql://to-dws-1174404951-W8W4cW8I.datasource.com:8000/postgres"
  dbtable = "customer"
  user = "dbadmin"
  password = "######"
  driver = "org.postgresql.Driver"

  # Create a DataFrame and initialize the DataFrame data.
  dataList = sparkSession.sparkContext.parallelize([(1, "Katie", 19)])

  # Setting schema
  schema = StructType([StructField("id", IntegerType(), False),
                       StructField("name", StringType(), False),
                       StructField("age", IntegerType(), False)])

  # Create a DataFrame from RDD and schema
  dataFrame = sparkSession.createDataFrame(dataList, schema)

  # Write data to the DWS table
  dataFrame.write \
      .format("jdbc") \
      .option("url", url) \
      .option("dbtable", dbtable) \
      .option("user", user) \
      .option("password", password) \
      .option("driver", driver) \
      .mode("Overwrite") \
      .save()

  # Read data
  jdbcDF = sparkSession.read \
      .format("jdbc") \
      .option("url", url) \
      .option("dbtable", dbtable) \
      .option("user", user) \
      .option("password", password) \
      .option("driver", driver) \
      .load()
  jdbcDF.show()
```

```
# close session
sparkSession.stop()
```

- Connecting to datasources through SQL APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-dws").getOrCreate()

  # Createa data table for DLI - associated DWS
  sparkSession.sql(
    "CREATE TABLE IF NOT EXISTS dli_to_dws USING JDBC OPTIONS (
      'url'='jdbc:postgresql://to-dws-1174404951-W8W4cW8I.datasource.com:8000/postgres',
      'dbtable'='customer',
      'user'='dbadmin',
      'password'='######',
      'driver'='org.postgresql.Driver')")

  # Insert data into the DLI data table
  sparkSession.sql("insert into dli_to_dws values(2,'John',24)")

  # Read data from DLI data table
  jdbcDF = sparkSession.sql("select * from dli_to_dws").show()

  # close session
  sparkSession.stop()
```

6.4 Connecting to HBase

6.4.1 Scala Example Code

Development description

The CloudTable HBase and MRS HBase can be connected to DLI as data sources.

- Prerequisites

  A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Construct dependency information and create a Spark session.

  a. Import dependencies

  Involved Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

  Dependencies related to **import**

```
import scala.collection.mutable
import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._
```

  b. Create a session

```
val sparkSession = SparkSession.builder().getOrCreate()
```

  c. Create a table to connect to HBase datasource

```
sparkSession.sql("CREATE TABLE test_hbase('id' STRING, 'location' STRING, 'city' STRING,
'booleanf' BOOLEAN,
```

```
      'shortf' SHORT, 'intf' INT, 'longf' LONG, 'floatf' FLOAT,'doublef' DOUBLE) using hbase
OPTIONS (
   'ZKHost'='cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,
             cloudtable-cf82-zk2-weBkIrjI.cloudtable.com:2181,
             cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181',
   'TableName'='table_DupRowkey1',
   'RowKey'='id:5,location:6,city:7',
'Cols'='booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,longf:CF1.longf,floatf:CF1.floatf,doubl
ef:CF1.doublef')",
      'krb5conf'='$krb5conf',
      'keytab' = '$keytab',
      'principal' = '$principal'
)
```

Table 6-3 Parameters for creating a table

Parameter	Description
ZKHost	ZooKeeper IP address of the HBase cluster. You need to create a datasource connection first. For details, see **Data Lake Insight User Guide**. • To connect to a CloudTable cluster, enter the ZooKeeper IP address (internal network). • To connect to an MRS cluster, enter the IP address of the node where the ZooKeeper is located and the Zookeeper external port number. The format is **ZK_IP1:ZK_PORT1,ZK_IP2:ZK_PORT2**. **NOTE** To connect to an MRS cluster, you can create an enhanced datasource connection and configure host information. For details about operations on the management console, see **Enhanced Datasource Connections** in the *Data Lake Insight User Guide*. For related API reference, see **Creating an Enhanced Datasource Connection**.
RowKey	Specifies the row key field of the table connected to DLI. The single and composite row keys are supported. A single row key can be of the numeric or string type. The length does not need to be specified. The composite row key supports only fixed-length data of the string type. The format is ***attribute name 1:Length*, *attribute name 2:length***.
Cols	Specifies the mapping between the fields in the DLI table and the CloudTable table. In the preceding information, the DLI table field is placed before the colon (:) and the CloudTable table field is placed after the colon (:). The dot (.) is used to separate the column family and column name of the CloudTable table. For example: **DLI table field 1:CloudTable table.CloudTable table field 1**, **DLI table field 2:CloudTable table.CloudTable table field 2**, **DLI table field 3:CLoudTable table.CloudTable table field 3**
krb5conf	OBS path of the **krb5.conf** file. The format is **obs://ak:sk@bucket/path**.

Parameter	Description
keytab	OBS path of the **keytab** file. The format is **obs://ak:sk@bucket/path**.
principal	User principal for Kerberos machine-machine authentication

- Connecting to datasources through SQL APIs
  a. Insert data
  ```
  sparkSession.sql("insert into test_hbase values('12345','abc','guiyang',false,null,3,23,2.3,2.34)")
  ```
  b. Query data
  ```
  sparkSession.sql("select * from test_hbase").show ()
  ```
  Response:
  ```
  +-----+--------+-------+--------+------+----+-----+------+-------+\n
  |   id|location|   city|booleanf|shortf|intf|longf|floatf|doublef|\n
  +-----+--------+-------+--------+------+----+-----+------+-------+\n
  |12345|  huawei|guiyang|   false|  null|   3|   23|   2.3|   2.34|\n
  |abcde|  abcdef|abcdefg|    true|     1|   2|    3|   4.0|    5.0|\n
  |check|  abcdef|abcdefg|    true|     1|   2|    3|   4.0|    5.0|\n
  \n+-----+--------+-------+--------+------+----+-----+------+-------+\n\n
  ```
- Connecting to datasources through DataFrame APIs
  a. Construct a schema
  ```
  val attrId = new StructField("id",StringType)
  val location = new StructField("location",StringType)
  val city = new StructField("city",StringType)
  val booleanf = new StructField("booleanf",BooleanType)
  val shortf = new StructField("shortf",ShortType)
  val intf = new StructField("intf",IntegerType)
  val longf = new StructField("longf",LongType)
  val floatf = new StructField("floatf",FloatType)
  val doublef = new StructField("doublef",DoubleType)
  val attrs = Array(attrId, location,city,booleanf,shortf,intf,longf,floatf,doublef)
  ```
  b. //Construct data based on the schema type.
  ```
  val mutableRow: Seq[Any] = Seq("12345","abc","guiyang",false,null,3,23,2.3,2.34)
  val rddData: RDD[Row] =
  sparkSession.sparkContext.parallelize(Array(Row.fromSeq(mutableRow)), 1)
  ```
  c. Import data to HBase
  ```
  sparkSession.createDataFrame(rddData, new StructType(attrs)).write.insertInto("test_hbase")
  ```
  d. Read data from HBase
  ```
  val map = new mutable.HashMap[String, String]()
  map("TableName") = "table_DupRowkey1"
  map("RowKey") = "id:5,location:6,city:7"
  map("Cols") =
  "booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,longf:CF1.longf,floatf:CF1.floatf,doublef:CF1.
  doublef"
  map("ZKHost")="cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,
                 cloudtable-cf82-zk2-weBkIrjl.cloudtable.com:2181,
                 cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181"
  sparkSession.read.schema(new
  StructType(attrs)).format("hbase").options(map.toMap).load().show()
  ```
  Response:


```
+-----+--------+-------+--------+------+----+-----+------+-------+\n
|   id|location|   city|booleanf|shortf|intf|longf|floatf|doublef|\n
+-----+--------+-------+--------+------+----+-----+------+-------+\n
|12345|  huawei|guiyang|   false|  null|   3|   23|   2.3|   2.34|\n
|abcde|  abcdef|abcdefg|    true|     1|   2|    3|   4.0|    5.0|\n
|check|  abcdef|abcdefg|    true|     1|   2|    3|   4.0|    5.0|\n
\n+-----+--------+-------+--------+------+----+-----+------+-------+\n\n
```

- Submitting a Spark Job

  a. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.hbase**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

- Connecting to datasources through SQL APIs

```
import org.apache.spark.sql.SparkSession

object Test_SparkSql_HBase {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    /**
     * Create an association table for the DLI association Hbase table
     */
    sparkSession.sql("CREATE TABLE test_hbase('id' STRING, 'location' STRING, 'city' STRING, 'booleanf'
BOOLEAN,
        'shortf' SHORT, 'intf' INT, 'longf' LONG, 'floatf' FLOAT,'doublef' DOUBLE) using hbase OPTIONS (
    'ZKHost'='cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,
            cloudtable-cf82-zk2-weBkIrjI.cloudtable.com:2181,
            cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181',
    'TableName'='table_DupRowkey1',
    'RowKey'='id:5,location:6,city:7',
    'Cols'='booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,
       longf:CF1.longf,floatf:CF1.floatf,doublef:CF1.doublef')")

    //**************************SQL model**********************************
    sparkSession.sql("insert into test_hbase values('12345','abc','guiyang',false,null,3,23,2.3,2.34)")
    sparkSession.sql("select * from test_hbase").collect()
```

```
    sparkSession.close()
  }
}
```

- Connecting to datasources through DataFrame APIs

```
import scala.collection.mutable

import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._

object Test_SparkSql_HBase {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    // Create an association table for the DLI association Hbase table
    sparkSession.sql("CREATE TABLE test_hbase('id' STRING, 'location' STRING, 'city' STRING, 'booleanf'
BOOLEAN,
        'shortf' SHORT, 'intf' INT, 'longf' LONG, 'floatf' FLOAT,'doublef' DOUBLE) using hbase OPTIONS (
    'ZKHost'='cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,
            cloudtable-cf82-zk2-weBkIrjI.cloudtable.com:2181,
            cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181',
    'TableName'='table_DupRowkey1',
    'RowKey'='id:5,location:6,city:7',

'Cols'='booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,longf:CF1.longf,floatf:CF1.floatf,doublef:CF1.
doublef')")

    //****************************DataFrame model**********************************
    // Setting schema
    val attrId = new StructField("id",StringType)
    val location = new StructField("location",StringType)
    val city = new StructField("city",StringType)
    val booleanf = new StructField("booleanf",BooleanType)
    val shortf = new StructField("shortf",ShortType)
    val intf = new StructField("intf",IntegerType)
    val longf = new StructField("longf",LongType)
    val floatf = new StructField("floatf",FloatType)
    val doublef = new StructField("doublef",DoubleType)
    val attrs = Array(attrId, location,city,booleanf,shortf,intf,longf,floatf,doublef)

    // Populate data according to the type of schema
    val mutableRow: Seq[Any] = Seq("12345","abc","guiyang",false,null,3,23,2.3,2.34)
    val rddData: RDD[Row] = sparkSession.sparkContext.parallelize(Array(Row.fromSeq(mutableRow)),
1)

    // Import the constructed data into Hbase
    sparkSession.createDataFrame(rddData, new StructType(attrs)).write.insertInto("test_hbase")

    // Read data on Hbase
    val map = new mutable.HashMap[String, String]()
    map("TableName") = "table_DupRowkey1"
    map("RowKey") = "id:5,location:6,city:7"
    map("Cols") =
"booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,longf:CF1.longf,floatf:CF1.floatf,doublef:CF1.doubl
ef"
    map("ZKHost")="cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,
                cloudtable-cf82-zk2-weBkIrjI.cloudtable.com:2181,
                cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181"
    sparkSession.read.schema(new
StructType(attrs)).format("hbase").options(map.toMap).load().collect()

    sparkSession.close()
  }
}
```


6.4.2 PySpark Example Code

Development description

The CloudTable HBase and MRS HBase can be connected to DLI as data sources.

- Prerequisites

  A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Code implementation

  a. Dependencies related to **import**

  ```
  from __future__ import print_function
  from pyspark.sql.types import StructType, StructField, IntegerType, StringType, BooleanType, ShortType, LongType, FloatType, DoubleType
  from pyspark.sql import SparkSession
  ```

  b. Create a session

  ```
  sparkSession = SparkSession.builder.appName("datasource-hbase").getOrCreate()
  ```

  c. Create a table to connect to HBase datasource

  ```
  sparkSession.sql(
    "CREATE TABLE test_hbase(id STRING, location STRING, city STRING, booleanf BOOLEAN, shortf SHORT, intf INT, longf LONG,
       floatf FLOAT, doublef DOUBLE) using hbase OPTIONS (
    'ZKHost' = 'cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,
                cloudtable-cf82-zk2-weBkIrjI.cloudtable.com:2181,
                cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181',
    'TableName' = 'table_DupRowkey1',
    'RowKey' = 'id:5,location:6,city:7',
    'Cols' = 'booleanf:CF1.booleanf, shortf:CF1.shortf, intf:CF1.intf, longf:CF1.longf, floatf:CF1.floatf, doublef:CF1.doublef')")
  ```

  NOTE

  - For details about the **ZKHost**, **RowKey**, and **Cols** parameters, see **Table 6-3**.
  - **TableName**: Name of a table in the CloudTable file. If no table name exists, the system automatically creates a table name.

- Connecting to Datasources Through DataFrame APIs

  a. Construct a schema

  ```
  schema = StructType([StructField("id", StringType()),
                       StructField("location", StringType()),
                       StructField("city", StringType()),
                       StructField("booleanf", BooleanType()),
                       StructField("shortf", ShortType()),
                       StructField("intf", IntegerType()),
                       StructField("longf", LongType()),
                       StructField("floatf", FloatType()),
                       StructField("doublef", DoubleType())])
  ```

  b. Configure data

  ```
  dataList = sparkSession.sparkContext.parallelize([("11111", "beijin", "beijing", False, 4, 3, 23, 2.3, 2.34)])
  ```

  c. Create a DataFrame

  ```
  dataFrame = sparkSession.createDataFrame(dataList, schema)
  ```

  d. Import data to HBase

  ```
  dataFrame.write.insertInto("test_hbase")
  ```

  e. Read data from HBase

  ```
  // Set cross-source connection parameters
  TableName = "table_DupRowkey1"
  RowKey = "id:5,location:6,city:7"
  Cols =
  ```



```
"booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,longf:CF1.longf,floatf:CF1.floatf,doublef:CF1.
doublef"
ZKHost = "cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,cloudtable-cf82-zk2-
weBkIrjI.cloudtable.com:2181,
        cloudtable-cf82-zk1- WY09px9l.cloudtable.com:2181"

// select
jdbcDF = sparkSession.read.schema(schema)\
              .format("hbase")\
              .option("ZKHost",ZKHost)\
              .option("TableName",TableName)\
              .option("RowKey",RowKey)\
              .option("Cols",Cols)\
              .load()
jdbcDF.filter("id = '12333' or id='11111'").show()
```

NOTE

The length of **id**, **location**, and **city** parameter is limited. When inserting data, the data value must be specified based on the length. Otherwise, an encoding format error occurs.

f. Operation result

```
+-----+--------+-------+--------+------+----+-----+------+--------+
|   id|location|   city|booleanf|shortf|intf|longf|floatf|doublef|
+-----+--------+-------+--------+------+----+-----+------+--------+
|11111|  beijin|beijing|   false|     4|   3|   23|   2.3|   2.34|
|12333|  nanjin|nanjing|   false|  null|   3|   23|   2.3|   2.34|
+-----+--------+-------+--------+------+----+-----+------+--------+
```

- Submitting a Spark job

  a. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.hbase**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, BooleanType, ShortType,
LongType, FloatType, DoubleType
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-hbase").getOrCreate()

  # Createa data table for DLI-associated ct
```



```
 sparkSession.sql(
   "CREATE TABLE test_hbase(id STRING, location STRING, city STRING, booleanf BOOLEAN, shortf SHORT,
intf INT, longf LONG,
      floatf FLOAT,doublef DOUBLE) using hbase OPTIONS (
   'ZKHost' = 'cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,
              cloudtable-cf82-zk2-weBkIrjI.cloudtable.com:2181,
              cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181',
   'TableName' = 'table_DupRowkey1',
   'RowKey' = 'id:5,location:6,city:7',
   'Cols' =
'booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,longf:CF1.longf,floatf:CF1.floatf,doublef:CF1.doublef')")

 # Create a DataFrame and initialize the DataFrame data.
 dataList = sparkSession.sparkContext.parallelize([("11111", "beijin", "beijing", False, 4, 3, 23, 2.3, 2.34)])

 # Setting schema
 schema = StructType([StructField("id", StringType()),
                      StructField("location", StringType()),
                      StructField("city", StringType()),
                      StructField("booleanf", BooleanType()),
                      StructField("shortf", ShortType()),
                      StructField("intf", IntegerType()),
                      StructField("longf", LongType()),
                      StructField("floatf", FloatType()),
                      StructField("doublef", DoubleType())])

