Cloud Data Migration

FAQ

Issue 28

Date 2020-08-25

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.

Contents

1 General....1
1.1 What Are the Advantages of CDM?....1
1.2 What Kind of Service Data Can Be Migrated Using CDM?....2
1.3 What Are the Security Protection Mechanisms of CDM?....6
1.4 How Is the Migration Performance of CDM?....6
1.5 How Do I Reduce Migration Costs?....6
2 Functions....8
2.1 Does CDM Support Incremental Data Migration?....8
2.2 Does CDM Support Field Conversion?....8
2.3 What Data Formats Are Supported When the Data Source Is Hive?....17
2.4 Can I Synchronize Jobs to Other Clusters?....17
2.5 Can I Create Jobs in Batches?....17
2.6 Can I Schedule Jobs in Batches?....17
2.7 Can I Back Up Jobs?....18
2.8 How Do I Use Java to Invoke CDM RESTful APIs to Create Data Migration Jobs?....18
2.9 How Do I Connect the On-Premises Intranet or Third-Party Private Network to CDM?....24
2.10 What Is the Migration Performance in the Same VPC and Different VPCs?....26
3 Troubleshooting....27
3.1 What Do I Do If the Log Prompts that the Date Format Fails to Be Parsed?....27
3.2 What Can I Do If the Map Field Tab Page Cannot Display All Columns?....29
3.3 How Do I Select Distribution Columns When Using CDM to Migrate Data to DWS?....33
3.4 What Do I Do If the Error Message "value too long for type character varying" Is Displayed When I Migrate Data to DWS?....34
3.5 What Can I Do If Error Message "Unable to execute the SQL statement" Is Displayed When I Import Data from OBS to SQL Server?....35

1 General

1.1 What Are the Advantages of CDM?

CDM is developed based on a distributed computing framework and leverages the parallel data processing technology. **Table 1-1** details the advantages of CDM.

Table 1-1 CDM advantages

Item	User-Developed Script	CDM
Ease of use	You need to prepare server resources, and install and configure software, which is time-consuming. Because the data source types are different, the program uses different access interfaces, such as JDBC and native APIs, to read and write data. In this case, various libraries and SDKs are required when you write data migration scripts, resulting in high development and management costs.	CDM provides a web-based management console for enabling services on web pages in real time. You can migrate data by configuring data sources and migration jobs on the GUI and CDM will manage and maintain the data sources and migration jobs for you. In other words, you only need to focus on the data migration logic without worrying about the environment, which greatly reduces development and maintenance costs. CDM also provides RESTful APIs to support third-party system calling and integration.
Real-time monitoring	You need to select specific versions to develop as required.	You can use Cloud Eye to automatically monitor CDM clusters in real time and manage alarms and notifications, so that you can keep track of CDM cluster performance metrics.

Item	User-Developed Script	CDM
O&M free	You need to develop and optimize O&M functions, especially alarm and notification functions, to ensure system availability. Otherwise, manual attendance is required.	With CDM, you do not need to maintain resources such as servers and VMs. CDM has the log, monitoring, and alarm functions, which send notifications to related personnel in a timely manner to avoid 24/7 hours of manual O&M.
High efficiency	During data migration, the read and write process is completed in one job. Limited by available resources, the performance is poor and cannot meet the requirements of scenarios where massive sets of data need to be migrated.	Based on the distributed computing framework, CDM jobs are split into independent sub-jobs and executed concurrently, which drastically improves data migration efficiency. In addition, efficient data import interfaces are provided to import data from Hive, HBase, MySQL databases, and Data Warehouse Service (DWS).
Various data sources	Different tasks must be developed for different data sources, generating a number of scripts.	Data sources such as databases, Hadoop services, NoSQL databases, data warehouses, and files are supported.
Different network environments	As the cloud computing technology develops, user data may be stored in different environments, such as public clouds, on-premises or hosted Internet data centers (IDCs), and hybrid scenarios. In heterogeneous environments, data migration is subject to various factors, for example, network connectivity, which causes inconvenience for development and maintenance.	CDM helps you easily cope with various data migration scenarios, including data migration to the cloud, data exchange on the cloud, and data migration to on-premises service systems, regardless of whether the data is stored on on-premises IDCs, cloud services, third-party clouds, or self-built databases or file systems on ECSs.

1.2 What Kind of Service Data Can Be Migrated Using CDM?

Cloud Data Migration (CDM) implements data mobility by enabling batch data migration among homogeneous and heterogeneous data sources. It supports on-premises and cloud file systems, relational databases, data warehouses, NoSQL, big data, and object storage.

CDM supports the following migration modes:

- Table/file migration: It is applicable to data migration to the cloud, data exchange on the cloud, and data migration to on-premises service systems.
- Entire DB migration: It is applicable to database migration to the cloud.

Table 1-2 describes the supported data sources.

Table 1-2 Supported data sources during table/file migration

Category	Data Source	Export	Import
Data warehouse	Data Warehouse Service (DWS)	Supported	Supported
	Data Lake Insight (DLI)	Not supported	Supported
	FusionInsight LibrA	Supported	Supported
Hadoop	MRS HDFS	Supported	Supported
	MRS HBase	Supported	Supported
	MRS Hive	Supported	Supported
	FusionInsight HDFS	Supported	Supported
	Apache HDFS	Supported	Supported
	Hadoop release version	Not supported	Not supported
	FusionInsight HBase	Supported	Supported
	FusionInsight Hive	Supported	Supported
	Apache HBase	Supported	Supported
	Apache Hive	Supported	Supported
Object storage	Object Storage Service (OBS)	Supported	Supported
	Alibaba Cloud Object Storage Service (OSS)	Supported	Not supported
	Qiniu Cloud Object Storage (KODO)	Supported	Not supported
	Amazon Simple Storage Service (Amazon S3)	Supported	Not supported
	Tencent Cloud Object Storage (COS)	Supported	Not supported
File system	FTP	Supported	Supported
	SFTP	Supported	Supported
	HTTP	Supported	Not supported

Category	Data Source	Export	Import
	Network Attached Storage (NAS)	Supported	Supported
	Scalable File Service (SFS Turbo)	Supported	Supported
Relational database	RDS for MySQL	Supported	Supported
	RDS for PostgreSQL	Supported	Supported
	RDS for SQL Server	Supported	Supported
	Distributed Database Middleware (DDM)	Supported	Not supported
	MySQL	Supported	Supported
	PostgreSQL	Supported	Supported
	Microsoft SQL Server	Supported	Supported
	Oracle	Supported	Supported
	IBM Db2	Supported	Supported
	Derecho (GaussDB)	Supported	Supported
	NewSQL (GaussDB)	Supported	Supported
	SAP HANA	Supported	Supported
	MYCAT	Supported	Supported
NoSQL	Distributed Cache Service (DCS)	Not supported	Supported
	Document Database Service (DDS)	Supported	Supported
	CloudTable Service (CloudTable)	Supported	Supported
	CloudTable OpenTSDB	Supported	Supported
	Redis	Supported	Supported
	MongoDB	Supported	Supported
	Cassandra	Supported	Supported
Search	Cloud Search Service (CSS)	Supported	Supported
	Elasticsearch	Supported	Supported
Message system	Data Ingestion Service (DIS)	Supported	Supported
	Apache Kafka	Supported	Supported

Category	Data Source	Export	Import
	MRS Kafka	Supported	Supported
	DMS Kafka	Supported	Supported

NOTE

- In the preceding table, the non-cloud data sources, such as MySQL, can be the on-premises MySQL, MySQL built on ECSs, or MySQL on the third-party cloud.
- When DIS, Apache Kafka, or DMS Kafka functions as the migration source, CDM can migrate data to only the following types of data sources:
  - CSS
  - DIS
  - Apache Kafka
  - DMS Kafka

Table 1-3 lists the data sources supporting entire DB migration using Cloud Data Migration.

Table 1-3 Supported data sources in entire DB migration

Source Data Type	Destination Data Type					
	RDS for MySQL	MRS (Hive)	DWS	CSS	OBS	CloudTable
MySQL	√	√	√	×	√	×
PostgreSQL	√	√	√	×	√	×
Microsoft SQL Server	√	√	√	×	×	×
Oracle	√	√	√	×	√	×
Elasticsearch	×	×	×	√	×	×
MongoDB	×	×	×	×	×	×
HBase	×	×	×	×	×	√
IBM Db2	√	√	√	×	√	×
Derecho (GaussDB)	√	√	√	×	√	×
DDM	√	√	√	×	√	×
SAP HANA	√	√	√	×	√	×
DWS	√	√	√	×	×	×
Hive	√	×	√	×	×	×

1.3 What Are the Security Protection Mechanisms of CDM?

CDM is a fully hosted service that provides the following capabilities to protect user data security:

- Instance isolation: CDM users can use only their own instances. Instances are isolated from each other and cannot access each other.
- System hardening: System hardening for security has been performed on the operating system of the CDM instance, so attackers cannot access the operating system from the Internet.
- Key encryption: Keys of various data sources entered when users create links on CDM are stored in CDM databases using high-strength encryption algorithms.
- No intermediate storage: During data migration, CDM processes only data mapping and conversion without storing any user data or data fragments.

1.4 How Is the Migration Performance of CDM?

Theoretically, a single CDM instance allows 1 TB to 8 TB data to be migrated per day, depending on the network bandwidth and read and write performance of the data source. Different business departments, such as finance and online mall, can use different CDM instances.

1.5 How Do I Reduce Migration Costs?

1. If data is exported at a specified time every day, you can use the CDM shutdown function. You can start the CDM cluster only when data needs to be migrated. You are charged for only ¥0.05 per hour (¥1.2 per day) for a stopped cluster.
2. When migrating the data on the public network, use **NAT Gateway** on HUAWEI CLOUD to share the EIPs with other ECSs in the subnet. In this way, data on the on-premises data center or third-party cloud can be migrated in a more economical and convenient manner.

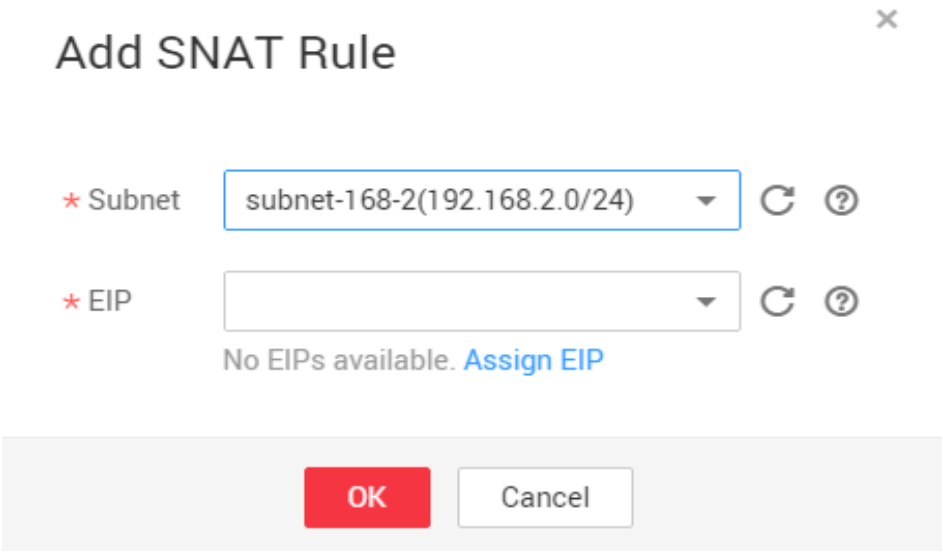
   The following details the operations:

   a. Suppose that you have created a CDM cluster (no dedicated EIP needs to be bound to the CDM cluster). Record the VPC and subnet where the CDM cluster is located.

   b. Create a NAT gateway. Select the same VPC and subnet as the CDM cluster.

   c. After the NAT gateway is created, return to the NAT gateway console list, click the created gateway name, and then click **Add SNAT Rule**.

Figure 1-1 Adding an SNAT rule



d. Select a subnet and an EIP. If no EIP is available, apply for one.