 # Create a DataFrame from RDD and schema
 dataFrame = sparkSession.createDataFrame(dataList, schema)

 # Write data to the cloudtable-hbase
 dataFrame.write.insertInto("test_hbase")

 # Set cross-source connection parameters
 TableName = "table_DupRowkey1"
 RowKey = "id:5,location:6,city:7"
 Cols =
"booleanf:CF1.booleanf,shortf:CF1.shortf,intf:CF1.intf,longf:CF1.longf,floatf:CF1.floatf,doublef:CF1.doublef"
 ZKHost = "cloudtable-cf82-zk3-pa6HnHpf.cloudtable.com:2181,cloudtable-cf82-zk2-
weBkIrjI.cloudtable.com:2181,
          cloudtable-cf82-zk1-WY09px9l.cloudtable.com:2181"
 # Read data on CloudTable-HBase
 jdbcDF = sparkSession.read.schema(schema)\
                      .format("hbase")\
                      .option("ZKHost", ZKHost)\
                      .option("TableName",TableName)\
                      .option("RowKey", RowKey)\
                      .option("Cols", Cols)\
                      .load()
 jdbcDF.filter("id = '12333' or id='11111'").show()

 # close session
 sparkSession.stop()
```

6.5 Connecting to OpenTSDB

6.5.1 Scala Example Code

Development description

The CloudTable OpenTSDB and MRS OpenTSDB can be connected to DLI as data sources.

- Prerequisites

A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Construct dependency information and create a Spark session.

  a. Import dependencies

  Involved Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

  Dependencies related to **import**

```
import scala.collection.mutable
import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._
```

  b. Create a session

```
val sparkSession = SparkSession.builder().getOrCreate()
```

  c. Create a table to connect to OpenTSDB datasource

```
sparkSession.sql("create table opentsdb_test using opentsdb options(
    'Host'='opentsdb-3xcl8dir15m58z3.cloudtable.com:4242',
        'metric'='ctopentsdb',
    'tags'='city,location')")
```

Table 6-4 Parameters for creating a table

Parameter	Description
host	OpenTSDB IP address. • To connect to the CloudTable OpenTSDB, you need to enter the IP address of the OpenTSDB. For details about how to obtain the IP address, see *CloudTable OpenTSDB IP address*. • You can also access the MRS OpenTSDB. If you have created an enhanced datasource connection, enter the IP address and port number of the node where the OpenTSDB is located. The format is **IP:PORT**. If the OpenTSDB has multiple nodes, enter one of the node IP addresses. For details about how to obtain the IP address, see *MRS cluster OpenTSDB IP address* and *MRS cluster OpenTSDB port number*. If you use a basic datasource connection, enter the connection address returned. For details about operations on the management console, see the *Data Lake Insight User Guide*.
metric	Name of the metric in OpenTSDB corresponding to the DLI table to be created.
tags	Tags corresponding to the metric, which is used for operations such as classification, filtering, and quick search. A maximum of 8 tags, including all **tagk** values under the metric, can be added, which are separated by commas (,).

Figure 6-2 CloudTable OpenTSDB IP address

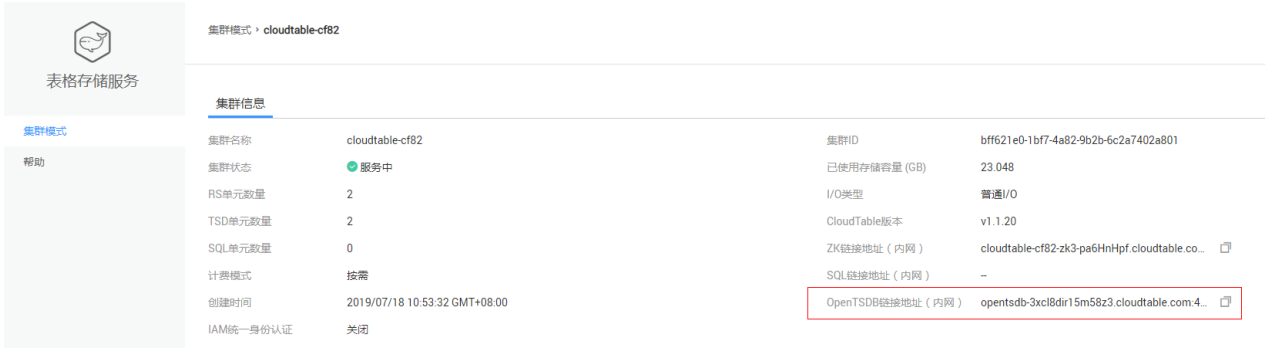


Figure 6-3 MRS cluster OpenTSDB IP address

Figure 6-4 MRS cluster OpenTSDB port number

- Connecting to datasources through SQL APIs

  a. Insert data

  ```
  sparkSession.sql("insert into opentsdb_test values('futian', 'abc', '1970-01-02 18:17:36', 30.0)")
  ```

  b. Query data

  ```
  sparkSession.sql("select * from opentsdb_test").show()
  ```

  Response

  ```
  +-------+--------+-------------------+-----+\n
  |   city|location|          timestamp|value|\n
  +-------+--------+-------------------+-----+\n
  | futian|  huawei|1970-01-02 18:17:36| 30.0|\n
  |beijing|  huawei|1970-01-02 18:17:36| 30.0|\n
  +-------+--------+-------------------+-----+\n\n
  ```


- Connecting to datasources through DataFrame APIs

  a. Construct a schema

  ```
  val attrTag1Location = new StructField("location", StringType)
  val attrTag2Name = new StructField("name", StringType)
  val attrTimestamp = new StructField("timestamp", LongType)
  val attrValue = new StructField("value", DoubleType)
  val attrs = Array(attrTag1Location, attrTag2Name, attrTimestamp, attrValue)
  ```

  b. Construct data based on the schema type.

  ```
  val mutableRow: Seq[Any] = Seq("beijing", "abc", 123456L, 30.0)
  val rddData: RDD[Row] =
  sparkSession.sparkContext.parallelize(Array(Row.fromSeq(mutableRow)), 1)
  ```

  c. Import data to the OpenTSDB

  ```
  sparkSession.createDataFrame(rddData, new StructType(attrs)).write.insertInto("opentsdb_test")
  ```

  d. Read data from the OpenTSDB

  ```
  val map = new mutable.HashMap[String, String]()
  map("metric") = "ctopentsdb"
  map("tags") = "city,location"
  map("Host") = "opentsdb-3xcl8dir15m58z3.cloudtable.com:4242"
  sparkSession.read.format("opentsdb").options(map.toMap).load().show()
  ```

  Response

  ```
  +-------+--------+-------------------+-----+\n
  |   city|location|          timestamp|value|\n
  +-------+--------+-------------------+-----+\n
  | futian|  huawei|1970-01-02 18:17:36| 30.0|\n
  |beijing|  huawei|1970-01-02 18:17:36| 30.0|\n
  +-------+--------+-------------------+-----+\n\n
  ```

- Submitting a Spark Job

  a. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.opentsdb**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

- Connecting to datasources through SQL APIs

```
import org.apache.spark.sql.SparkSession

object Test_OpenTSDB_CT {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    // Create a data table for DLI association OpenTSDB
    sparkSession.sql("create table opentsdb_test using opentsdb options(
    'Host'='opentsdb-3xcl8dir15m58z3.cloudtable.com:4242',
    'metric'='ctopentsdb',
    'tags'='city,location')")

    //*****************************SQL module*********************************
    sparkSession.sql("insert into opentsdb_test values('futian', 'abc', '1970-01-02 18:17:36', 30.0)")
    sparkSession.sql("select * from opentsdb_test").show()

    sparkSession.close()
  }
}
```

- Connecting to datasources through DataFrame APIs

```
import scala.collection.mutable
import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._

object Test_OpenTSDB_CT {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    // Create a data table for DLI association OpenTSDB
    sparkSession.sql("create table opentsdb_test using opentsdb options(
    'Host'='opentsdb-3xcl8dir15m58z3.cloudtable.com:4242',
    'metric'='ctopentsdb',
    'tags'='city,location')")

    //*****************************DataFrame model*********************************
    // Setting schema
    val attrTag1Location = new StructField("location", StringType)
    val attrTag2Name = new StructField("name", StringType)
    val attrTimestamp = new StructField("timestamp", LongType)
    val attrValue = new StructField("value", DoubleType)
    val attrs = Array(attrTag1Location, attrTag2Name, attrTimestamp,attrValue)

    // Populate data according to the type of schema
    val mutableRow: Seq[Any] = Seq("beijing", "abc", 123456L, 30.0)
    val rddData: RDD[Row] = sparkSession.sparkContext.parallelize(Array(Row.fromSeq(mutableRow)),
1)

    //Import the constructed data into OpenTSDB
    sparkSession.createDataFrame(rddData, new StructType(attrs)).write.insertInto("opentsdb_test")

    //Read data on OpenTSDB
    val map = new mutable.HashMap[String, String]()
    map("metric") = "ctopentsdb"
    map("tags") = "city,location"
    map("Host") = "opentsdb-3xcl8dir15m58z3.cloudtable.com:4242"
    sparkSession.read.format("opentsdb").options(map.toMap).load().show()

    sparkSession.close()
  }
}
```


6.5.2 PySpark Example Code

Development description

The CloudTable OpenTSDB and MRS OpenTSDB can be connected to DLI as data sources.

- Prerequisites

  A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Code implementation

  a. Dependencies related to **import**

  ```
  from __future__ import print_function
  from pyspark.sql.types import StructType, StructField, StringType, LongType, DoubleType
  from pyspark.sql import SparkSession
  ```

  b. Create a session

  ```
  sparkSession = SparkSession.builder.appName("datasource-opentsdb").getOrCreate()
  ```

  c. Create a table to connect to OpenTSDB datasource

  ```
  sparkSession.sql("create table opentsdb_test using opentsdb options(
    'Host'='opentsdb-3xcl8dir15m58z3.cloudtable.com:4242',
    'metric'='ct_opentsdb',
    'tags'='city,location')")
  ```

  NOTE

  For details about the **Host**, **Metric**, and **tags** parameters, see **Table 6-4**.

- Connecting to datasources through DataFrame APIs

  a. Construct a schema

  ```
  schema = StructType([StructField("location", StringType()),
                       StructField("name", StringType()),
                       StructField("timestamp", LongType()),
                       StructField("value", DoubleType())])
  ```

  b. Configure data

  ```
  dataList = sparkSession.sparkContext.parallelize([("beijing", "abc", 123456L, 30.0)])
  ```

  c. Create a DataFrame

  ```
  dataFrame = sparkSession.createDataFrame(dataList, schema)
  ```

  d. Import data to the OpenTSDB

  ```
  dataFrame.write.insertInto("opentsdb_test")
  ```

  e. Read data from the OpenTSDB

  ```
  jdbdDF = sparkSession.read
     .format("opentsdb")\
     .option("Host","opentsdb-3xcl8dir15m58z3.cloudtable.com:4242")\
     .option("metric","ctopentsdb")\
     .option("tags","city,location")\
     .load()
  jdbdDF.show()
  ```

  f. Operation result

  ```
  +-------+--------+-------------------+-----+
  |   city|location|          timestamp|value|
  +-------+--------+-------------------+-----+
  | futian|  huawei|1970-01-02 18:17:36| 30.0|
  |nanjing|     tom|2019-08-28 00:00:00| 30.0|
  |beijing|  huawei|1970-01-02 18:17:36| 30.0|
  +-------+--------+-------------------+-----+
  ```


- Submitting a Spark Job
  a. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.
  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.opentsdb**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql.types import StructType, StructField, StringType, LongType, DoubleType
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-opentsdb").getOrCreate()

  # Create a DLI cross-source association opentsdb data table
  sparkSession.sql("create table opentsdb_test using opentsdb options(
    'Host'='opentsdb-3xcl8dir15m58z3.cloudtable.com:4242',
    'metric'='ct_opentsdb',
    'tags'='city,location')")

  # Create a DataFrame and initialize the DataFrame data.
  dataList = sparkSession.sparkContext.parallelize([("beijing", "abc", 123456L, 30.0)])

  # Setting schema
  schema = StructType([StructField("location", StringType()),
                       StructField("name", StringType()),
                       StructField("timestamp", LongType()),
                       StructField("value", DoubleType())])

  # Create a DataFrame from RDD and schema
  dataFrame = sparkSession.createDataFrame(dataList, schema)

  # Set cross-source connection parameters
  metric = "ctopentsdb"
  tags = "city,location"
  Host = "opentsdb-3xcl8dir15m58z3.cloudtable.com:4242"

  # Write data to the cloudtable-opentsdb
  dataFrame.write.insertInto("opentsdb_test")
  # ******* Opentsdb does not currently implement the ctas method to save data, so the save() method
cannot be used.*******
  # dataFrame.write.format("opentsdb").option("Host", Host).option("metric", metric).option("tags",
tags).mode("Overwrite").save()

  # Read data on CloudTable-OpenTSDB
  jdbdDF = sparkSession.read\
     .format("opentsdb")\
```



```
    .option("Host",Host)\
    .option("metric",metric)\
    .option("tags",tags)\
    .load()
jdbdDF.show()

# close session
sparkSession.stop()
```

6.6 Connecting to RDS

6.6.1 Scala Example Code

Development description

- Prerequisites

  A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Construct dependency information and create a Spark session.

  a. Import dependencies

  Involved Maven dependency

  ```
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>2.3.2</version>
  </dependency>
  ```

  Dependencies related to **import**

  ```
  import java.util.Properties
  import org.apache.spark.sql.{Row,SparkSession}
  import org.apache.spark.sql.SaveMode
  ```

  b. Create a session

  ```
  val sparkSession = SparkSession.builder().getOrCreate()
  ```

- Connecting to datasources through SQL APIs

  a. Create a table to connect to RDS datasource

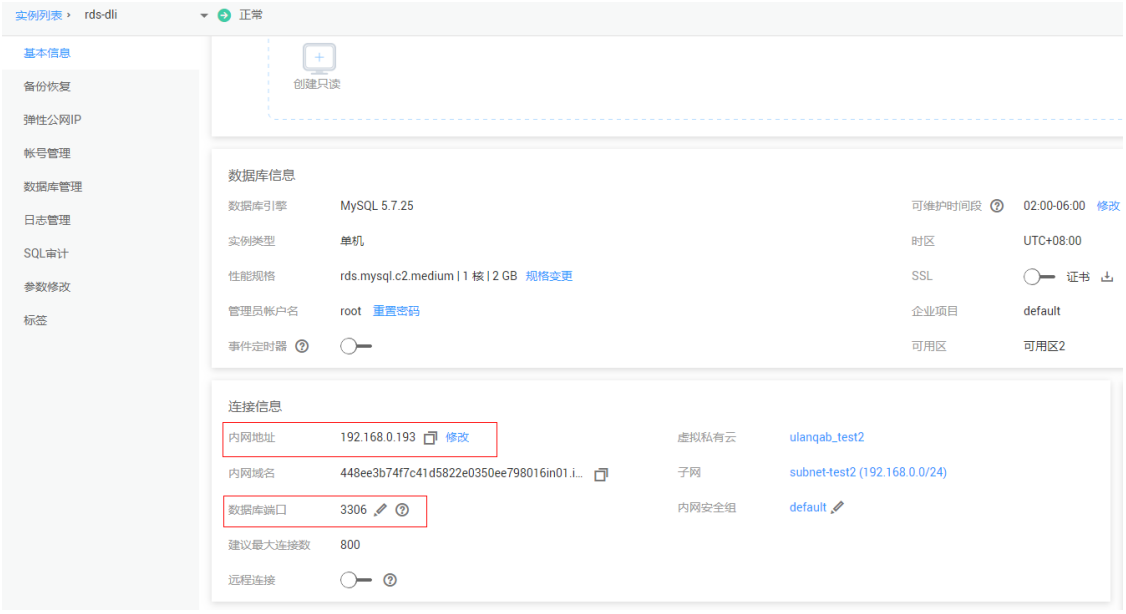
  ```
  sparkSession.sql(
    "CREATE TABLE IF NOT EXISTS dli_to_rds USING JDBC OPTIONS (
       'url'='jdbc:mysql://to-rds-1174404209-cA37siB6.datasource.com:3306',
       'dbtable'='test.customer',
       'user'='root',
       'password'='######',
       'driver'='com.mysql.jdbc.Driver')")
  ```


Table 6-5 Parameters for creating a table

Parameter	Description
url	To obtain an RDS IP address, you need to create a datasource connection first. Refer to the *Data Lake Insight User Guide* for more information. After a basic datasource connection is created, the returned IP address is used. After an enhanced datasource connection is created, use the internal network domain name or internal network address and database port number provided by RDS to connect to DLI. If MySQL is used, the format is ***protocol header.//internal IP address:internal network port number***. If PostgreSQL is used, the format is ***protocol header.//internal IP address:internal network port number/database name***. For example: **jdbc:mysql://192.168.0.193:3306** or **jdbc:postgresql://192.168.0.193:3306/postgres**. For details about how to obtain the value, see *RDS cluster information*. **NOTE** The default format of a datasource connection address is ***protocol header.//IP address:port number***. For example: **jdbc:mysql://to-rds-1174405119-oLRHAGE7.datasource.com:3306** To connect to an RDS PostgreSQL cluster, change the protocol header in the IP address to **jdbc:postgresql** and add **/database name** to the end of the IP address. For example: **jdbc:postgresql://to-rds-1174405119-oLRHAGE7.datasource.com:3306/postgreDB**
user	RDS database username.
password	RDS database password.
dbtable	To connect to a MySQL cluster, enter **database name.table name**. To connect to a PostgreSQL cluster, enter **mode name.table name**.
driver	JDBC driver class name. To connect to a MySQL cluster, enter **com.mysql.jdbc.Driver**. To connect to a PostgreSQL cluster, enter **org.postgresql.Driver**.
partitionColumn	This parameter is used to set the numeric field used concurrently when data is read. **NOTE** • The **partitionColumn**, **lowerBound**, **upperBound**, and **numPartitions** parameters must be set at the same time. • To improve the concurrent read performance, you are advised to use auto-increment columns.