After that, access the CDM management console and migrate data from the public network to the cloud through the Internet. For example, migrate files from the FTP server in the on-premises data center to OBS and migrate relational databases from the third-party cloud to RDS.

NOTE

If SSL encryption is configured for the access channel of a local data source, CDM cannot connect to the data source using the EIP.

2 Functions

2.1 Does CDM Support Incremental Data Migration?

CDM supports incremental data migration. With scheduled jobs and macro variables of date and time, CDM provides incremental data migration in the following scenarios:

- **Incremental File Migration**
- **Incremental Migration of Relational Databases**
- **HBase/CloudTable Incremental Migration**
- **Incremental Synchronization Using the Macro Variables of Date and Time**

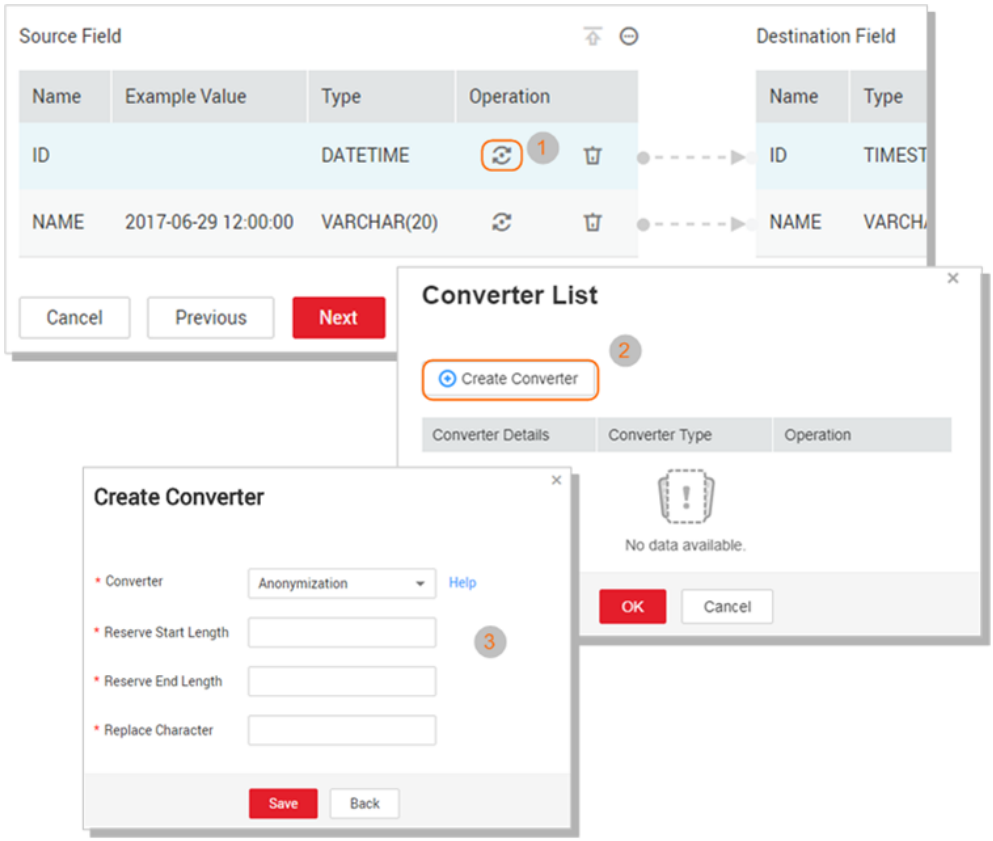
2.2 Does CDM Support Field Conversion?

Yes. CDM supports the following field converters:

- **Anonymization**
- **Trim**
- **Reverse String**
- **Replace String**
- **Expression Conversion**

You can create a field converter on the **Map Field** tab page when creating a table/file migration job. See **Figure 2-1**.

Figure 2-1 Creating a field converter

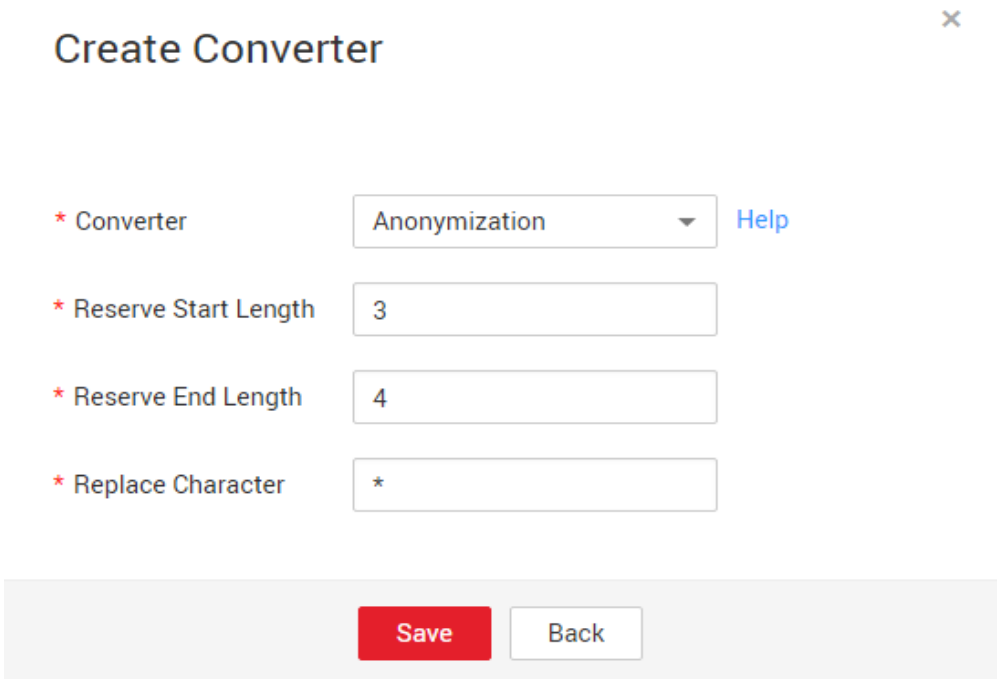


Anonymization

This converter is used to hide key information about the character string. For example, if you want to convert **12345678910** to **123****8910**, configure the parameters as follows:

- Set **Reserve Start Length** to **3**.
- Set **Reserve End Length** to **4**.
- Set **Replace Character** to *****.