Parameter	Description
lowerBound	Minimum value of a column specified by **partitionColumn**. The value is contained in the returned result.
upperBound	Maximum value of a column specified by **partitionColumn**. The value is not contained in the returned result.
numPartitions	Number of concurrent read operations. **NOTE** When data is read, **lowerBound** and **upperBound** are evenly allocated to each task to obtain data. Example: 'partitionColumn'='id', 'lowerBound'='0', 'upperBound'='100', 'numPartitions'='2' Two concurrent tasks are started in DLI. The execution ID of one task is greater than or equal to **0** and the ID is less than **50**, and the execution ID of the other task is greater than or equal to **50** and the ID is less than **100**.
fetchsize	Number of data records obtained in each batch during data reading. The default value is **1000**. If this parameter is set to a large value, the performance is good but more memory is occupied. If this parameter is set to a large value, memory overflow may occur.
batchsize	Number of data records written in each batch. The default value is **1000**. If this parameter is set to a large value, the performance is good but more memory is occupied. If this parameter is set to a large value, memory overflow may occur.
truncate	Indicates whether to clear the table without deleting the original table when **overwrite** is executed. The options are as follows: ● true ● false The default value is **false**, indicating that the original table is deleted and then a new table is created when the **overwrite** operation is performed.
isolationLevel	Transaction isolation level. The options are as follows: ● NONE ● READ_UNCOMMITTED ● READ_COMMITTED ● REPEATABLE_READ ● SERIALIZABLE The default value is **READ_UNCOMMITTED**.

Figure 6-5 RDS cluster information



b. Insert data

```
sparkSession.sql("insert into dli_to_rds values(1, 'John',24),(2, 'Bob',32)")
```

c. Query data

```
val dataFrame = sparkSession.sql("select * from dli_to_rds")
dataFrame.show()
```

Before data is inserted:

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  4|  kobe| 24|
|  1|   tom| 18|
|  2|  ammy| 18|
|  5|jordan| 22|
|  7|   chm| 13|
|  6|    qz| 13|
|  3|  mark| 20|
+---+------+---+
```

After data is inserted:

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  4|  kobe| 24|
|  6|    qz| 13|
|  7|   chm| 13|
|  3|  mark| 20|
|  1|   tom| 18|
|  2|  ammy| 18|
|  5|jordan| 22|
|  1|  John| 24|
|  2|   Bob| 32|
+---+------+---+
```


d. Delete the datasource connection table

```
sparkSession.sql("drop table dli_to_rds")
```

- Connecting to datasources through DataFrame APIs

a. Configure datasource connection parameters

```
val url = "jdbc:mysql://to-rds-1174405057-EA1Kgo8H.datasource.com:3306"
val username = "root"
val password = "######"
val dbtable = "test.customer"
```

b. Create a DataFrame, add data, and rename fields

```
var dataFrame_1 = sparkSession.createDataFrame(List((8, "Jack_1", 18)))
val df = dataFrame_1.withColumnRenamed("_1", "id")
                    .withColumnRenamed("_2", "name")
                    .withColumnRenamed("_3", "age")
```

c. Import data to RDS

```
df.write.format("jdbc")
  .option("url", url)
  .option("dbtable", dbtable)
  .option("user", username)
  .option("password", password)
  .option("driver", "com.mysql.jdbc.Driver")
  .mode(SaveMode.Append)
  .save()
```

NOTE

The value of **SaveMode** can be one of the following:

- ErrorIfExis: If the data already exists, the system throws an exception.
- Overwrite: If the data already exists, the original data will be overwritten.
- Append: If the data already exists, the system saves the new data.
- Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

d. Read data from RDS

- Method 1: read.format()

```
val jdbcDF = sparkSession.read.format("jdbc")
                .option("url", url)
                .option("dbtable", dbtable)
                .option("user", username)
                .option("password", password)
                .option("driver", "org.postgresql.Driver")
                .load()
```

- Method 2: read.jdbc()

```
val properties = new Properties()
properties.put("user", username)
properties.put("password", password)
val jdbcDF2 = sparkSession.read.jdbc(url, dbtable, properties)
```

Before data is inserted:

```
+---+------+---+\n
| id|  name|age|\n
+---+------+---+\n
|  4|  kobe| 24|\n
|  3|  mark| 20|\n
|  7|   chm| 13|\n
|  6|    qz| 13|\n
|  1|   tom| 18|\n
|  2|  ammy| 18|\n
|  5|jordan| 22|\n
+---+------+---+\n
```


After data is inserted:

```
+---+------+---+\n
| id|  name|age|\n
+---+------+---+\n
|  7|   chm| 13|\n
|  1|   tom| 18|\n
|  2|  ammy| 18|\n
|  5|jordan| 22|\n
|  8|Jack_1| 18|\n
|  3|  mark| 20|\n
|  6|    qz| 13|\n
|  4|  kobe| 24|\n
+---+------+---+\n
```

The DataFrame read by using the **read.format()** or **read.jdbc()** method is registered as a temporary table. Then, you can use SQL statements to query data.

```
jdbcDF.registerTempTable("customer_test")
sparkSession.sql("select * from customer_test where id = 1").show()
```

Query results

```
+---+----+---+
| id|name|age|
+---+----+---+
|  1| tom| 18|
+---+----+---+
```

- DataFrame-related operations

  The data created by the **createDataFrame()** method and the data queried by the **read.format()** method and the **read.jdbc()** method are all DataFrame objects. You can directly query a single record. (In step **d**, the DataFrame data is registered as a temporary table.)

  – where

  Where conditions can be combined with filter expressions such as AND and OR. The DataFrame object after filtering is returned. The following is an example:

  ```
  jdbcDF.where("id = 1 or age <=10").show()
  ```

  ```
  +---+------+---+
  | id|  name|age|
  +---+------+---+
  |  7|   chm| 13|
  |  1|   tom| 18|
  |  2|  ammy| 18|
  |  5|jordan| 22|
  |  6|    qz| 13|
  |  3|  mark| 20|
  +---+------+---+
  ```

  – filter

  The **Filter** condition can be used in the same way as **Where**. After you input the **Filter** criteria expression, the result after the filter is returned. The following is an example:


```
jdbcDF.filter("id = 1 or age <=10").show()
```

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  4|  kobe| 24|
|  7|   chm| 13|
|  5|jordan| 22|
|  6|    qz| 13|
|  3|  mark| 20|
+---+------+---+
```

- select

  Select is used to query the DataFrame object of the specified field. Multiple fields can be queried.

  - Example 1:

    ```
    jdbcDF.select("id").show()
    ```

    ```
    +---+
    | id|
    +---+
    |  4|
    |  7|
    |  3|
    |  6|
    |  1|
    |  2|
    |  5|
    +---+
    ```

  - Example 2:

    ```
    jdbcDF.select("id", "name").show()
    ```

    ```
    +---+------+
    | id|  name|
    +---+------+
    |  4|  kobe|
    |  7|   chm|
    |  6|    qz|
    |  3|  mark|
    |  1|   tom|
    |  2|  ammy|
    |  5|jordan|
    +---+------+
    ```

  - Example 3:

    ```
    jdbcDF.select("id","name").where("id<4").show()
    ```

    ```
    +---+----+
    | id|name|
    +---+----+
    |  1| tom|
    |  2|ammy|
    |  3|mark|
    +---+----+
    ```


- selectExpr

  selectExpr is used to perform special processing on a field. For example, the **selectExpr** function can be used to change the field name. The following is an example:

  If you want to name the **name** field as **name_test** and add 1 to the value of **age**, run the following statement.

  ```
  jdbcDF.selectExpr("id", "name as name_test", "age+1").show()
  ```

- col

  Col is used to obtain a specified field. Different from **Select**, **Col** can only be used to query the column type and one field can be returned at a time. The following is an example:

  ```
  val idCol = jdbcDF.col("id")
  ```

- drop

  Drop is used to delete a specified field. Specify a field you need to delete (only one field can be deleted at a time), the DataFrame object that does not contain the field is returned. The following is an example:

  ```
  jdbcDF.drop("id").show()
  ```

  ```
  +----+---+
  |name|age|
  +----+---+
  |  qz| 13|
  | chm| 13|
  | tom| 18|
  |ammy| 18|
  +----+---+
  ```

- Submitting a Spark Job

  a. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.rds**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Maven dependency

  ```
  <dependency>
    <groupId>org.apache.spark</groupId>
  ```



```
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

- Connecting to datasources through SQL APIs

```
import java.util.Properties
import org.apache.spark.sql.SparkSession

object Test_SQL_RDS {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    // Create a data table for DLI-associated RDS
    sparkSession.sql("CREATE TABLE IF NOT EXISTS dli_to_rds USING JDBC OPTIONS (
   'url'='jdbc:mysql://to-rds-1174404209-cA37siB6.datasource.com:3306,
      'dbtable'='test.customer',
      'user'='root',
      'password'='######',
          'driver'='com.mysql.jdbc.Driver')")

    //*****************************SQL model***********************************
    //Insert data into the DLI data table
    sparkSession.sql("insert into dli_to_rds values(1,'John',24),(2,'Bob',32)")

    //Read data from DLI data table
    val dataFrame = sparkSession.sql("select * from dli_to_rds")
    dataFrame.show()

    //drop table
    sparkSession.sql("drop table dli_to_rds")

    sparkSession.close()
  }
}
```

- Connecting to datasources through DataFrame APIs

```
import java.util.Properties
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SaveMode

object Test_SQL_RDS {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().getOrCreate()

    //*****************************DataFrame model***********************************
    // Set the connection configuration parameters. Contains url, username, password, dbtable.
    val url = "jdbc:mysql://to-rds-1174404209-cA37siB6.datasource.com:3306"
    val username = "root"
    val password = "######"
    val dbtable = "test.customer"

    // Create a DataFrame and initialize the DataFrame data.
    var dataFrame_1 = sparkSession.createDataFrame(List((1, "Jack", 18)))

    // Rename the fields set by the createDataFrame() method.
    val df = dataFrame_1.withColumnRenamed("_1", "id")
                  .withColumnRenamed("_2", "name")
                  .withColumnRenamed("_3", "age")

    // Write data to the rds_table_1 table
    df.write.format("jdbc")
      .option("url", url)
      .option("dbtable", dbtable)
      .option("user", username)
      .option("password", password)
      .option("driver", "com.mysql.jdbc.Driver")
      .mode(SaveMode.Append)
      .save()
```



```
  // DataFrame object for data manipulation
  //Filter users with id=1
  var newDF = df.filter("id!=1")
  newDF.show()

  // Filter the id column data
  var newDF_1 = df.drop("id")
  newDF_1.show()

  // Read the data of the customer table in the RDS database
  // Way one: Read data from RDS using read.format()
  val jdbcDF = sparkSession.read.format("jdbc")
               .option("url", url)
               .option("dbtable", dbtable)
               .option("user", username)
               .option("password", password)
               .option("driver", "com.mysql.jdbc.Driver")
               .load()
  // Way two: Read data from RDS using read.jdbc()
  val properties = new Properties()
  properties.put("user", username)
  properties.put("password", password)
  val jdbcDF2 = sparkSession.read.jdbc(url, dbtable, properties)

  /**
   * Register the dateFrame read by read.format() or read.jdbc() as a temporary table, and query the
data
   * using the sql statement.
   */
  jdbcDF.registerTempTable("customer_test")
  val result = sparkSession.sql("select * from customer_test where id = 1")
  result.show()

  sparkSession.close()
 }
}
```

- DataFrame-related operations

```
 // The where() method uses " and" and "or" for condition filters, returning filtered DataFrame
objects
 jdbcDF.where("id = 1 or age <=10").show()

 // The filter() method is used in the same way as the where() method.
 jdbcDF.filter("id = 1 or age <=10").show()

 // The select() method passes multiple arguments and returns the DataFrame object of the specified
field.
 jdbcDF.select("id").show()
 jdbcDF.select("id", "name").show()
 jdbcDF.select("id","name").where("id<4").show()

 /**
  * The selectExpr() method implements special handling of fields, such as renaming, increasing or
  * decreasing data values.
  */
 jdbcDF.selectExpr("id", "name as name_test", "age+1").show()

 // The col() method gets a specified field each time, and the return type is a Column type.
 val idCol = jdbcDF.col("id")

 /**
  * The drop() method returns a DataFrame object that does not contain deleted fields, and only one
field
  * can be deleted at a time.
  */
 jdbcDF.drop("id").show()
```


6.6.2 PySpark Example Code

Development description

- Prerequisites

  A datasource connection has been created on the DLI management console. For details, see **Data Lake Insight User Guide**.

- Code implementation

  a. Dependencies related to **import**

  ```
  from __future__ import print_function
  from pyspark.sql.types import StructType, StructField, IntegerType, StringType
  from pyspark.sql import SparkSession
  ```

  b. Create a session

  ```
  sparkSession = SparkSession.builder.appName("datasource-rds").getOrCreate()
  ```

- Connecting to datasources through DataFrame APIs

  a. Configure datasource connection parameters

  ```
  url = "jdbc:mysql://to-rds-1174404952-ZgPo1nNC.datasource.com:3306"
  dbtable = "test.customer"
  user = "root"
  password = "######"
  driver = "com.mysql.jdbc.Driver"
  ```

  b. Configure data

  ```
  dataList = sparkSession.sparkContext.parallelize([(123, "Katie", 19)])
  ```

  c. Configure the schema

  ```
  schema = StructType([StructField("id", IntegerType(), False),
                       StructField("name", StringType(), False),
                       StructField("age", IntegerType(), False)])
  ```

  d. Create a DataFrame

  ```
  dataFrame = sparkSession.createDataFrame(dataList, schema)
  ```

  e. Save the data to RDS

  ```
  dataFrame.write \
      .format("jdbc") \
      .option("url", url) \
      .option("dbtable", dbtable) \
      .option("user", user) \
      .option("password", password) \
      .option("driver", driver) \
      .mode("Append") \
      .save()
  ```

  NOTE

  The value of **mode** can be one of the following:

  - ErrorIfExis: If the data already exists, the system throws an exception.
  - Overwrite: If the data already exists, the original data will be overwritten.
  - Append: If the data already exists, the system saves the new data.
  - Ignore: If the data already exists, no operation is required. This is similar to the SQL statement **CREATE TABLE IF NOT EXISTS**.

  f. Read data from RDS

  ```
  jdbcDF = sparkSession.read \
      .format("jdbc") \
      .option("url", url) \
      .option("dbtable", dbtable) \
      .option("user", user) \
      .option("password", password) \
      .option("driver", driver) \
  ```



```
    .load()
jdbcDF.show()
```

g. Operation result

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|123|Katie| 19|
+---+-----+---+
```

- Connecting to datasources through SQL APIs

a. Create a table to connect to RDS datasource

```
sparkSession.sql(
 "CREATE TABLE IF NOT EXISTS dli_to_rds USING JDBC OPTIONS (
   'url'='jdbc:mysql://to-rds-1174404952-ZgPo1nNC.datasource.com:3306',
   'dbtable'='test.customer',
   'user'='root',
   'password'='######',
   'driver'='com.mysql.jdbc.Driver')")
```

NOTE

For details about table creation parameters, see **Table 6-5**.

b. Insert data

```
sparkSession.sql("insert into dli_to_rds values(3,'John',24)")
```

c. Query data