Figure 2-2 Anonymization



Trim

This converter is used to automatically delete the spaces before and after a string. No parameters need to be configured.

Reverse String

This converter is used to automatically reverse a string. For example, reverse **ABC** into **CBA**. No parameters need to be configured.

Replace String

This converter is used to replace a character string. You need to configure the object to be replaced and the new value.

Expression Conversion

This converter uses the JSP expression language (EL) to convert the current field or a row of data. The JSP EL is used to create arithmetic and logical expressions. Within a JSP EL expression, you can use integers, floating point numbers, strings, the built-in constants **true** and **false** for boolean values, and **null**.

The expression supports the following environment variables:

- **value**: indicates the current field value.
- **row**: indicates the current row, which is an array type.

The expression supports the following tool classes:

- StringUtils: string processing tool class. For details, see **org.apache.commons.lang.StringUtils** of the Java SDK code.

- DateUtils: date tool class
- CommonUtils: common tool class
- NumberUtils: string-to-value conversion class
- HttpsUtils: network file read class

Application examples:

1. Set a string constant for the current field, for example, VIP.
   Expression: **"***VIP***"**
2. If the field is of the string type, convert all character strings into lowercase letters, for example, convert **aBC** to **abc**.
   Expression: StringUtils.lowerCase(value)
3. Convert all character strings of the current field to uppercase letters.
   Expression: StringUtils.upperCase(value)
4. If the field value is a date string in *yyyy-MM-dd* format, extract the year from the field value, for example, extract **2017** from **2017-12-01**.
   Expression: StringUtils.substringBefore(value,"-")
5. If the field value is of the numeric type, convert the value to a new value which is two times greater than the original value:
   Expression: value*2
6. Convert the field value **true** to **Y** and other field values to **N**.
   Expression: value=="true"?"Y":"N"
7. If the field value is of the string type and is left empty, convert it to **Default**. Otherwise, the field value will not be converted.
   Expression: empty value? "Default":value
8. If the first and second fields are of the numeric type, convert the field to the combination of the first and second field values.
   Expression: row[0]+row[1]
9. If the field is of the date or timestamp type, return the current year after conversion. The data type is int.
   Expression: DateUtils.getYear(value)
10. If the field is a date and time string in *yyyy-MM-dd* format, convert it to the date type:
    Expression: DateUtils.format(value,"yyyy-MM-dd")
11. Convert date format **2018/01/05 15:15:05** to **2018-01-05 15:15:05**:
    Expression: DateUtils.format(DateUtils.parseDate(value,"yyyy/MM/dd HH:mm:ss"),"yyyy-MM-dd HH:mm:ss")
12. Obtain a 36-bit universally unique identifier (UUID):
    Expression: CommonUtils.randomUUID()
13. If the field is of the string type, capitalize the first letter, for example, convert **cat** to **Cat**.
    Expression: StringUtils.capitalize(value)
14. If the field is of the string type, convert the first letter to a lowercase letter, for example, convert **Cat** to **cat**.
    Expression: StringUtils.uncapitalize(value)

15. If the field is of the string type, use a space to fill in the character string to the specified length and center the character string. If the length of the character string is not shorter than the specified length, do not convert the character string. For example, convert **ab** to meet the specified length 4.

    Expression: StringUtils.center(value,*4*)

16. Delete a newline (including **\n**, **\r**, and **\r\n**) at the end of a character string. For example, convert **abc\r\n\r\n** to **abc\r\n**.

    Expression: StringUtils.chomp(value)

17. If the string contains the specified string, **true** is returned; otherwise, **false** is returned. For example, **abc** contains **a** so that **true** is returned.

    Expression: StringUtils.contains(value,"*a*")

18. If the string contains any character of the specified string, **true** is returned; otherwise, **false** is returned. For example, **zzabyycdxx** contains either **z** or **a** so that **true** is returned.

    Expression: StringUtils.containsAny("value","*za*")

19. If the string does not contain any one of the specified characters, **true** is returned. If any specified character is contained, **false** is returned. For example, **abz** contains one character of **xyz** so that **false** is returned.

    Expression: StringUtils.containsNone(value,"*xyz*")

20. If the string contains only the specified characters, **true** is returned. If any other character is contained, **false** is returned. For example, **abab** contains only characters among **abc** so that **true** is returned.

    Expression: StringUtils.containsOnly(value,"*abc*")

21. If the character string is empty or null, convert it to the specified character string. Otherwise, do not convert the character string. For example, convert the empty character string to null.

    Expression: StringUtils.defaultIfEmpty(value,*null*)

22. If the string ends with the specified suffix (case sensitive), **true** is returned; otherwise, **false** is returned. For example, if the suffix of **abcdef** is not null, **false** is returned.

    Expression: StringUtils.endsWith(value,*null*)

23. If the string is the same as the specified string (case sensitive), **true** is returned; otherwise, **false** is returned. For example, after strings **abc** and **ABC** are compared, **false** is returned.

    Expression: StringUtils.equals(value,"*ABC*")

24. Obtain the first index of the specified character string in a character string. If no index is found, **-1** is returned. For example, the first index of **ab** in **aabaabaa** is 1.

    Expression: StringUtils.indexOf(value,"*ab*")

25. Obtain the last index of the specified character string in a character string. If no index is found, **-1** is returned. For example, the last index of **k** in **aFkyk** is 4.

    Expression: StringUtils.lastIndexOf(value,"*k*")

26. Obtain the first index of the specified character string from the position specified in the character string. If no index is found, **-1** is returned. For example, the first index of **b** obtained after the index 3 of **aabaabaa** is 5.

Expression: StringUtils.indexOf(value,"*b*",*3*)

27. Obtain the first index of any specified character in a character string. If no index is found, **-1** is returned. For example, the first index of **z** or **a** in **zzabyycdxx.** is 0.

    Expression: StringUtils.indexOfAny(value,"*za*")

28. If the string contains any Unicode character, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains only non-Unicode characters so that **false** is returned.

    Expression: StringUtils.isAlpha(value)

29. If the string contains only Unicode characters and digits, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains only Unicode characters and digits, so that **true** is returned.

    Expression: StringUtils.isAlphanumeric(value)

30. If the string contains only Unicode characters, digits, and spaces, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains only Unicode characters and digits, so that **true** is returned.

    Expression: StringUtils.isAlphanumericSpace(value)

31. If the string contains only Unicode characters and spaces, **true** is returned; otherwise, **false** is returned. For example, **ab2c** contains Unicode characters and digits so that **false** is returned.

    Expression: StringUtils.isAlphaSpace(value)

32. If the string contains only printable ASCII characters, **true** is returned; otherwise, **false** is returned. For example, for **!ab-c~**, **true** is returned.

    Expression: StringUtils.isAsciiPrintable(value)

33. If the string is empty or null, **true** is returned; otherwise, **false** is returned.

    Expression: StringUtils.isEmpty(value)

34. If the string contains only Unicode digits, **true** is returned; otherwise, **false** is returned.

    Expression: StringUtils.isNumeric(value)

35. Obtain the leftmost characters of the specified length. For example, obtain the leftmost two characters **ab** from **abc**.

    Expression: StringUtils.left(value,*2*)

36. Obtain the rightmost characters of the specified length. For example, obtain the rightmost two characters **bc** from **abc**.

    Expression: StringUtils.right(value,*2*)

37. Concatenate the specified character string to the left of the current character string and specify the length of the concatenated character string. If the length of the current character string is not shorter than the specified length, the character string will not be converted. For example, if **yz** is concatenated to the left of **bat** and the length must be 8 after concatenation, the character string is **yzyzybat** after conversion.

    Expression: StringUtils.leftPad(value,*8*,"*yz*")

38. Concatenate the specified character string to the right of the current character string and specify the length of the concatenated character string. If the length of the current character string is not shorter than the specified length, the character string will not be converted. For example, if **yz** is

concatenated to the right of **bat** and the length must be 8 after concatenation, the character string is **batyzyzy** after conversion.

Expression: StringUtils.rightPad(value,*8*,"*yz*")

39. If the field is of the string type, obtain the length of the current character string. If the character string is null, **0** is returned.

    Expression: StringUtils.length(value)

40. If the field is of the string type, delete all the specified character strings from it. For example, delete **ue** from **queued** to obtain **qd**.

    Expression: StringUtils.remove(value,"*ue*")

41. If the field is of the string type, remove the substring at the end of the field. If the specified substring is not at the end of the field, no conversion is performed. For example, remove **.com** at the end of **www.domain.com**.

    Expression: StringUtils.removeEnd(value,"*.com*")

42. If the field is of the string type, delete the substring at the beginning of the field. If the specified substring is not at the beginning of the field, no conversion is performed. For example, delete **www.** at the beginning of **www.domain.com**.

    Expression: StringUtils.removeStart(value,"*www.*")

43. If the field is of the string type, replace all the specified character strings in the field. For example, replace **a** in **aba** with **z** to obtain **zbz**.

    Expression: StringUtils.replace(value,"*a*","*z*")

44. If the field is of the string type, replace multiple characters in the character string at a time. For example, replace **h** in **hello** with **j** and **o** with **y** to obtain **jelly**.

    Expression: StringUtils.replaceChars(value,"*ho*","*jy*")

45. If the field is of the string type, use the specified delimiter to split the text into arrays. For example, use **:** to split **ab:cd:ef** into **["ab","cd","ef"]**.

    Expression: StringUtils.split(value,":")

46. If the string starts with the specified prefix (case sensitive), **true** is returned; otherwise, **false** is returned. For example, **abcdef** starts with **abc**, so that **true** is returned.

    Expression: StringUtils.startsWith(value,"*abc*")

47. If the field is of the string type, delete all the specified characters from the field. For example, delete all **x**, **y**, and **z** from **abcyx** to obtain **abc**.

    Expression: StringUtils.strip(value,"*xyz*")

48. If the field is of the string type, delete all the specified characters at the end of the field, for example, delete all spaces at the end of the field.

    Expression: StringUtils.stripEnd(value,*null*)

49. If the field is of the string type, delete all the specified characters at the beginning of the field, for example, delete all spaces at the beginning of the field.

    Expression: StringUtils.stripStart(value,*null*)

50. If the field is of the string type, obtain the substring after the specified position (excluding the character at the specified position) of the character string. If the specified position is a negative number, calculate the position in

the descending order. For example, obtain the character string after the second character of **abcde**, that is, **cde**.

Expression: StringUtils.substring(value,*2*)

51. If the field is of the string type, obtain the substring within the specified range of the character string. If the specified range is a negative number, calculate the range in the descending order. For example, obtain the character string between the second and fifth characters of **abcde**, that is, **cd**.

    Expression: StringUtils.substring(value,*2*,*5*)

52. If the field is of the string type, obtain the substring after the first specified character. For example, obtain the substring after the first **b** in **abcba**, that is, **cba**.

    Expression: StringUtils.substringAfter(value,"*b*")

53. If the field is of the string type, obtain the substring after the last specified character. For example, obtain the substring after the last **b** in **abcba**, that is, **a**.

    Expression: StringUtils.substringAfterLast(value,"*b*")

54. If the field is of the string type, obtain the substring before the first specified character. For example, obtain the substring before the first **b** in **abcba**, that is, **a**.

    Expression: StringUtils.substringBefore(value,"*b*")

55. If the field is of the string type, obtain the substring before the last specified character. For example, obtain the substring before the last **b** in **abcba**, that is, **abc**.

    Expression: StringUtils.substringBeforeLast(value,"*b*")

56. If the field is of the string type, obtain the substring nested within the specified string. If no substring is found, **null** is returned. For example, obtain the substring between **tag** in **tagabctag**, that is, **abc**.

    Expression: StringUtils.substringBetween(value,"*tag*")

57. If the field is of the string type, delete the control characters (char≤32) at both ends of the character string, for example, delete the spaces at both ends of the character string.

    Expression: StringUtils.trim(value)

58. Convert the character string to a value of the byte type. If the conversion fails, **0** is returned.

    Expression: NumberUtils.toByte(value)

59. Convert the character string to a value of the byte type. If the conversion fails, the specified value, for example, **1**, is returned.