```
jdbcDF_after = sparkSession.sql("select * from dli_to_rds")
jdbcDF_after.show()
```

d. Operation result

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|123|Katie| 19|
|  3| John| 24|
+---+-----+---+
```

- Submitting a Spark Job

a. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

NOTE

- When submitting a job, you need to specify a dependency module named **sys.datasource.rds**.
- For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
- For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Connecting to datasources through DataFrame APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession
if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-rds").getOrCreate()

  # Set cross-source connection parameters.
  url = "jdbc:mysql://to-rds-1174404952-ZgPo1nNC.datasource.com:3306"
  dbtable = "test.customer"
  user = "root"
  password = "######"
  driver = "com.mysql.jdbc.Driver"

  # Create a DataFrame and initialize the DataFrame data.
  dataList = sparkSession.sparkContext.parallelize([(123, "Katie", 19)])

  # Setting schema
  schema = StructType([StructField("id", IntegerType(), False),
                       StructField("name", StringType(), False),
                       StructField("age", IntegerType(), False)])

  # Create a DataFrame from RDD and schema
  dataFrame = sparkSession.createDataFrame(dataList, schema)

  # Write data to the RDS.
  dataFrame.write \
      .format("jdbc") \
      .option("url", url) \
      .option("dbtable", dbtable) \
      .option("user", user) \
      .option("password", password) \
      .option("driver", driver) \
      .mode("Append") \
      .save()

  # Read data
  jdbcDF = sparkSession.read \
      .format("jdbc") \
      .option("url", url) \
      .option("dbtable", dbtable) \
      .option("user", user) \
      .option("password", password) \
      .option("driver", driver) \
      .load()
  jdbcDF.show()

  # close session
  sparkSession.stop()
```

- Connecting to datasources through SQL APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-rds").getOrCreate()

  # Createa data table for DLI - associated RDS
  sparkSession.sql(
    "CREATE TABLE IF NOT EXISTS dli_to_rds USING JDBC OPTIONS (
       'url'='jdbc:mysql://to-rds-1174404952-ZgPo1nNC.datasource.com:3306',
       'dbtable'='test.customer',
       'user'='root',
       'password'='######',
```



```
    'driver'='com.mysql.jdbc.Driver')")

# Insert data into the DLI data table
sparkSession.sql("insert into dli_to_rds values(3,'John',24)")

# Read data from DLI data table
jdbcDF = sparkSession.sql("select * from dli_to_rds")
jdbcDF.show()

# close session
sparkSession.stop()
```

6.7 Connecting to Redis

6.7.1 Scala Example Code

Development description

Redis supports only enhanced datasource connections. Only yearly/monthly queues can be used.

- Prerequisites

  An enhanced datasource connection has been created on the DLI management console and bound to a queue in yearly/monthly packages. For details, see **Data Lake Insight User Guide**.

- Construct dependency information and create a Spark session.

  a. Import dependencies

  Involved Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>3.1.0</version>
</dependency>
<dependency>
  <groupId>com.redislabs</groupId>
  <artifactId>spark-redis</artifactId>
  <version>2.4.0</version>
</dependency>
```

  Dependencies related to **import**

```
import org.apache.spark.sql.{Row, SaveMode, SparkSession}
import org.apache.spark.sql.types._
import com.redislabs.provider.redis._
import scala.reflect.runtime.universe._
import org.apache.spark.{SparkConf, SparkContext}
```

- Connecting to Datasources Through DataFrame APIs

  a. Create a session

```
val sparkSession = SparkSession.builder().appName("datasource_redis").getOrCreate()
```

  b. Construct a schema

```
//method one
var schema = StructType(Seq(StructField("name", StringType, false), StructField("age",
IntegerType, false)))
var rdd = sparkSession.sparkContext.parallelize(Seq(Row("abc",34),Row("Bob",19)))
```



```
var dataFrame = sparkSession.createDataFrame(rdd, schema)
// //method two
// var jdbcDF= sparkSession.createDataFrame(Seq(("Jack",23)))
// val dataFrame = jdbcDF.withColumnRenamed("_1", "name").withColumnRenamed("_2",
"age")
// //method three
// case class Person(name: String, age: Int)
// val dataFrame = sparkSession.createDataFrame(Seq(Person("John", 30), Person("Peter", 45)))
```

NOTE

case class Person(name: String, age: Int) must be written outside the object. For details, see **Connecting to datasources through DataFrame APIs**.

c. Import data to Redis

```
dataFrame .write
  .format("redis")
  .option("host","192.168.4.199")
  .option("port","6379")
  .option("table","person")
  .option("password","******")
  .option("key.column","name")
  .mode(SaveMode.Overwrite)
  .save()
```

Table 6-6 Redis operation parameters

Parameter	Description
host	IP address of the Redis cluster to be connected. To obtain the IP address, log in to the HUAWEI CLOUD official website, search for redis, go to the Distributed Cache Service page, and click Cache Management. Select an IP address (including the port information) based on the IP address required by the host name to copy the data. For details, see **Figure 6-6**.
port	Access port.
password	Specifies the password for the connection. This parameter is optional if no password is required.
table	The key or hash key in Redis. • This parameter is mandatory when Redis data is inserted. • Either this parameter or the **keys.pattern** parameter when Redis data is queried.
keys.pattern	Use a regular expression to match multiple keys or hash keys. This parameter is used only for query. Either this parameter or **table** is used to query Redis data.
key.column	Specifies the key value of a column. This parameter is optional. If a key is specified when data is written, the key must be specified during query. Otherwise, the key will be abnormally loaded during query.

Parameter	Description
partitions.number	Number of concurrent tasks during data reading.
scan.count	Number of data records read in each batch. The default value is **100**. If the CPU usage of the Redis cluster still needs to be improved during data reading, increase the value of this parameter.
iterator.grouping.size	Number of data records inserted in each batch. The default value is **100**. If the CPU usage of the Redis cluster still needs to be improved during the insertion, increase the value of this parameter.
timeout	Timeout interval for connecting to the Redis, in milliseconds. The default value is **2000** (2 seconds).

NOTE

- Save type. The options are **Overwrite**, **Append**, **ErrorIfExis**, and **Ignore**.
- To save nested DataFrames, use **.option("model", "binary")**.
- Specify the data expiration time by **.option("ttl", 1000)**. The unit is second.

Figure 6-6 Obtaining the IP address and port number of Redis

d. Read data from Redis

```
sparkSession.read
 .format("redis")
 .option("host","192.168.4.199")
 .option("port","6379")
 .option("table", "person")
 .option("password","######")
 .option("key.column","name")
 .load()
 .show()
```

Operation result:

```
+------+---+\n
|  name|age|\n
+------+---+\n
|huawei| 34|\n
+------+---+\n
|   Bob| 19|\n
+------+---+\n\n
```


- Connecting to Datasources Using Spark RDD

  a. Create a datasource connection.

```
val sparkContext = new SparkContext(new SparkConf()
  .setAppName("datasource_redis")
  .set("spark.redis.host", "192.168.4.199")
  .set("spark.redis.port", "6379")
  .set("spark.redis.auth", "######")
  .set("spark.driver.allowMultipleContexts","true"))
```

  NOTE

  When the **spark.driver.allowMultipleContexts** is set to **true**, it indicates that when multiple contexts are started, only the current context is used to prevent context invoking conflicts.

  b. Insert data

     i. Saving the string

```
val stringRedisData:RDD[(String,String)] = sparkContext.parallelize(Seq[(String,String)]
(("high","111"), ("together","333")))
sparkContext.toRedisKV(stringRedisData)
```

     ii. Saving the hash

```
val hashRedisData:RDD[(String,String)] = sparkContext.parallelize(Seq[(String,String)]
(("saprk","123"), ("data","222")))
sparkContext.toRedisHASH(hashRedisData, "hashRDD")
```

     iii. Saving the list

```
val data = List(("school","112"), ("tom","333"))
val listRedisData:RDD[String] = sparkContext.parallelize(Seq[(String)](data.toString()))
sparkContext.toRedisLIST(listRedisData, "listRDD")
```

     iv. Saving the set

```
val setData = Set(("bob","133"),("kity","322"))
val setRedisData:RDD[(String)] = sparkContext.parallelize(Seq[(String)]
(setData.mkString))
sparkContext.toRedisSET(setRedisData, "setRDD")
```

     v. Saving the zset

```
val zsetRedisData:RDD[(String,String)] = sparkContext.parallelize(Seq[(String,String)]
(("whight","234"), ("bobo","343")))
sparkContext.toRedisZSET(zsetRedisData, "zsetRDD")
```

  c. Query data

     i. Query by traversing keys.

```
val keysRDD = sparkContext.fromRedisKeys(Array("high","together", "hashRDD",
"listRDD", "setRDD","zsetRDD"), 6)
keysRDD.getKV().collect().foreach(println)
keysRDD.getHash().collect().foreach(println)
keysRDD.getList().collect().foreach(println)
keysRDD.getSet().collect().foreach(println)
keysRDD.getZSet().collect().foreach(println)
```

     ii. Query by string

```
sparkContext.fromRedisKV(Array( "high","together")).collect().foreach{println}
```

     iii. Query by hash

```
sparkContext.fromRedisHash(Array("hashRDD")).collect().foreach{println}
```

     iv. Query by list

```
sparkContext.fromRedisList(Array("listRDD")).collect().foreach{println}
```

     v. Query by set

```
sparkContext.fromRedisSet(Array("setRDD")).collect().foreach{println}
```

     vi. Query by zset

```
sparkContext.fromRedisZSet(Array("zsetRDD")).collect().foreach{println}
```

- Connect to datasources through SQL APIs

a. Create a table to connect to Redis datasource

```
sparkSession.sql(
  "CREATE TEMPORARY VIEW person (name STRING, age INT) USING org.apache.spark.sql.redis
OPTIONS (
    'host' = '192.168.4.199',
    'port' = '6379',
    'password' = '######',
    table  'person')".stripMargin)
```

b. Insert data

```
sparkSession.sql("INSERT INTO TABLE person VALUES ('John', 30),('Peter', 45)".stripMargin)
```

c. Query data

```
sparkSession.sql("SELECT * FROM person".stripMargin).collect().foreach(println)
```

- Submitting a Spark Job

a. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

NOTE

- When submitting a job, you need to specify a dependency module named **sys.datasource.redis**.
- For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
- For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>3.1.0</version>
</dependency>
<dependency>
  <groupId>com.redislabs</groupId>
  <artifactId>spark-redis</artifactId>
  <version>2.4.0</version>
</dependency>
```

- Connecting to datasources through SQL APIs

```
import org.apache.spark.sql.{SparkSession};

object Test_Redis_SQL {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkSession = SparkSession.builder().appName("datasource_redis").getOrCreate();

    sparkSession.sql(
```



```
    "CREATE TEMPORARY VIEW person (name STRING, age INT) USING org.apache.spark.sql.redis
OPTIONS (
        'host' = '192.168.4.199', 'port' = '6379', 'password' = '******',table 'person')".stripMargin)

   sparkSession.sql("INSERT INTO TABLE person VALUES ('John', 30),('Peter', 45)".stripMargin)

   sparkSession.sql("SELECT * FROM person".stripMargin).collect().foreach(println)

   sparkSession.close()
  }
}
```

- Connecting to datasources through DataFrame APIs

```
import org.apache.spark.sql.{Row, SaveMode, SparkSession}
import org.apache.spark.sql.types._

object Test_Redis_SparkSql {
  def main(args: Array[String]): Unit = {
  // Create a SparkSession session.
  val sparkSession = SparkSession.builder().appName("datasource_redis").getOrCreate()

  // Set cross-source connection parameters.
  val host = "192.168.4.199"
  val port = "6379"
  val table = "person"
  val auth = "######"
  val key_column = "name"

  // ******** setting DataFrame ********
  // method one
  var schema = StructType(Seq(StructField("name", StringType, false),StructField("age", IntegerType,
false)))
  var rdd = sparkSession.sparkContext.parallelize(Seq(Row("huawei",34),Row("Bob",19)))
  var dataFrame = sparkSession.createDataFrame(rdd, schema)

// // method two
// var jdbcDF= sparkSession.createDataFrame(Seq(("Jack",23)))
// val dataFrame = jdbcDF.withColumnRenamed("_1", "name").withColumnRenamed("_2", "age")

// // method three
// val dataFrame = sparkSession.createDataFrame(Seq(Person("John", 30), Person("Peter", 45)))

  // Write data to redis
  dataFrame.write.format("redis").option("host",host).option("port",port).option("table",
table).option("password",auth).mode(SaveMode.Overwrite).save()

  // Read data from redis
  sparkSession.read.format("redis").option("host",host).option("port",port).option("table",
table).option("password",auth).load().show()

  // Close session
  sparkSession.close()
  }
}
// methoe two
// case class Person(name: String, age: Int)
```

- Connecting to datasources using Spark RDD

```
import com.redislabs.provider.redis._
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}

object Test_Redis_RDD {
  def main(args: Array[String]): Unit = {
    // Create a SparkSession session.
    val sparkContext = new SparkContext(new SparkConf()
          .setAppName("datasource_redis")
          .set("spark.redis.host", "192.168.4.199")
          .set("spark.redis.port", "6379")
          .set("spark.redis.auth", "@@@@@@")
```



```
        .set("spark.driver.allowMultipleContexts","true"))

    //****************** Write data to redis **********************
    // Save String type data
    val stringRedisData:RDD[(String,String)] = sparkContext.parallelize(Seq[(String,String)]
(("high","111"), ("together","333")))
    sparkContext.toRedisKV(stringRedisData)

    // Save Hash type data
    val hashRedisData:RDD[(String,String)] = sparkContext.parallelize(Seq[(String,String)]
(("saprk","123"), ("data","222")))
    sparkContext.toRedisHASH(hashRedisData, "hashRDD")

    // Save List type data
    val data = List(("school","112"), ("tom","333"));
    val listRedisData:RDD[String] = sparkContext.parallelize(Seq[(String)](data.toString()))
    sparkContext.toRedisLIST(listRedisData, "listRDD")

    // Save Set type data
    val setData = Set(("bob","133"),("kity","322"))
    val setRedisData:RDD[(String)] = sparkContext.parallelize(Seq[(String)](setData.mkString))
    sparkContext.toRedisSET(setRedisData, "setRDD")

    // Save ZSet type data
    val zsetRedisData:RDD[(String,String)] = sparkContext.parallelize(Seq[(String,String)]
(("whight","234"), ("bobo","343")))
    sparkContext.toRedisZSET(zsetRedisData, "zsetRDD")

    // ***************************** Read data from redis ****************************************
    // Traverse the specified key and get the value
    val keysRDD = sparkContext.fromRedisKeys(Array("high","together", "hashRDD", "listRDD",
"setRDD","zsetRDD"), 6)
    keysRDD.getKV().collect().foreach(println)
    keysRDD.getHash().collect().foreach(println)
    keysRDD.getList().collect().foreach(println)
    keysRDD.getSet().collect().foreach(println)
    keysRDD.getZSet().collect().foreach(println)

    // Read String type data//
    val stringRDD = sparkContext.fromRedisKV("keyPattern *")
    sparkContext.fromRedisKV(Array( "high","together")).collect().foreach{println}

    // Read Hash type data//
    val hashRDD = sparkContext.fromRedisHash("keyPattern *")
    sparkContext.fromRedisHash(Array("hashRDD")).collect().foreach{println}

    // Read List type data//
    val listRDD = sparkContext.fromRedisList("keyPattern *")
    sparkContext.fromRedisList(Array("listRDD")).collect().foreach{println}

    // Read Set type data//
    val setRDD = sparkContext.fromRedisSet("keyPattern *")
    sparkContext.fromRedisSet(Array("setRDD")).collect().foreach{println}

    // Read ZSet type data//
    val zsetRDD = sparkContext.fromRedisZSet("keyPattern *")
    sparkContext.fromRedisZSet(Array("zsetRDD")).collect().foreach{println}

    // close session
    sparkContext.stop()
  }
}
```


6.7.2 PySpark Example Code

Development description

Redis supports only enhanced datasource connections. Only yearly/monthly queues can be used.

- Prerequisites

  An enhanced datasource connection has been created on the DLI management console and bound to a queue in yearly/monthly packages. For details, see **Data Lake Insight User Guide**.