    Expression: NumberUtils.toByte(value,*1*)

60. Convert the character string to a value of the double type. If the conversion fails, **0.0d** is returned.

    Expression: NumberUtils.toDouble(value)

61. Convert the character string to a value of the double type. If the conversion fails, the specified value, for example, **1.1d**, is returned.

    Expression: NumberUtils.toDouble(value,*1.1d*)

62. Convert the character string to a value of the float type. If the conversion fails, **0.0f** is returned.

Expression: NumberUtils.toFloat(value)

63. Convert the character string to a value of the float type. If the conversion fails, the specified value, for example, **1.1f**, is returned.

    Expression: NumberUtils.toFloat(value,*1.1f*)

64. Convert the character string to a value of the int type. If the conversion fails, **0** is returned.

    Expression: NumberUtils.toInt(value)

65. Convert the character string to a value of the int type. If the conversion fails, the specified value, for example, **1**, is returned.

    Expression: NumberUtils.toInt(value,*1*)

66. Convert the character string to a value of the long type. If the conversion fails, **0** is returned.

    Expression: NumberUtils.toLong(value)

67. Convert the character string to a value of the long type. If the conversion fails, the specified value, for example, **1L**, is returned.

    Expression: NumberUtils.toLong(value,*1L*)

68. Convert the character string to a value of the short type. If the conversion fails, **0** is returned.

    Expression: NumberUtils.toShort(value)

69. Convert the character string to a value of the short type. If the conversion fails, the specified value, for example, **1**, is returned.

    Expression: NumberUtils.toShort(value,*1*)

70. Convert the IP string to a value of the long type, for example, convert **10.78.124.0** to **172915712**.

    Expression: CommonUtils.ipToLong(value)

71. Read an IP address and physical address mapping file from the network, and download the mapping file to the map collection. *url* indicates the address for storing the IP mapping file, for example, **http://10.114.205.45:21203/sqoop/IpList.csv**.

    Expression: HttpsUtils.downloadMap("*url*")

72. Cache the IP address and physical address mappings and specify a key for retrieval, for example, **ipList**.

    Expression: CommonUtils.setCache("*ipList*",HttpsUtils.downloadMap("*url*"))

73. Obtain the cached IP address and physical address mappings.

    Expression: CommonUtils.getCache("*ipList*")

74. Check whether the IP address and physical address mappings are cached.

    Expression: CommonUtils.cacheExists("*ipList*")

75. Obtain the physical addresses corresponding to the IP address in *Country_Province_City_Carrier* format. For example, the physical address corresponding to **1xx.78.124.0** is **China_Guangdong_Shenzhen_China Telecom**. If the corresponding physical address cannot be obtained, the default value ****_*_*_**** is returned. If necessary, you can use the StringUtil class expression to further split the addresses.

    Expression:
    CommonUtils.getMapValue(CommonUtils.ipToLong(value),CommonUtils.cach

eExists("*ipLis*")?
CommonUtils.getCache("*ipLis*"):CommonUtils.setCache("*ipLis*",HttpsUtils.downloadMap("*url*")))

76. Based on the specified offset type (month/day/hour/minute/second) and offset (positive number indicates increase and negative number indicates decrease), convert the time in the specified format to a new time, for example, add 8 hours to **2019-05-21 12:00:00**.

    Expression: DateUtils.getCurrentTimeByZone("*yyyy-MM-dd HH:mm:ss*",value, "*hour*", *8*)

2.3 What Data Formats Are Supported When the Data Source Is Hive?

CDM can read and write data in SequenceFile, TextFile, ORC, or Parquet format from the Hive data source.

2.4 Can I Synchronize Jobs to Other Clusters?

CDM does not support direct job migration across clusters. However, you can use the batch job import/export function to indirectly implement cross-cluster migration as follows:

1. Export all jobs from CDM cluster 1 and save the jobs' JSON files to a local PC.

   For security purposes, no link password is exported when CDM exports jobs. All passwords are replaced by *Add password here*.

2. Edit each JSON file on the local PC by replacing *Add password here* with the actual password of the corresponding link.
3. Import the edited JSON files to CDM cluster 2 in batches to implement job migration between cluster 1 and cluster 2.

2.5 Can I Create Jobs in Batches?

CDM supports batch job creation with the help of the batch import function. You can create jobs in batches as follows:

1. Create a job manually.
2. Export the job and save the job's JSON file to a local PC.
3. Edit the JSON file and replicate more jobs in the JSON file according to the job configuration.
4. Import the JSON file to the CDM cluster to implement batch job creation.

2.6 Can I Schedule Jobs in Batches?

Symptom

I have many jobs in CDM. Can I schedule Jobs in batches?

Solution

1. Access the Data Development module of DADU.
2. In the navigation pane of the Data Development homepage, choose **Development** > **Develop Job** to create a job.
3. Drag multiple CDM Job nodes to the canvas and orchestrate the jobs.

2.7 Can I Back Up Jobs?

Yes. If you do not need to use the CDM cluster for a long time, you can stop or delete it to reduce costs.

Before the deletion, you can use the batch export function of CDM to save all job scripts to a local PC. Then, you can create a cluster and import the jobs again when necessary.

2.8 How Do I Use Java to Invoke CDM RESTful APIs to Create Data Migration Jobs?

CDM provides RESTful APIs to implement automatic job creation or execution control by program invocation.

The following describes how to use CDM to migrate data from table **city1** in the MySQL database to table **city2** on DWS, and how to use Java to invoke CDM RESTful APIs to create, start, query, and delete a CDM job.

Prepare the following data in advance:

1. Username, account name, and project ID of the cloud account

   Go to the **My Credentials** page to obtain the username and account name. In the project list, obtain the project ID of the corresponding region, for example, **1af30ca47b5a4eb987e325a846458b7a**.

2. Create a CDM cluster and obtain the cluster ID.

   On the **Cluster Management** page, click the CDM cluster name to view the cluster ID, for example, **c110beff-0f11-4e75-8b10-da7cd882b0ef**.

3. Create a MySQL database and a DWS database, and create tables **city1** and **city2**. The statements for creating tables are as follows:

   ```
   MySQL:
   create table city1(code varchar(10),name varchar(32));
   insert into city1 values('NY','New York');
   DWS:
   create table city2(code varchar(10),name varchar(32));
   ```

4. In the CDM cluster, create a link to MySQL, such as a link named **mysqltestlink**. Create a link to DWS, such as a link named **dwstestlink**.

5. Run the following code. You are advised to use the HttpClient package of version 4.5. Maven configuration is as follows:

   ```
   <project>
   <modelVersion>4.0.0</modelVersion>
   <groupId>cdm</groupId>
   <artifactId>cdm-client</artifactId>
   <version>1</version>
   <dependencies>
   ```



```
<dependency>
<groupId>org.apache.httpcomponents</groupId>
<artifactId>httpclient</artifactId>
<version>4.5</version>
</dependency>
</dependencies>
</project>
```

Sample Code

The code for using Java to invoke CDM RESTful APIs to create, start, query, and delete a CDM job is as follows:

```
package cdmclient;
import java.io.IOException;
import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
public class CdmClient {
private final static String DOMAIN_NAME="Cloud account name";
private final static String USER_NAME="Cloud username";
Private final static String USER_PASSWORD= "Password of the cloud user";
private final static String PROJECT_ID="Project ID";
private final static String CLUSTER_ID="CDM cluster ID";
private final static String JOB_NAME="Job name";
private final static String FROM_LINKNAME="Source link name";
private final static String TO_LINKNAME="Destination link name";
Private final static String IAM_ENDPOINT= "IAM endpoint";
Private final static String CDM_ENDPOINT= "CDM endpoint";
private CloseableHttpClient httpclient;
private String token;

public CdmClient() {
this.httpclient = createHttpClient();
this.token = login();
}

private CloseableHttpClient createHttpClient() {
CloseableHttpClient httpclient =HttpClients.createDefault();
return httpclient;
}

private String login(){
HttpPost httpPost = new HttpPost("https://"+IAM_ENDPOINT+"/v3/auth/tokens");
String json =
"{\r\n"+
"\"auth\": {\r\n"+
"\"identity\": {\r\n"+
```



```
"\"methods\": [\"password\"],\r\n"+
"\"password\": {\r\n"+
"\"user\": {\r\n"+
"\"name\": \""+USER_NAME+"\",\r\n"+
"\"password\": \""+USER_PASSWORD+"\",\r\n"+
"\"domain\": {\r\n"+
"\"name\": \""+DOMAIN_NAME+"\"\r\n"+
"}\r\n"+
"}\r\n"+
"}\r\n"+
"},\r\n"+
"\"scope\": {\r\n"+
"\"project\": {\r\n"+
"\"name\": \"PROJECT_NAME\"\r\n"+
"}\r\n"+
"}\r\n"+
"}\r\n"+
"}\r\n";
try {
StringEntity s = new StringEntity(json);
s.setContentEncoding("UTF-8");
s.setContentType("application/json");
httpPost.setEntity(s);
CloseableHttpResponse response = httpclient.execute(httpPost);
Header tokenHeader = response.getFirstHeader("X-Subject-Token");
String token = tokenHeader.getValue();
System.out.println("Login successful");
return token;
} catch (Exception e) {
throw new RuntimeException("login failed.", e);
}
}
/*Create a job.*/

public void createJob(){
HttpPost httpPost = new HttpPost("https://"+CDM_ENDPOINT+"/cdm/v1.0/"+PROJECT_ID+"/
clusters/"+CLUSTER_ID+"/cdm/job");

/**The JSON information here is complex. You can create a job on the job management page,
click Job JSON Definition next to the job, copy the JSON content and convert it into a Java
character string, and paste it here.
*In the JSON message body, you only need to replace the link name, data import and export
table names, field list of the tables, and fields used for partitioning in the source table.**/

String json =
"{\r\n"+
"\"jobs\": [\r\n"+
"{\r\n"+
"\"from-connector-name\": \"generic-jdbc-connector\",\r\n"+
"\"name\": \""+JOB_NAME+"\",\r\n"+
"\"to-connector-name\": \"generic-jdbc-connector\",\r\n"+
"\"driver-config-values\": {\r\n"+
"\"configs\": [\r\n"+
"{\r\n"+
"\"inputs\": [\r\n"+
"{\r\n"+
"\"name\": \"throttlingConfig.numExtractors\",\r\n"+
"\"value\": \"1\"\r\n"+
"}\r\n"+
"],\r\n"+
"\"validators\": [],\r\n"+
"\"type\": \"JOB\",\r\n"+
```



```
"\"id\": 30,\r\n"+
"\"name\": \"throttlingConfig\"\r\n"+
"}\r\n"+
"]\r\n"+
"},\r\n"+
"\"from-link-name\": \""+FROM_LINKNAME+"\",\r\n"+
"\"from-config-values\": {\r\n"+
"\"configs\": [\r\n"+
"{\r\n"+
"\"inputs\": [\r\n"+
"{\r\n"+
"\"name\": \"fromJobConfig.schemaName\",\r\n"+
"\"value\": \"sqoop\"\r\n"+
"},\r\n"+
"{\r\n"+
"\"name\": \"fromJobConfig.tableName\",\r\n"+
"\"value\": \"city1\"\r\n"+
"},\r\n"+
"{\r\n"+
"\"name\": \"fromJobConfig.columnList\",\r\n"+
"\"value\": \"code&name\"\r\n"+
"},\r\n"+
"{\r\n"+
"\"name\": \"fromJobConfig.partitionColumn\",\r\n"+
"\"value\": \"code\"\r\n"+
"}\r\n"+
"],\r\n"+
"\"validators\": [],\r\n"+
"\"type\": \"JOB\",\r\n"+
"\"id\": 7,\r\n"+
"\"name\": \"fromJobConfig\"\r\n"+
"}\r\n"+
"]\r\n"+
"},\r\n"+
"\"to-link-name\": \""+TO_LINKNAME+"\",\r\n"+
"\"to-config-values\": {\r\n"+
"\"configs\": [\r\n"+
"{\r\n"+
"\"inputs\": [\r\n"+
"{\r\n"+
"\"name\": \"toJobConfig.schemaName\",\r\n"+
"\"value\": \"sqoop\"\r\n"+
"},\r\n"+
"{\r\n"+
"\"name\": \"toJobConfig.tableName\",\r\n"+