- Connecting to datasources through DataFrame APIs

  a. Import dependencies

  ```
  from __future__ import print_function
  from pyspark.sql.types import StructType, StructField, IntegerType, StringType
  from pyspark.sql import SparkSession
  ```

  b. Create a session

  ```
  sparkSession = SparkSession.builder.appName("datasource-redis").getOrCreate()
  ```

  c. Set connection parameters

  ```
  host = "192.168.4.199"
  port = "6379"
  table = "person"
  auth = "@@@@@@"
  ```

  d. Create a DataFrame

  i. Method 1:

  ```
  dataList = sparkSession.sparkContext.parallelize([(1, "Katie", 19),(2,"Tom",20)])
  schema = StructType([StructField("id", IntegerType(), False),
                       StructField("name", StringType(), False),
                       StructField("age", IntegerType(), False)])
  dataFrame = sparkSession.createDataFrame(dataList, schema)
  ```

  ii. Method 2:

  ```
  jdbcDF = sparkSession.createDataFrame([(3,"Jack", 23)])
  dataFrame = jdbcDF.withColumnRenamed("_1", "id").withColumnRenamed("_2",
  "name").withColumnRenamed("_3", "age")
  ```

  e. Importing data to Redis

  ```
  dataFrame.write
    .format("redis")
    .option("host", host)
    .option("port", port)
    .option("table", table)
    .option("password", auth)
    .mode("Overwrite")
    .save()
  ```

  NOTE

  - Save type. The options are **Overwrite**, **Append**, **ErrorIfExis**, and **Ignore**.
  - To specify a key, use **.option("key.column", "name")**. **name** indicates the column name.
  - To save nested DataFrames, use **.option("model", "binary")**.
  - If you need to specify the data expiration time, use **.option("ttl", 1000)**. The unit is second.

  f. Read data from Redis

  ```
  sparkSession.read.format("redis").option("host", host).option("port", port).option("table",
  table).option("password", auth).load().show()
  ```

  g. Operation result


```
+---+-----+---+\n
| id| name|age|\n
+---+-----+---+\n
|  2|  Tom| 20|\n
|  1|Katie| 19|\n
+---+-----+---+\n\n
```

- Connect to datasources through SQL APIs
  a. Create a table to connect to Redis datasource

```
sparkSession.sql(
 "CREATE TEMPORARY VIEW person (name STRING, age INT) USING org.apache.spark.sql.redis
OPTIONS (
    'host' = '192.168.4.199',
    'port' = '6379',
    'password' = '######',
    table  'person')".stripMargin)
```

  b. Insert data

```
sparkSession.sql("INSERT INTO TABLE person VALUES ('John', 30),('Peter', 45)".stripMargin)
```

  c. Query data

```
sparkSession.sql("SELECT * FROM person".stripMargin).collect().foreach(println)
```

- Submitting a Spark job
  a. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.
  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.redis**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Connecting to datasources through DataFrame APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession
if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-redis").getOrCreate()

  # Set cross-source connection parameters.
  host = "192.168.4.199"
  port = "6379"
  table = "person"
  auth = "######"

  # Create a DataFrame and initialize the DataFrame data.
  # *******   method noe   *********
```



```
 dataList = sparkSession.sparkContext.parallelize([(1, "Katie", 19),(2,"Tom",20)])
 schema = StructType([StructField("id", IntegerType(), False),StructField("name", StringType(),
False),StructField("age", IntegerType(), False)])
 dataFrame_one = sparkSession.createDataFrame(dataList, schema)

 # ****** method two ******
 # jdbcDF = sparkSession.createDataFrame([(3,"Jack", 23)])
 # dataFrame = jdbcDF.withColumnRenamed("_1", "id").withColumnRenamed("_2",
"name").withColumnRenamed("_3", "age")

 # Write data to the redis table
 dataFrame.write.format("redis").option("host", host).option("port", port).option("table",
table).option("password", auth).mode("Overwrite").save()
 # Read data
 sparkSession.read.format("redis").option("host", host).option("port", port).option("table",
table).option("password", auth).load().show()

 # close session
 sparkSession.stop()
```

- Connecting to datasources through SQL APIs

```
# _*_ coding: utf-8 _*_
from __future__ import print_function
from pyspark.sql import SparkSession

if __name__ == "__main__":
 # Create a SparkSession
 sparkSession = SparkSession.builder().appName("datasource_redis").getOrCreate()

 sparkSession.sql("CREATE TEMPORARY VIEW person (name STRING, age INT) USING
org.apache.spark.sql.redis OPTIONS (
   'host' = '192.168.4.199',
   'port' = '6379',
   'password' = '######',
   'table'= 'person')".stripMargin);

 sparkSession.sql("INSERT INTO TABLE person VALUES ('John', 30),('Peter', 45)".stripMargin)

 sparkSession.sql("SELECT * FROM person".stripMargin).collect().foreach(println)

 # close session
 sparkSession.stop()
```

6.7.3 Java Example Code

Development description

Redis supports only enhanced datasource connections. Only yearly/monthly queues can be used.

- Prerequisites

  An enhanced datasource connection has been created on the DLI management console and bound to a queue in yearly/monthly packages. For details, see **Data Lake Insight User Guide**.

- Code implementation

  a. Dependencies related to **import**

  ```
  import org.apache.spark.SparkConf;
  import org.apache.spark.api.java.JavaRDD;
  import org.apache.spark.api.java.JavaSparkContext;
  import org.apache.spark.sql.*;
  import org.apache.spark.sql.types.DataTypes;
  import org.apache.spark.sql.types.StructField;
  import org.apache.spark.sql.types.StructType;
  import java.util.*;
  ```


b. Create a session

```
SparkConf sparkConf = new SparkConf();
sparkConf.setAppName("datasource-redis")
      .set("spark.redis.host", "192.168.4.199")
      .set("spark.redis.port", "6379")
      .set("spark.redis.auth", "******")
      .set("spark.driver.allowMultipleContexts","true");
JavaSparkContext javaSparkContext = new JavaSparkContext(sparkConf);
SQLContext sqlContext = new SQLContext(javaSparkContext);
```

- Connecting to Datasources Through DataFrame APIs

  a. Reads JSON data as DataFrame.

  ```
  JavaRDD<String> javaRDD = javaSparkContext.parallelize(Arrays.asList(
        "{\"id\":\"1\",\"name\":\"zhangsan\",\"age\":\"18\"}",
        "{\"id\":\"2\",\"name\":\"lisi\",\"age\":\"21\"}"));
  Dataset dataFrame = sqlContext.read().json(javaRDD);
  ```

  b. Construct the Redis connection configuration parameters.

  ```
  Map map = new HashMap<String, String>();
  map.put("table","person");
  map.put("key.column","id");
  ```

  c. Save data to Redis

  ```
  dataFrame.write().format("redis").options(map).mode(SaveMode.Overwrite).save();
  ```

  d. Read data from Redis

  ```
  sqlContext.read().format("redis").options(map).load().show();
  ```

  e. Operation result

  ```
  +---+--------+---+\n
  | id|    name|age|\n
  +---+--------+---+\n
  |  1|zhangsan| 18|\n
  +---+--------+---+\n
  |  2|    lisi| 21|\n
  +---+--------+---+\n\n
  ```

- Submitting a Spark job

  a. Upload the Java code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.redis**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

```
public class Test_Redis_DaraFrame {
  public static void main(String[] args) {
    //create a SparkSession session
```



```
  SparkConf sparkConf = new SparkConf();
  sparkConf.setAppName("datasource-redis")
          .set("spark.redis.host", "192.168.4.199")
          .set("spark.redis.port", "6379")
          .set("spark.redis.auth", "******")
          .set("spark.driver.allowMultipleContexts","true");
  JavaSparkContext javaSparkContext = new JavaSparkContext(sparkConf);
  SQLContext sqlContext = new SQLContext(javaSparkContext);

  //Read RDD in JSON format to create DataFrame
  JavaRDD<String> javaRDD = javaSparkContext.parallelize(Arrays.asList(
          "{\"id\":\"1\",\"name\":\"zhangsan\",\"age\":\"18\"}",
          "{\"id\":\"2\",\"name\":\"lisi\",\"age\":\"21\"}"));
  Dataset dataFrame = sqlContext.read().json(javaRDD);

  Map map = new HashMap<String, String>();
  map.put("table","person");
  map.put("key.column","id");
  dataFrame.write().format("redis").options(map).mode(SaveMode.Overwrite).save();
  sqlContext.read().format("redis").options(map).load().show();

 }
}
```

6.8 Connecting to MongoDB

6.8.1 Scala Example Code

Development description

MongoDB can be connected only through enhanced datasource connection. Only yearly/monthly queues can be used.

- Prerequisites

  An enhanced datasource connection has been created on the DLI management console and bound to a queue in yearly/monthly packages. For details, see **Data Lake Insight User Guide**.

- Construct dependency information and create a Spark session.

  a. Import dependencies

  Involved Maven dependency

  ```
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>2.3.2</version>
  </dependency>
  ```

  Dependencies related to **import**

  ```
  import org.apache.spark.sql.SparkSession
  import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
  ```

  Create a session

  ```
  val sparkSession = SparkSession.builder().appName("datasource-mongo").getOrCreate()
  ```

- Connecting to datasources through SQL APIs

  a. Create a table to connect to Mongo datasource

  ```
  sparkSession.sql(
    "create table test_mongo(id string, name string, age int) using mongo options(
      'url' = '192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin',
      'database' = 'test',
      'collection' = 'test',
  ```



```
  'user' = 'rwuser',
  'password' = '######')")
```

Table 6-7 Parameters for creating a table

Parameter	Description
url	• URL format: "IP:PORT[,IP:PORT]/[DATABASE][.COLLECTION][AUTH_PROPERTIES]" Example: "192.168.4.62:8635/test?authSource=admin" • The URL needs to be obtained from the mongo (DDS) connection address.. **NOTE** The obtained MongoDB connection address is in the following format: ***Protocol header*://*Username*:*Password*@*Connection address*:*port number*/*Database name*?authSource=admin** Example: mongodb://rwuser:****@192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin
database	DDS database name. If the database name is specified in the URL, the database name in the URL does not take effect.
collection	Collection name in the DDS. If the collection is specified in the URL, the collection in the URL does not take effect. **NOTE** If a collection already exists in DDS, you do not need to specify schema information when creating a table. DLI automatically generates schema information based on data in the collection.
user	Username for accessing the DDS cluster.
password	Password for accessing the DDS cluster.

Figure 6-7 MongoDB link address

b. Insert data

```
sparkSession.sql("insert into test_mongo values('3', 'zhangsan',23)")
```

c. Query data

```
sparkSession.sql("select * from test_mongo").show()
```

Operation result

```
+---+--------+---+\n
| id|    name|age|\n
+---+--------+---+\n
|  2|     Bob| 32|\n
|  1|    John| 23|\n
|  3|zhangsan| 23|\n
+---+--------+---+\n\n
```

- Connecting to datasources through DataFrame APIs

a. Set connection parameters

```
val url = "192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin"
val user = "rwuser"
val database = "test"
val collection = "test"
val password = "######"
```

b. Construct a schema

```
val schema = StructType(List(StructField("id", StringType), StructField("name", StringType),
StructField("age", IntegerType)))
```

c. Construct a DataFrame

```
val rdd = spark.sparkContext.parallelize(Seq(Row("1", "John", 23), Row("2", "Bob", 32)))
val dataFrame = spark.createDataFrame(rdd, schema)
```

d. Importing data to the MongoDB

```
dataFrame.write .format("mongo")
  .option("url", url)
  .option("database", database)
  .option("collection", collection)
  .option("user", user)
  .option("password", password)
  .mode(SaveMode.Overwrite)
  .save()
```

NOTE

Save type. The options are **Overwrite**, **Append**, **ErrorIfExis**, and **Ignore**.

e. Read data from the MongoDB

```
val jdbcDF = spark.read.format("mongo").schema(schema)
  .option("url", url)
  .option("database", database)
  .option("collection", collection)
  .option("user", user)
  .option("password", password)
  .load()
```

Operation result

```
+---+----+---+\n
| id|name|age|\n
+---+----+---+\n
|  2| Bob| 32|\n
|  1|John| 23|\n
+---+----+---+\n\n
```

- Submitting a Spark job

  a. Generate a JAR package based on the code and upload the package to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.mongo**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Maven dependency

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.3.2</version>
</dependency>
```

- Connecting to datasources through SQL APIs

```
import org.apache.spark.sql.SparkSession

object TestMongoSql {
  def main(args: Array[String]): Unit = {
    val sparkSession = SparkSession.builder().getOrCreate()
    sparkSession.sql(
      "create table test_mongo(id string, name string, age int) using mongo options(
        'url' = '192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin',
        'database' = 'test',
        'collection' = 'test',
        'user' = 'rwuser',
        'password' = '######')")
    sparkSession.sql("insert into test_mongo values('3', 'zhangsan',23)")
```

```
    sparkSession.sql("select * from test_mongo").show()
    sparkSession.close()
  }
}
```

- Connecting to datasources through DataFrame APIs

```
import org.apache.spark.sql.{Row, SaveMode, SparkSession}
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}

object Test_Mongo_SparkSql {
  def main(args: Array[String]): Unit = {
  //  Create a SparkSession session.
  val spark = SparkSession.builder().appName("mongodbTest").getOrCreate()

  // Set the connection configuration parameters.
  val url = "192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin"
  val user = "rwuser"
  val database = "test"
  val collection = "test"
  val password = "######"

  // Setting up the schema
  val schema = StructType(List(StructField("id", StringType), StructField("name", StringType),
StructField("age", IntegerType)))

  // Setting up the DataFrame
  val rdd = spark.sparkContext.parallelize(Seq(Row("1", "John", 23), Row("2", "Bob", 32)))
  val dataFrame = spark.createDataFrame(rdd, schema)


  // Write data to mongo
  dataFrame.write .format("mongo")
    .option("url", url)
    .option("database", database)
    .option("collection", collection)
    .option("user", user)
    .option("password", password)
    .mode(SaveMode.Overwrite)
    .save()

  // Reading data from mongo
  val jdbcDF = spark.read.format("mongo").schema(schema)
    .option("url", url)
    .option("database", database)
    .option("collection", collection)
    .option("user", user)
    .option("password", password)
    .load()
  jdbcDF.show()

  spark.close()
 }
}
```

6.8.2 PySpark Example Code

Development description

MongoDB supports only the enhanced datasource connection. Only yearly/monthly queues can be used.

- Prerequisites

  An enhanced datasource connection has been created on the DLI management console and bound to a queue in yearly/monthly packages. For details, see **Data Lake Insight User Guide**.

- Connecting to datasources through DataFrame APIs

  a. Import dependencies.

  ```
  from __future__ import print_function
  from pyspark.sql.types import StructType, StructField, IntegerType, StringType
  from pyspark.sql import SparkSession
  ```

  b. Create a session.

  ```
  sparkSession = SparkSession.builder.appName("datasource-mongo").getOrCreate()
  ```

  c. Configure connection parameters.

  ```
  url = "192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin"
  user = "rwuser"
  database = "test"
  collection = "test"
  password = "######"
  ```

  d. Create a DataFrame.

  ```
  dataList = sparkSession.sparkContext.parallelize([("1", "Katie", 19),("2","Tom",20)])
  schema = StructType([StructField("id", IntegerType(), False),
                       StructField("name", StringType(), False),
                       StructField("age", IntegerType(), False)])
  dataFrame = sparkSession.createDataFrame(dataList, schema)
  ```

  e. Import data to MongoDB.

  ```
  dataFrame.write.format("mongo")
    .option("url", url)
    .option("user",user)
    .option("password",password)
    .option("database",database)
    .option("collection",collection)
    .mode("Overwrite")
    .save()
  ```

  f. Read data from Mongo.

  ```
  jdbcDF = sparkSession.read
    .format("mongo")
    .option("url", url)
    .option("user",user)
    .option("password",password)
    .option("database",database)
    .option("collection",collection)
    .load()
  jdbcDF.show()
  ```

  g. Operation result:

  ```
  +---+-----+---+\n
  | id| name|age|\n
  +---+-----+---+\n
  |  2|  Tom| 20|\n
  |  1|Katie| 19|\n
  +---+-----+---+\n\n
  ```

- Connecting to datasources through SQL APIs

  a. Create a table to connect to Mongo datasource.

  ```
  sparkSession.sql(
    "CREATE TEMPORARY VIEW person (name STRING, age INT) USING
  org.apache.spark.sql.mongo OPTIONS (
      'host' = '192.168.4.199',
      'port' = '6379',
      'password' = '######',
      table  'person')".stripMargin)
  ```

  b. Insert data.

  ```
  sparkSession.sql("INSERT INTO TABLE person VALUES ('John', 30),('Peter', 45)".stripMargin)
  ```

  c. Query data.

  ```
  sparkSession.sql("SELECT * FROM person".stripMargin).collect().foreach(println)
  ```


- Submitting a Spark job
  a. Upload the Python code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.
  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.mongo**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

- Connecting to datasources through DataFrame APIs

```
from __future__ import print_function
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-mongo").getOrCreate()

  # Create a DataFrame and initialize the DataFrame data.
  dataList = sparkSession.sparkContext.parallelize([("1", "Katie", 19),("2","Tom",20)])

  # Setting schema
  schema = StructType([StructField("id", IntegerType(), False),StructField("name", StringType(), False),
StructField("age", IntegerType(), False)])

  # Create a DataFrame from RDD and schema
  dataFrame = sparkSession.createDataFrame(dataList, schema)

  # Setting connection parameters
  url = "192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin"
  user = "rwuser"
  database = "test"
  collection = "test"
  password = "######"

  # Write data to the mongodb table
  dataFrame.write.format("mongo")
    .option("url", url)
    .option("user",user)
    .option("password",password)
    .option("database",database)
    .option("collection",collection)
    .mode("Overwrite").save()

  # Read data
  jdbcDF = sparkSession.read.format("mongo")
    .option("url", url)
    .option("user",user)
    .option("password",password)
    .option("database",database)
    .option("collection",collection)
```

```
  .load()
jdbcDF.show()

# close session
sparkSession.stop()
```

- Connecting to datasources through SQL APIs

```
from __future__ import print_function
from pyspark.sql import SparkSession

if __name__ == "__main__":
  # Create a SparkSession session.
  sparkSession = SparkSession.builder.appName("datasource-mongo").getOrCreate()

  # Createa data table for DLI - associated mongo
  sparkSession.sql(
    "create table test_mongo(id string, name string, age int) using mongo options(
      'url' = '192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin',
      'database' = 'test',
      'collection' = 'test',
      'user' = 'rwuser',
      'password' = '######')")

  # Insert data into the DLI-table
  sparkSession.sql("insert into test_mongo values('3', 'zhangsan',23)")

  # Read data from DLI-table
  sparkSession.sql("select * from test_mongo").show()

  # close session
  sparkSession.stop()
```

6.8.3 Java Example Code

Development description

MongoDB supports only the enhanced datasource connection. Only yearly/monthly queues can be used.

- Prerequisites

  An enhanced datasource connection has been created on the DLI management console and bound to a queue in yearly/monthly packages. For details, see **Data Lake Insight User Guide**.

- Code implementation

  a. Dependencies related to **import**

  ```
  import org.apache.spark.SparkConf;
  import org.apache.spark.SparkContext;
  import org.apache.spark.api.java.JavaRDD;
  import org.apache.spark.api.java.JavaSparkContext;
  import org.apache.spark.sql.Dataset;
  import org.apache.spark.sql.Row;
  import org.apache.spark.sql.SQLContext;
  import org.apache.spark.sql.SaveMode;
  ```

  b. Create a session

  ```
  SparkContext sparkContext = new SparkContext(new SparkConf().setAppName("datasource-mongo"));
  JavaSparkContext javaSparkContext = new JavaSparkContext(sparkContext);
  SQLContext sqlContext = new SQLContext(javaSparkContext);
  ```

- Connecting to datasources through DataFrame APIs

  a. Reads JSON data as DataFrame.

  ```
  JavaRDD<String> javaRDD = javaSparkContext.parallelize(Arrays.asList("{\"id\":\"5\",\"name\":\"zhangsan\",\"age\":\"23\"}"));
  Dataset<Row> dataFrame = sqlContext.read().json(javaRDD);
  ```

b. Set connection parameters

```
String url = "192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin";
String user = "rwuser";
String database = "test";
String collection = "test";
String password = "######";
```

c. Importing data to the MongoDB

```
dataFrame.write().format("mongo")
    .option("url",url)
    .option("database",database)
    .option("collection",collection)
    .option("user",user)
    .option("password",password)
    .mode(SaveMode.Overwrite)
    .save();
```

d. Read data from HBase

```
sqlContext.read().format("mongo")
   .option("url",url)
   .option("database",database)
   .option("collection",collection)
   .option("user",user)
   .option("password",password)
   .load().show();
```

e. Operation result

```
+---+--------+---+\n
| id|    name|age|\n
+---+--------+---+\n
|  5|zhangsan| 23|\n
+---+--------+---+\n\n
```

- Submitting a Spark job

  a. Upload the Java code file to DLI. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Uploading a Resource Package** in the *Data Lake Insight API Reference*.

  b. In the Spark job editor, select the corresponding dependency and execute the Spark job. For details about console operations, see the **Data Lake Insight User Guide**. For API references, see **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

  NOTE

  - When submitting a job, you need to specify a dependency module named **sys.datasource.mongo**.
  - For details about how to submit a job on the console, see **Table 6-Dependency Resources parameter description** in the **Data Lake Insight User Guide**.
  - For details about how to submit a job through an API, see the **modules** parameter in **Table 2-Request parameter description** of **Creating a Batch Processing Job** in the *Data Lake Insight API Reference*.

Complete example code

```
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SQLContext;
```

```
import org.apache.spark.sql.SaveMode;
import java.util.Arrays;

public class TestMongoSparkSql {
  public static void main(String[] args) {
    SparkContext sparkContext = new SparkContext(new SparkConf().setAppName("datasource-mongo"));
    JavaSparkContext javaSparkContext = new JavaSparkContext(sparkContext);
    SQLContext sqlContext = new SQLContext(javaSparkContext);

//    // Read json file as DataFrame, read csv / parquet file, same as json file distribution
//    DataFrame dataFrame = sqlContext.read().format("json").load("filepath");

    // Read RDD in JSON format to create DataFrame
    JavaRDD<String> javaRDD = javaSparkContext.parallelize(Arrays.asList("{\"id\":\"5\",\"name\":\"zhangsan
\",\"age\":\"23\"}"));
    Dataset<Row> dataFrame = sqlContext.read().json(javaRDD);

    String url = "192.168.4.62:8635,192.168.5.134:8635/test?authSource=admin";
    String user = "rwuser";
    String database = "test";
    String collection = "test";
    String password = "######";

    dataFrame.write().format("mongo")
            .option("url",url)
            .option("database",database)
           .option("collection",collection)
            .option("user",user)
            .option("password",password)
            .mode(SaveMode.Overwrite)
            .save();

    sqlContext.read().format("mongo")
            .option("url",url)
            .option("database",database)
            .option("collection",collection)
            .option("user",user)
            .option("password",password)
            .load().show();
    sparkContext.stop();
    javaSparkContext.close();
  }
}
```


7 Accessing a DLI Table Using a Spark Job

Scenario

DLI allows you to access DLI tables created by SQL jobs with Spark job requests. All DLI non-partition tables and DLI partition tables created using the Datasource syntax are supported. DLI partition tables created using Hive statements are not supported. For details about table creation statements, see the **Data Lake Insight SQL Syntax Reference**.

Procedure

1. Grant the permission to access DLI tables

   You need to authorize the Spark permission to access the DLI table or database where the DLI table is located. Currently, only SQL statements can be used for authorization. You can check whether the user has the permission on the **Databases and Tables** page of the management console.

   - Statements for granting permissions:

     ```
     grant DLI_SPARK_APP_READ on databases.${dbName} TO USER ${userName}
     grant DLI_SPARK_APP_READ on databases.${dbName}.tables.${tableName} TO USER ${userName}
     ```

   - Statements for revoking permissions:

     ```
     revoke DLI_SPARK_APP_READ on databases.${dbName} FROM USER ${userName}
     revoke DLI_SPARK_APP_READ on databases.${dbName}.tables.${tableName} FROM USER ${userName}
     ```

   NOTE

   - After the permission is granted or revoked, the permission takes effect after a period of time (not more than 15 minutes) due to token cache reasons.
   - After a database is authorized, all DLI tables in the database can be accessed using Spark. Other tables cannot.
   - Data write permission in DLI tables is not included.
   - Before authorizing a sub-user to use this function, ensure that the sub-user have the **Agent Operator** permission. Otherwise, the sub-user cannot use this function.
   - When you grant permissions to a sub-user for the first time, you need to create a custom policy for the tenant to which the sub-user belongs. Therefore, the account that submits the authorization request must be an authenticated token with the **Security Administrator** permission, for example, a tenant account.

2. Compile a Spark program to access the DLI table.

a. Specify the database and table names using parameters.

```
val map = new mutable.HashMap[String, String]();
map("dli.table.databaseName") = dbName
map("dli.table.tableName") = tableName
```

b. Set format to **dli_table** to access the specified DLI table.

c. To access a CarbonData table, you need to add a configuration item.

```
sparkConf.setAll(Map("spark.sql.extensions" ->
Seq("org.apache.spark.sql.CarbonInternalExtensions",
      "org.apache.spark.sql.DliSparkExtension").mkString(",")))
```

Troubleshooting

- Problem 1
  - Problem description: When **granting the permission to access DLI tables**, the following error message is displayed: "Execute Job failed. Add user: ** to group: ** failed. status code: 400, body: {"error": {"message": "group number of user [***] over 10.", "code": 400, "error_code": null, "error_msg": null, "title": "Bad Request"}}".
  - Cause: The number of user groups to which the grantee belongs exceeds the upper limit 10.
  - Solution: Check the user group to which the grantee belongs, delete some users, or check whether the IAM quota can be increased.
- Problem 2:
  - Problem description: When a Spark task is executed for query, the following error message is displayed: "reason: User class threw exception: java.lang.Exception: Get table meta failed. User do not have DLI_SPARK_APP_READ privilege on db.tableName.".
  - Cause: The user who submits the job does not have the permission to access the table to be accessed. Note: Even a resource tenant or a table owner can access the table only after being authorized.
  - Solution: Perform authorization by referring to **1**.

Complete Scala Example Code

```
val dbName = "test_db"
val tableName = "test_table"
var isCarbonData = "false"
val sparkConf = new SparkConf()
if (isCarbonData.equals("true")) {
  sparkConf.setAll(Map("spark.sql.extensions" -> Seq("org.apache.spark.sql.CarbonInternalExtensions",
    "org.apache.spark.sql.DliSparkExtension").mkString(",")))
}
val sparkSession = SparkSession.builder().appName("ReadDLITableTest").config(sparkConf).getOrCreate()
val map = new mutable.HashMap[String, String]()
map("dli.table.databaseName") = dbName
map("dli.table.tableName") = tableName
val dataFrame = sparkSession.read.format("dli_table").options(map.toMap).load()
dataFrame.collect()
```

When executing the sample code, you need to select a system module (named **sys.datasource.dli-inner-table**) as the dependency in **Advanced Settings** > **Select Dependency Resources** > **Module Name** on the Spark job creation page. For details about how to submit a Spark job, see the **Data Lake Insight User Guide**.

Complete Python Example Code

```
# _*_ coding: utf-8 _*_
from __future__ import print_function

from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, SQLContext

if __name__ == "__main__":
    conf = SparkConf().set('spark.sql.extensions',
"org.apache.spark.sql.CarbonInternalExtensions,org.apache.spark.sql.DliSparkExtension")
    sparkSession =
SparkSession.builder.appName("SparkVisitDliCarbonTest").config(conf=conf).getOrCreate()
    dataFrame = sparkSession.read.format("dli_table").option('dli.table.databaseName', 'test_db').
        option('dli.table.tableName','test_table').load().show()
    sparkSession.close()
```

When executing the sample code, you need to select a system module (named **sys.datasource.dli-inner-table**) as the dependency in **Advanced Settings** > **Select Dependency Resources** > **Module Name** on the Spark job creation page. For details about how to submit a Spark job, see the **Data Lake Insight User Guide**.

8 Using the Spark Job to Access DLI Metadata

Scenario

DLI allows you to access OBS tables created by SQL jobs with Spark job requests. All and OBS non-partition tables and partition tables are supported. For details about how to create an OBS table, see the ***Data Lake Insight SQL Syntax Reference***.

Description

- You can create databases and tables in SQL jobs and read and insert data using SQL or Spark jobs.
- You can create databases and tables in Spark jobs and read and insert data using SQL or Spark jobs.
- You cannot create databases in SQL jobs, create tables in Spark jobs, or read and insert data through SQL or Spark jobs.

Procedure

1. Compile a Spark program to access DLI metadata.
   a. Specify **spark.sql.session.state.builder** and **spark.sql.catalog.class** when creating SparkSession.

```
sparkConf
    .set("spark.sql.session.state.builder",
      "org.apache.spark.sql.hive.UQueryHiveACLSessionStateBuilder")
    .set("spark.sql.catalog.class",
      "org.apache.spark.sql.hive.UQueryHiveACLExternalCatalog")
```

   b. Specify **org.apache.spark.sql.CarbonInternalExtensions** when creating a Spark session.

```
val spark =
   SparkSession.builder
     .config(sparkConf)
     .enableHiveSupport()
     .config("spark.sql.extensions",
   Seq("org.apache.spark.sql.CarbonInternalExtensions",
      "org.apache.spark.sql.DliSparkExtension").mkString(","))
     .appName("SparkTest")
     .getOrCreate()
```


c. Run the SQL statement or perform other operations to access the DLI table.

NOTE

If the SQL statement is a DDL statement, it can be executed only after **collect()** is executed.

2. Submit the Spark application to DLI.

"catalog_name": "dli" needs to be added to the submitted parameter. For details, see the *Data Lake Insight User Guide* or *Data Lake Insight API Reference*. Add **"spark.dli.metaAccess.enable":"true"** to the conf file. Configure **"spark.sql.warehouse.dir": "obs://bucket/warehousepath"** in the conf file if you need to run the DDL.

Example:

```
{
   "queue":"citest",
   "file":"spark-dli-catalog-tester-1.0-SNAPSHOT.jar",
   "className":"DliCatalogTest",
   "args":[
       " select * from db_all_tb_type.tb_part_obs_hive_orc"
   ],
   "conf":{"spark.sql.warehouse.dir": "obs://bucket/warehousepath",
   "spark.dli.metaAccess.enable":"true"},
   "sc_type":"A",
   "executorCores":1,
   "numExecutors":6,
   "executorMemory":"4G",
   "driverCores":2,
   "driverMemory":"7G",
   "catalog_name": "dli"
}
```

Example Code

- Spark example code

```
object DliCatalogTest {
  def main(args:Array[String]): Unit = {
    val sql = args(0)
    val runDdl =
Try(args(1).toBoolean).getOrElse(true)
    System.out.println(s"sql is $sql
runDdl is $runDdl")
    val sparkConf = new SparkConf(true)
    sparkConf
      .set("spark.sql.session.state.builder",
        "org.apache.spark.sql.hive.UQueryHiveACLSessionStateBuilder")
      .set("spark.sql.catalog.class",
        "org.apache.spark.sql.hive.UQueryHiveACLExternalCatalog")
    sparkConf.setAppName("dlicatalogtester")

    val spark = SparkSession.builder
      .config(sparkConf)
      .enableHiveSupport()
      .config("spark.sql.extensions",
Seq("org.apache.spark.sql.CarbonInternalExtensions",

"org.apache.spark.sql.DliSparkExtension").mkString(","))
      .appName("SparkTest")
      .getOrCreate()

    System.out.println("catalog is "
+ spark.sessionState.catalog.toString)
    if (runDdl) {
      val df = spark.sql(sql).collect()
```



```
    } else {
      spark.sql(sql).show()
    }

    spark.close()
  }

}
```

- Python example code

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from __future__ import print_function

import sys

from pyspark.sql import SparkSession

if __name__ == "__main__":
    url = sys.argv[1]
    creatTbl = "CREATE TABLE test_sparkapp.dli_rds USING JDBC OPTIONS ('url'='jdbc:mysql://%s'," \
               "'driver'='com.mysql.jdbc.Driver','dbtable'='test.test'," \
               " 'passwdauth' = 'DatasourceRDSTest_pwd','encryption' = 'true')" % url

    spark = SparkSession \
        .builder \
        .enableHiveSupport()
\   .config("spark.sql.session.state.builder","org.apache.spark.sql.hive.UQueryHiveACLSessionStateBuilde
r")
\
.config("spark.sql.catalog.class", "org.apache.spark.sql.hive.UQueryHiveACLExternalCatalog")
\
.config("spark.sql.extensions",',',join(["org.apache.spark.sql.CarbonInternalExtensions","org.apache.spar
k.sql.DliSparkExtension"]))
\
        .appName("python Spark test
catalog") \
        .getOrCreate()

    spark.sql("CREATE database if not
exists test_sparkapp").collect()
    spark.sql("drop table if exists
test_sparkapp.dli_rds").collect()
    spark.sql(creatTbl).collect()
    spark.sql("select * from
test_sparkapp.dli_rds").show()
    spark.sql("insert into table
test_sparkapp.dli_rds select 12,'aaa'").collect()
    spark.sql("select * from
test_sparkapp.dli_rds").show()
    spark.sql("insert overwrite table
test_sparkapp.dli_rds select 1111,'asasasa'").collect()
    spark.sql("select * from
test_sparkapp.dli_rds").show()
    spark.sql("drop table
test_sparkapp.dli_rds").collect()
    spark.stop()
```


9 TPC-H Usage Guide

TPC-H Sample Data

TPC-H is a test set developed by the Transaction Processing Performance Council (TPC) to simulate decision-making support applications. It is widely used in academia and industry to evaluate the performance of decision-making support technology. This business test has higher requirements on vendors, because it can comprehensively evaluate the overall business computing capability. With universal business significance, is widely used in analysis of bank credit, credit card, telecom operation, tax, as well as tobacco industry decision-making analysis.

The TPC-H benchmark test is developed from TPC-D (a standard specified by TPC in 1994 and used as the test benchmark for decision-making support systems). TPC-H implements a 3NF data warehouse that contains eight basic relationships, with a data volume range from 1 GB to 3 TB. The TPC-H benchmark test includes 22 queries (Q1 to Q22). The main evaluation indicator is the response time of each query (from submission to result return). The unit of the TPC-H benchmark test is the query number per hour (QphH@size). **H** indicates the average number of complex queries per hour. **size** indicates the size of database, which reflects the query processing capability of the system. TPC-H can evaluate key performance parameters that other tests cannot evaluate, because it is modeled based on the actual production and operation environment. In a word, the TPC-H standard by TPC meets the test requirements of data warehouse and motivate vendors and research institutes to stretch the limit of this technology.

In this example, DLI directly queries the TPC-H dataset on OBS. DLI has generated a standard TPC-H-2.18 dataset of 100 MB which is uploaded to the tpch folder on OBS. The read-only permission is granted to you to facilitate query operations.

TPC-H Test and Metrics

TPC-H test is divided into three sub-tests: data loading test, Power test, and Throughput test. Data loading indicates the process of setting up a test database, and the loading test is to test the data loading ability of DBMS. The first test is data loading test that tests data loading time, which is time-consuming. The second test is Power test, also called raw query. After data loading test is complete, the database is in the initial state without any other operation, especially the data in the buffer is not tested. Power test requires that the 22 queries be executed once in sequence and a pair of RF1 and RF2 operations be

executed at the same time. The third test is Throughput test, the core and most complex test, more similar to the actual application environment. With multiple query statement groups and a pair of RF1 and RF2 update flows, Throughput test pose greater pressure on the SUT system than Power test does.

The basic data in the test is related to the execution time (the time of each data loading step, each query execution, and each update execution), based on which you can calculate the data loading time, Power@Size, Throughput@Size, qphH@Size and $/QphH@Size.

Power@Size is the result of the Power test, which is defined as the reciprocal of the geometric average value of the query time and change time. The formula is as follows:

$$\text{TPC-H Power@Size} = \frac{3600 * SF}{\sqrt[24]{\prod_{i=1}^{i=22} QI(i,0) * \prod_{j=1}^{j=2} RI(j,0)}}$$

Size indicates the data size. SF is the scaling factor of data scale. QI (i, 0) indicates the time of the ith query, in seconds. R (I j, 0) is the update time of RFj, in seconds.

Throughput@Size is the Throughput test result, which is defined as the reciprocal of the average value of all query execution time. The formula is as follows:

$$\text{QphH@Size} = \sqrt{Power\ @\ Size * Throughput\ @\ Size}$$

Service Scenario

You can use the built-in TPC-H test suite of DLI to perform interactive query without uploading data.

Advantages of DLI Built-in TPC-H

- You can log in to DLI and get permission to run SQL statements without creating tables or import data.
- The 22 preset TPC-H SQL query templates with rich functions meet the requirements of most business scenarios. You do not need to download TPC-H query statements, which saves your time and energy.
- Data Lake gives you brand-new experience of serverless DLI product within the minimum time.

Precautions

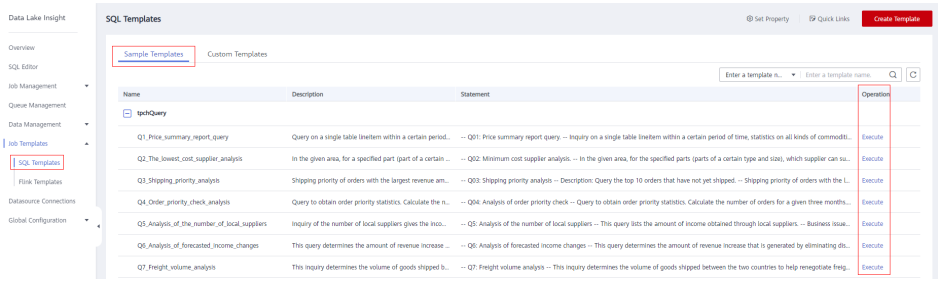
When a sub-account uses the TPC-H test suite, the main account needs to grant the sub-account the OBS access permission and the permission to view the main account table. If the master account has not logged in to DLI, the sub-account needs to have the permissions to create databases and tables in addition to the

preceding permissions. For more information about DLI permissions, see **Data Lake Insight User Guide** .

Operation Description

1. Log in to the HUAWEI CLOUD official website and search for DLI. On the management console, choose **Job Template** > **SQL Template** > **Sample Template** to search for the SQL template created on DLI, which contains 22 standard TPC-H query statements and meets the requirements of most scenarios.

   Figure 9-1 SQL sample template

2. Select a template, click **Execute** in the **Operation** column, and enter the **SQL Editor** page.
3. On the **SQL Editor** page, click **Execute** in the upper right corner of the editing window to run the SQL statement and view the result.

10 Geometry Query

10.1 Basic Concepts

Geospatial Data

Also known as geospatial data, geospatial data is the data or information that identifies the location, form, distribution, and more of an object. It quantitatively describes the objects and phenomena with positioning sense in the real world.

Geospatial data is usually represented in the form of point, line, surface, and body. You can run queries against spatial data to obtain the area, length, and spatial relationship of the target object.

DLI supports geometry data of the following types:

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon

NOTE

The preceding data types are all referred to as specific Geometry types.

Scenario

Geometry queries allow you to collect the number of points of interest in a certain space range, check whether two areas overlap, and calculate the distance between two places.

10.2 Support of Geometry Query Functions

10.2.1 Support of Geometry Query Functions

DLI integrates the geometry query function APIs of GeoMesa. The functions can be classified into the following types:

- **Geometry Constructors**
- **Geometry Accessors**
- **Geometry Cast**
- **Geometry Editors**
- **Geometry Outputs**
- **Spatial Relationships**
- **Geometry Processing**

This section exemplifies how to use the geometry query functions.

10.2.2 Preparing for Geometry Query

1. Create a DLI table named **geotbl**.

   create table geotbl(id String,name String,loadtime date,location String)

2. Insert data into the table. In this step, insert the space data coordinates represented by the WKT string into the **location** field. Sample code is provided as follows:

   insert into geotbl select '1','amy','2015-05-01','POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'

   insert into geotbl select '2','amy','2015-05-01','POLYGON ((60 10,70 60,80 0,60 10))'

   insert into geotbl select '3','amy','2015-05-01','MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))'

   insert into geotbl select '4','amy','2015-05-01','POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10),(20 30, 35 35, 30 20, 20 30))'

   insert into geotbl select '5','amy','2015-05-01','LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)'

   insert into geotbl select '6','amy','2015-05-01','POINT (11 22)'

   insert into geotbl select '7','amy','2015-05-01','MULTILINESTRING ((30 10, 40 40, 20 40, 10 20, 30 10),(30 10, 40 40, 20 40, 10 20, 30 10))'

   insert into geotbl select '8','amy','2015-05-01','MULTIPOINT ((11 22),(10 22))'

The subsequent query examples are illustrated based on this table.

NOTE

This version does not support the functions of directly creating tables with space type fields and importing spatial data in batches.

10.2.3 Geometry Constructors

The geometry constructors can be used to create a Geometry using the well-known binary (WKB) representation, well-known text (WKT) representation, or Geohash.

st_geomFromGeoHash

```
Geometry st_geomFromGeoHash(String geohash, Int prec)
```

This function returns the Geometry of the bounding box corresponding to the Geohash string **geohash** (base-32 encoded) with the precision of prec bits. For more information about Geohash, see **Geohash**.

Example:

- Query command:

  select st_astext(st_geomFromGeoHash('ssf17',25))

- The query result is as follows:

  ```
  POLYGON ((25.4443359375 26.9384765625, 25.4443359375 26.982421875, 25.48828125 26.982421875, 25.48828125 26.9384765625, 25.4443359375 26.9384765625))
  ```

st_box2DFromGeoHash

```
Geometry st_box2DFromGeoHash(String geohash, Int prec)
```

Alias for **st_geomFromGeoHash**. Refer to the example in **st_geomFromGeoHash**.

st_geomFromWKB

```
Geometry st_geomFromWKB(Array[Byte] wkb)
```

This function creates a Geometry from the given WKB.

Example:

- Query command:

  select st_astext((st_geomFromWKB(st_asBinary(st_geomFromText(location))))) from geotbl where id='3'

- The query result is as follows:

  ```
  MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))
  ```

st_geomFromWKT

```
Geometry st_geomFromWKT(String wkt)
```

This function creates a Geometry from the given WKT.

st_geomFromText

```
Geometry st_geomFromText(String wkt)
```

Alias for **st_geomFromWKT**.

Example:

- Query command:

  select st_astext((st_geomFromText(location))) from geotbl where id='1'

- The query result is as follows:

  ```
  POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
  ```

st_geometryFromText(String wkt)

```
Geometry st_geometryFromText(String wkt)
```


Alias for **st_geomFromWKT**.

Example:

- Query command:

  select st_astext((st_geometryFromText(location))) from geotbl where id='5'

- The query result is as follows:

```
LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)
```

st_lineFromText

```
LineString st_lineFromText(String wkt)
```

This function creates a LineString from the given WKT representation.

Example:

- Query command:

  select st_astext((st_lineFromText(location))) from geotbl where id='5'

- The query result is as follows:

```
LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)
```

st_mLineFromText

```
MultiLineString st_mLineFromText(String wkt)
```

This function creates a MultiLineString corresponding to the given WKT representation.

Example:

- Query command:

  select st_astext((st_mLineFromText(location))) from geotbl where id='7'

- The query result is as follows:

```
MULTILINESTRING ((30 10, 40 40, 20 40, 10 20, 30 10), (30 10, 40 40, 20 40, 10 20, 30 10))
```

st_mPointFromText

```
MultiPoint st_mPointFromText(String wkt)
```

This function creates a MultiPoint corresponding to the given WKT representation.

Example:

- Query command:

  select st_astext((st_mPointFromText(location))) from geotbl where id='8'

- The query result is as follows:

```
MULTIPOINT ((11 22), (10 22))
```

st_mPolyFromText

```
MultiPolygon st_mPolyFromText(String wkt)
```

This function creates a MultiPolygon corresponding to the given WKT representation.

Example:

- Query command:

  select st_astext((st_mPolyFromText(location))) from geotbl where id='3'

- The query result is as follows:

```
MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))
```

st_makeBBOX

```
Geometry st_makeBBOX(Double lowerX, Double lowerY, Double upperX, Double upperY)
```

This function creates a Geometry representing a bounding box with the given boundaries.

Example:

- Query command:

  select st_astext((st_makeBBOX(10,20,10,20)))

- The query result is as follows:

```
POINT (10 20)
```

st_makeBox2D

```
Geometry st_makeBox2D(Point lowerLeft, Point upperRight)
```

This function creates a Geometry representing a bounding box defined by the given points.

Example:

- Query command:

  select st_astext(st_makeBox2D(st_castToPoint(st_geomFromWKT('POINT (11 22)')),st_castToPoint(st_geomFromWKT('POINT (10 20)'))))

- The query result is as follows:

```
POLYGON ((10 20, 10 22, 11 22, 11 20, 10 20))
```

st_makePoint

```
Point st_makePoint(Double x, Double y)
```

This function creates a point with x and y coordinates.

Example:

- Query command:

  select st_astext(st_makePoint(1,2))

- The query result is as follows:

```
POINT (1 2)
```

st_makePointM

```
Point st_makePointM(Double x, Double y, Double m)
```

This function creates a point with x, y, and m coordinates.

Example:

- Query command:

  select st_astext(st_makePointM(1,2,2))

- The query result is as follows:
  ```
  POINT (1 2)
  ```

st_makePolygon

```
Polygon st_makePolygon(LineString shell)
```

This function creates a Polygon formed by the given LineString shell, which must be closed.

Example:

- Query command:

  select st_astext(st_makePolygon(st_castToLineString(st_geomFromWKT('LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)'))))
- The query result is as follows:
  ```
  POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
  ```

st_point

```
Point st_point(Double x, Double y)
```

This function creates a point with the given coordinate values. This is an OGC alias for **st_makePoint**.

Example:

- Query command:

  select st_astext(st_point(1,2))
- The query result is as follows:
  ```
  POINT (1 2)
  ```

st_pointFromGeoHash

```
Point st_pointFromGeoHash(String geohash, Int prec)
```

This function creates the point at the geometric center of the bounding box defined by the Geohash string geohash (base-32 encoded) with the precision of prec bits. For more information about Geohash, see **Geohash**.

Example:

- Query command:

  select st_astext(st_pointFromGeoHash('s5zv4',25))
- The query result is as follows:
  ```
  POINT (11.00830078125 21.99462890625)
  ```

st_pointFromText

```
Point st_pointFromText(String wkt)
```

This function creates a point corresponding to the given WKT representation.

Example:

- Query command:

  select st_astext(st_pointFromText('POINT (1 2)'))

- The query result is as follows:
  POINT (1 2)

st_pointFromWKB

```
Point st_pointFromWKB(Array[Byte] wkb)
```

This function creates a point corresponding to the given WKB representation.

Example:

- Query command:

  select st_astext((st_pointFromWKB(st_asBinary(st_geomFromText(location))))) from geotbl where id='6'
- The query result is as follows:
  POINT (11 22)

st_polygon

```
Polygon st_polygon(LineString shell)
```

This function creates a Polygon formed by the given LineString shell, which must be closed.

Example:

- Query command:

  select st_astext(st_polygon(st_castToLineString(st_geomFromWKT('LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)'))))
- The query result is as follows:
  POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))

st_polygonFromText

```
Polygon st_polygonFromText(String wkt)
```

This function creates a polygon corresponding to the given WKT representation.

Example:

- Query command:

  select st_astext(st_polygonFromText('POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'))
- The query result is as follows:
  POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))

10.2.4 Geometry Accessors

st_boundary

```
Geometry st_boundary(Geometry geom)
```

This function returns the boundary, or an empty geometry of appropriate dimension, if geom is empty.

Example:

- Query command:

  select st_astext(st_boundary(st_geomFromWKT(location))) from geotbl where id='1'

- The query result is as follows:

```
LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)
```

st_coordDim

```
Int st_coordDim(Geometry geom)
```

This function returns the number of dimensions of the coordinates of Geometry geom.

Example:

- Query command:

  select st_coordDim(st_boundary(st_geomFromWKT(location))) from geotbl where id='2'

- The query result is as follows:

```
2
```

st_dimension

```
Int st_dimension(Geometry geom)
```

This function returns the inherent number of dimensions of this Geometry object, which must be less than or equal to the coordinate dimension.

Example:

- Query command:

  select st_dimension(st_boundary(st_geomFromWKT(location))) from geotbl where id='2'

- The query result is as follows:

```
1
```

st_envelope

```
Geometry st_envelope(Geometry geom)
```

This function returns a Geometry representing the bounding box of geom.

Example:

- Query command:

  select st_astext(st_envelope(st_geomFromWKT(location))) from geotbl where id='1'

- The query result is as follows:

```
POLYGON ((10 10, 10 40, 40 40, 40 10, 10 10))
```

st_exteriorRing

```
LineString st_exteriorRing(Geometry geom)
```

This function returns a LineString representing the exterior ring of the geometry; returns null if the Geometry is not a Polygon.

Example:

- Query command:

  select st_astext(st_exteriorRing(st_geomFromWKT(location))) from geotbl where id='1'

- The query result is as follows:

```
LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)
```

st_geometryN

```
Geometry st_geometryN(Geometry geom, Geometry n)
```

This function returns the *n*th Geometry (1-based index) of geom if the Geometry is a GeometryCollection, or geom if it is not.

Example:

- Query command:

  select st_astext(st_geometryN(st_geomFromWKT(location),1)) from geotbl where id='1'

- The query result is as follows:

```
POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
```

st_interiorRingN

```
Geometry st_interiorRingN(Geometry geom, Int n)
```

This function returns the *n*th interior LineString ring of the Polygon geom. If the geometry is not a Polygon or the given *n* is out of range, this function returns null.

Example:

- Query command:

  select st_astext(st_interiorRingN(st_geomFromWKT(location),1)) from geotbl where id='4'

- The query result is as follows:

```
LINESTRING (20 30, 35 35, 30 20, 20 30)
```

st_isClosed

```
Boolean st_isClosed(Geometry geom)
```

This function returns true if geom is a LineString or MultiLineString and its start and end points are coincident. This function returns true for all other Geometry types.

Example:

- Query command:

  select st_isClosed(st_geomFromWKT(location)) from geotbl where id='6'

- The query result is as follows:

```
TRUE
```

st_isCollection

```
Boolean st_isCollection(Geometry geom)
```

This function returns true if geom is a GeometryCollection.

Example:

- Query command:

  select st_isCollection(st_geomFromWKT(location)) from geotbl where id='7'

- The query result is as follows:

```
TRUE
```

st_isEmpty

```
Boolean st_isEmpty(Geometry geom)
```

This function returns true if geom is empty.

Example:

- Query command:

  select st_isEmpty(st_geomFromWKT(location)) from geotbl where id='1'

- The query result is as follows:

```
TRUE
```

st_isRing

```
Boolean st_isRing(Geometry geom)
```

This function returns true if geom is a LineString or a MultiLineString and is both closed and simple.

Example:

- Query command:

  select st_isRing(st_geomFromWKT(location)) from geotbl where id='1'

- The query result is as follows:

```
TRUE
```

st_isSimple

```
Boolean st_isSimple(Geometry geom)
```

This function returns true if geom has no anomalous geometric points, such as self intersection or self tangency.

Example:

- Query command:

  select st_isSimple(st_geomFromWKT(location)) from geotbl where id='2'

- The query result is as follows:

```
TRUE
```

st_isValid

```
Boolean st_isValid(Geometry geom)
```

This function returns **true** if the Geometry is topologically valid according to the OGC SFS specification.

Example:

- Query command:

  select st_isValid(st_geomFromWKT(location)) from geotbl where id='3'

- The query result is as follows:
  ```
  TRUE
  ```

st_numGeometries

```
Int st_numGeometries(Geometry geom)
```

If geom is a GeometryCollection, this function returns the number of geometries. Otherwise, it returns **1**.

Example:

- Query command:

  select st_numGeometries(st_geomFromWKT(location)) from geotbl where id='3'

- The query result is as follows:
  ```
  2
  ```

st_numPoints

```
Int st_numPoints(Geometry geom)
```

This function returns the number of vertices in Geometry geom.

Example:

- Query command:

  select st_numPoints(st_geomFromWKT(location)) from geotbl where id='4'

- The query result is as follows:
  ```
  9
  ```

st_pointN

```
Point st_pointN(Geometry geom, Int n)
```

If geom is a LineString, this function returns the *n*th vertex of geom as a point. Negative values are counted backwards from the end of the LineString. This function returns null if geom is not a LineString.

Example:

- Query command:

  select st_astext(st_pointN(st_geomFromWKT(location),2)) from geotbl where id='5'

- The query result is as follows:
  ```
  POINT (40 40)
  ```

st_x

```
Float st_x(Geometry geom)
```

If geom is a point, this function returns the X coordinate of that point.

Example:

- Query command:

  select st_x(st_geomFromWKT(location)) from geotbl where id='6'

- The query result is as follows:
  ```
  11
  ```

st_y

```
Float st_y(Geometry geom)
```

If geom is a point, this function returns the Y coordinate of that point.

Example:

- Query command:

  select st_y(st_geomFromWKT(location)) from geotbl where id='6'
- The query result is as follows:
  ```
  22
  ```

10.2.5 Geometry Cast

st_castToLineString

```
LineString st_castToLineString(Geometry g)
```

This function casts Geometry g to a LineString.

Example:

- Query command:

  select st_astext(st_castToLineString(st_geomFromWKT(location))) from geotbl where id='5'
- The query result is as follows:
  ```
  LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)
  ```

st_castToPoint

```
Point st_castToPoint(Geometry g)
```

This function casts Geometry g to a Point.

Example:

- Query command:

  select st_astext(st_castToPoint(st_geomFromWKT(location))) from geotbl where id='6'
- The query result is as follows:
  ```
  POINT (11 22)
  ```

st_castToPolygon

```
Polygon st_castToPolygon(Geometry g)
```

This function casts Geometry g to a polygon.

Example:

- Query command:

  select st_astext(st_castToPolygon(st_geomFromWKT(location))) from geotbl where id='1'

- The query result is as follows:
  POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))

st_byteArray

```
Array[Byte] st_byteArray(String s)
```

This function encodes strings into an array of bytes using the UTF-8 charset.

Example:

- Query command:

  select st_byteArray(location) from geotbl where id='2'
- The query result is as follows:
  UE9MWUdPTiAoKDYwIDEwLDcwIDYwLDgwIDAsNjAgMTApKQ==

10.2.6 Geometry Editors

st_translate

```
Geometry st_translate(Geometry geom, Double deltaX, Double deltaY)
```

This function returns the Geometry produced when geom is translated by deltaX and deltaY.

Example:

- Query command:

  select st_asText(st_translate(st_geomFromWKT(location),1,2)) from geotbl where id='4'
- The query result is as follows:
  POLYGON ((36 12, 46 47, 16 42, 11 22, 36 12), (21 32, 36 37, 31 22, 21 32))

10.2.7 Geometry Outputs

st_asBinary

```
Array[Byte] st_asBinary(Geometry geom)
```

This function returns Geometry geom in WKB representation.

Example:

- Query command:

  select st_asBinary(st_geomFromWKT(location)) from geotbl where id='1'
- The query result is as follows:
  AAAAAAMAAAABAAAABUA
  +AAAAAAAAQCQAAAAAAABARAAAAAAAAEBEAAAAAAAAQDQAAAAAAABARAAAAAAAAEAkAAAAAAA
  AQDQAAAAAAABAPgAAAAAAAEAkAAAAAAA

st_asGeoJSON

```
String st_asGeoJSON(Geometry geom)
```

This function returns Geometry geom in GeoJSON representation.

Example:

- Query command:

  select st_asGeoJSON(st_geomFromWKT(location)) from geotbl where id='3'

- The query result is as follows:

```
{"type":"MultiPolygon","coordinates":[[[[30,20],[45,40],[10,40],[30,20]]],[[[15,5],[40,10],[10,20],[5,10],[15,5]]]]}
```

st_asLatLonText

```
String st_asLatLonText(Point p)
```

This function returns a String describing the latitude and longitude of Point p in degrees, minutes, and seconds. (This presumes that the units of the coordinates of p are latitude and longitude.)

Example:

- Query command:

  select st_asLatLonText(st_castToPoint(st_geomFromWKT(location))) from geotbl where id='6'

- The query result is as follows:

```
22°0'0.000\"N 11°0'0.000\"E
```

st_asText

```
String st_asText(Geometry geom)
```

This function returns Geometry geom in WKT representation.

st_geoHash

```
String st_geoHash(Geometry geom, Int prec)
```

This function returns the Geohash (in base-32 representation) of an interior point of Geometry geom. **prec** indicates the encoding precision. For more information about Geohash, see **Geohash**.

Example:

- Query command:

  select st_geoHash(st_geomFromWKT(location),25) from geotbl where id='1'

- The query result is as follows:

```
ssf17
```

10.2.8 Spatial Relationships

st_area

```
Double st_area(Geometry g)
```

If Geometry g is areal, this function returns the area of its surface in square units of the coordinate reference system (for example, degrees^2 for EPSG:4326). This function returns 0.0 for non-areal geometries (for example, Points and non-closed LineStrings).

Example:

- Query command:

  select st_area(st_geomFromWKT(location)) from geotbl where id='1'

- The query result is as follows:

```
550.0
```

st_centroid

```
Point st_centroid(Geometry g)
```

This function returns the geometric center of a geometry.

Example:

- Query command:

  select st_asText(st_centroid(st_geomFromWKT(location))) from geotbl where id='4'

- The query result is as follows:

```
POINT (27.40740740740741 28.765432098765434)
```

st_closestPoint

```
Point st_closestPoint(Geometry a, Geometry b)
```

This function returns the Point on a that is closest to b. This is the first point of the shortest line.

Example:

- Query command:

  select st_asText(st_closestPoint((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2')))

- The query result is as follows:

```
POINT (40 40)
```

st_contains

```
Boolean st_contains(Geometry a, Geometry b)
```

This function returns **true** if and only if no points of b lie in the exterior of a, and at least one point of the interior of b lies in the interior of a.

Example:

- Query command:

  select st_contains((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```

st_covers

```
Boolean st_covers(Geometry a, Geometry b)
```

This function returns true if no point in Geometry b is outside Geometry a.

Example:

- Query command:

  select st_covers((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```

st_crosses

```
Boolean st_crosses(Geometry a, Geometry b)
```

This function returns true if the supplied geometries have some, but not all, interior points in common.

Example:

- Query command:

  select st_crosses((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```

st_disjoint

```
Boolean st_disjoint(Geometry a, Geometry b)
```

This function returns true if the geometries do not spatially intersect, for example, they do not share any space together. This function is equivalent to NOT st_intersects(a, b).

Example:

- Query command:

  select st_disjoint((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
TRUE
```

st_distance

```
Double st_distance(Geometry a, Geometry b)
```

This function returns the 2D Cartesian distance between the two geometries in units of the coordinate reference system (for example, degrees for EPSG:4236).

Example:

- Query command:

  select st_distance((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
25.4950975679639
```

st_distanceSphere

```
Double st_distanceSphere(Geometry a, Geometry b)
```

This function approximates the minimum distance between two longitude/latitude geometries assuming a spherical earth.

Example:

- Query command:

  select st_distanceSphere((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
3284010.93490738
```

st_distanceSpheroid

```
Double st_distanceSpheroid(Geometry a, Geometry b)
```

This function returns the minimum distance between two longitude/latitude geometries assuming the WGS84 spheroid.

Example:

- Query command:

  select st_distanceSpheroid((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
3288016.93279476
```

st_equals

```
Boolean st_equals(Geometry a, Geometry b)
```

This function returns true if the given Geometries represent the same logical Geometry. Directionality is ignored.

Example:

- Query command:

  select st_equals((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```

st_intersects

```
Boolean st_intersects(Geometry a, Geometry b)
```

This function returns true if the geometries spatially intersect in 2D (for example, share any portion of space). This function is equivalent to NOT st_disjoint(a, b).

Example:

- Query command:

  select st_intersects((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```


st_length

```
Double st_length(Geometry geom)
```

This function returns the 2D path length of linear geometries, or perimeter of areal geometries, in units of the coordinate reference system (for example, degrees for EPSG:4236). It returns 0.0 for other geometry types (for example, Point).

Example:

- Query command:

  select st_length(st_geomFromWKT(location)) from geotbl where id='4'

- The query result is as follows:

  ```
  160.12062648706927
  ```

st_lengthSphere

```
Double st_lengthSphere(LineString line)
```

This function approximates the 2D path length of a LineString geometry using a spherical earth model. The returned length is in units of meters. The approximation is within 0.3% of st_lengthSpheroid and is computationally more efficient.

Example:

- Query command:

  select st_lengthSphere(st_castToLineString(st_geomFromWKT(location))) from geotbl where id='5'

- The query result is as follows:

  ```
  1.0013773137296816E7
  ```

st_lengthSpheroid

```
Double st_lengthSpheroid(LineString line)
```

This function calculates the 2D path length of a LineString geometry defined with longitude/latitude coordinates on the WGS84 spheroid. The returned length is in units of meters.

Example:

- Query command:

  select st_lengthSpheroid(st_castToLineString(st_geomFromWKT(location))) from geotbl where id='5'

- The query result is as follows:

  ```
  1.0001465052274073E7
  ```

st_overlaps

```
Boolean st_overlaps(Geometry a, Geometry b)
```

This function returns true if the geometries have some but not all points in common, are of the same dimension, and the intersection of the interiors of the two geometries has the same dimension as the geometries themselves.

Example:

- Query command:

  select st_overlaps((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```

st_relate

```
String st_relate(Geometry a, Geometry b)
```

This function returns the DE-9IM 3x3 interaction matrix pattern describing the dimensionality of the intersections between the interior, boundary and exterior of the two geometries.

Example:

- Query command:

  select st_relate((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FF2FF1212
```

st_relateBool

```
Boolean st_relateBool(Geometry a, Geometry b, String mask)
```

This function returns true if the DE-9IM interaction matrix mask matches the interaction matrix pattern obtained from st_relate(a, b).

Example:

- Query command:

  select st_relateBool((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'),'FF2FF1212')

- The query result is as follows:

```
TRUE
```

st_touches

```
Boolean st_touches(Geometry a, Geometry b)
```

This function returns true if the geometries have at least one point in common, but their interiors do not intersect.

Example:

- Query command:

  select st_touches((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```

st_within

```
Boolean st_within(Geometry a, Geometry b)
```

This function returns true if geometry a is completely inside geometry b.

Example:

- Query command:

  select st_within((select st_geomFromWKT(location) from geotbl where id='1'),(select st_geomFromWKT(location) from geotbl where id='2'))

- The query result is as follows:

```
FALSE
```

10.2.9 Geometry Processing

st_antimeridianSafeGeom

```
Geometry st_antimeridianSafeGeom(Geometry geom)
```

If geom spans the antimeridian, attempt to convert the geometry into an equivalent form that is "antimeridian-safe" (for example, the output geometry is covered by BOX(-180 -90, 180 90)). In certain circumstances, this method may fail, in which case the input geometry will be returned and an error will be logged.

Example:

- Query command:

  select st_astext(st_antimeridianSafeGeom(st_geomFromWKT(location))) from geotbl where id='5'

- The query result is as follows:

```
LINESTRING (30 10, 40 40, 20 40, 10 20, 30 10)
```

st_idlSafeGeom

Alias for **st_antimeridianSafeGeom**.

Example:

- Query command:

  select st_astext(st_idlSafeGeom(st_geomFromWKT(location))) from geotbl where id='6'

- The query result is as follows:

```
POINT (11 22)
```

st_bufferPoint

```
Geometry st_bufferPoint(Point p, Double buffer)
```

This function returns a Geometry covering all points within a given radius of Point p, where radius is given in meters.

Example:

- Query command:

  select st_astext(st_bufferPoint(st_castToPoint(st_geomFromWKT(location)), 0.1)) from geotbl where id='6'

- The query result is as follows:

```
10.999999429878716 22.000000784704625, 10.999999381731708 22.00000074735798,
10.999999336024723 22.00000070706185, 10.999999292938147 22.000000663975275,
10.99999925264202 22.00000061826829, 10.999999215295372 22.000000570121284,
```



```
10.999999181045595 22.000000519724267, 10.99999915002786 22.000000467276138,
10.999999122364573 22.000000412983884, 10.999999098164913 22.00000035706177,
10.999999077524384 22.000000299730495, 10.999999060524447 22.000000241216323,
10.999999047232189 22.00000018175018, 10.99999903770007 22.000000121566753,
10.999999031965709 22.000000060903556, 10.999999030051738 22, 10.999999031965709
21.999999939096444, 10.99999903770007 21.999999878433247, 10.999999047232189
21.99999981824982, 10.999999060524447 21.999999758783677, 10.999999077524384
21.999999700269505, 10.999999098164913 21.99999964293823, 10.999999122364573
21.999999587016116, 10.99999915002786 21.999999532723862, 10.999999181045595
21.999999480275733, 10.999999215295372 21.999999429878716, 10.99999925264202
21.99999938173171, 10.999999292938147 21.999999336024725, 10.999999336024723
21.99999929293815, 10.999999381731708 21.99999925264202, 10.999999429878716
21.999999215295375, 10.999999480275731 21.9999991810456, 10.99999953272386
21.99999915002786, 10.999999587016116 21.999999122364574, 10.99999964293823
21.999999098164913, 10.999999700269504 21.999999077524386, 10.999999758783677
21.99999906052445, 10.999999818249819 21.99999904723219, 10.999999878433249
21.99999903770007, 10.999999939096444 21.99999903196571, 11 21.99999903005174,
11.000000060903556 21.99999903196571, 11.000000121566751 21.99999903770007,
11.000000181750181 21.99999904723219, 11.000000241216323 21.99999906052445,
11.000000299730496 21.999999077524386, 11.00000035706177 21.999999098164913,
11.000000412983884 21.999999122364574, 11.00000046727614 21.99999915002786,
11.000000519724269 21.9999991810456, 11.000000570121284 21.999999215295375,
11.000000618268292 21.99999925264202, 11.000000663975277 21.99999929293815,
11.000000707061853 21.999999336024725, 11.00000074735798 21.99999938173171,
11.000000784704628 21.999999429878716, 11.000000818954405 21.999999480275733,
11.00000084997214 21.999999532723862, 11.000000877635427 21.999999587016116,
11.000000901835087 21.99999964293823, 11.000000922475616 21.999999700269505,
11.000000939475553 21.999999758783677, 11.000000952767811 21.99999981824982,
11.00000096229993 21.999999878433247, 11.000000968034291 21.999999939096444,
11.000000969948262 22))
```

st_convexHull

```
Geometry st_convexHull(Geometry geom)
```

Aggregate function. The convex hull of a geometry represents the minimum convex geometry that encloses all geometries geom in the aggregated rows.