"\"value\": \"city2\"\r\n"+
"},\r\n"+
"{\r\n"+
"\"name\": \"toJobConfig.columnList\",\r\n"+
"\"value\": \"code&name\"\r\n"+
"}, \r\n"+
"{\r\n"+
"\"name\": \"toJobConfig.shouldClearTable\",\r\n"+
"\"value\": \"true\"\r\n"+
"}\r\n"+
"],\r\n"+
"\"validators\": [],\r\n"+
"\"type\": \"JOB\",\r\n"+
"\"id\": 9,\r\n"+
"\"name\": \"toJobConfig\"\r\n"+
"}\r\n"+
"]\r\n"+
```



```
"}\r\n"+
"}\r\n"+
"]\r\n"+
"}\r\n";
try {
StringEntity s = new StringEntity(json);
s.setContentEncoding("UTF-8");
s.setContentType("application/json");
httpPost.setEntity(s);
httpPost.addHeader("X-Auth-Token", this.token);
httpPost.addHeader("X-Language", "en-us");
CloseableHttpResponse response = httpclient.execute(httpPost);
int status = response.getStatusLine().getStatusCode();
if(status == 200){
System.out.println("Create job successful.");
}else{
System.out.println("Create job failed.");
HttpEntity entity = response.getEntity();
System.out.println(EntityUtils.toString(entity));
}
} catch (Exception e) {
e.printStackTrace();
throw new RuntimeException("Create job failed.", e);
}
}
/*Start the job.*/

public void startJob(){
HttpPut httpPut = new HttpPut("https://"+CDM_ENDPOINT+"/cdm/v1.0/"+PROJECT_ID+"/
clusters/"+CLUSTER_ID+"/cdm/job/"+JOB_NAME+"/start");
String json = "";
try {
StringEntity s = new StringEntity(json);
s.setContentEncoding("UTF-8");
s.setContentType("application/json");
httpPut.setEntity(s);
httpPut.addHeader("X-Auth-Token", this.token);
httpPut.addHeader("X-Language", "en-us");
CloseableHttpResponse response = httpclient.execute(httpPut);
int status = response.getStatusLine().getStatusCode();
if(status == 200){
System.out.println("Start job successful.");
}else{
System.out.println("Start job failed.");
HttpEntity entity = response.getEntity();
System.out.println(EntityUtils.toString(entity));
}
} catch (Exception e) {
e.printStackTrace();
throw new RuntimeException("Start job failed.", e);
}
}
/*Query the job running status cyclically until the job is complete.*/

public void getJobStatus(){
HttpGet httpGet = new HttpGet("https://"+CDM_ENDPOINT+"/cdm/v1.0/"+PROJECT_ID+"/
clusters/"+CLUSTER_ID+"/cdm/job/"+JOB_NAME+"/status");
try {
httpGet.addHeader("X-Auth-Token", this.token);
httpGet.addHeader("X-Language", "en-us");
boolean flag = true;
while(flag){
```



```
CloseableHttpResponse response = httpclient.execute(httpGet);
int status = response.getStatusLine().getStatusCode();
if(status == 200){
HttpEntity entity = response.getEntity();
String msg = EntityUtils.toString(entity);
if(msg.contains("\"status\":\"SUCCEEDED\"")){
System.out.println("Job succeeded");
break;
}else if (msg.contains("\"status\":\"FAILED\"")){
System.out.println("Job failed.");
break;
}else{
Thread.sleep(1000);
}

}else{
System.out.println("Get job status failed.");
HttpEntity entity = response.getEntity();
System.out.println(EntityUtils.toString(entity));
break;
}
}
} catch (Exception e) {
e.printStackTrace();
throw new RuntimeException("Get job status failed.", e);
}
}
/*Delete the job.*/

public void deleteJob(){
HttpDelete httpDelte = new HttpDelete("https://"+CDM_ENDPOINT+"/cdm/v1.0/"+PROJECT_ID
+"/clusters/"+CLUSTER_ID+"/cdm/job/"+JOB_NAME);
try {
httpDelte.addHeader("X-Auth-Token", this.token);
httpDelte.addHeader("X-Language", "en-us");
CloseableHttpResponse response = httpclient.execute(httpDelte);
int status = response.getStatusLine().getStatusCode();
if(status == 200){
System.out.println("Delete job successful.");
}else{
System.out.println("Delete job failed.");
HttpEntity entity = response.getEntity();
System.out.println(EntityUtils.toString(entity));
}
} catch (Exception e) {
e.printStackTrace();
throw new RuntimeException("Delete job failed.", e);
}
}
/*Close the process.*/

public void close(){
try {
httpclient.close();
} catch (IOException e) {
throw new RuntimeException("Close failed.", e);
}
}

public static void main(String[] args){
CdmClient cdmClient = new CdmClient();
cdmClient.createJob();
```



```
cdmClient.startJob();
cdmClient.getJobStatus();
cdmClient.deleteJob();
cdmClient.close();
}
}
```

2.9 How Do I Connect the On-Premises Intranet or Third-Party Private Network to CDM?

Many enterprises deploy key data sources on the intranet, such as databases and file servers. CDM runs on the cloud. To migrate the intranet data to the cloud using CDM, use any of the following methods to connect the intranet to the cloud:

1. Bind the Internet IP addresses to the intranet data source nodes to enable CDM to access the data from the Internet directly.
2. Establish a VPN between the on-premises data center and the VPC where the service resides.
3. Use Direct Connect to connect the data center to the cloud service.
4. Leverage Network Address Translation (NAT) or port forwarding to access the network in proxy mode.

The following describes how to use the port forwarding tool to access intranet data. The process is as follows:

1. Use a Windows computer as the gateway. The computer must be able to access both the Internet and the intranet.
2. Install the port mapping tool IPOP on the computer.
3. Configure port mapping using the tool.

NOTICE

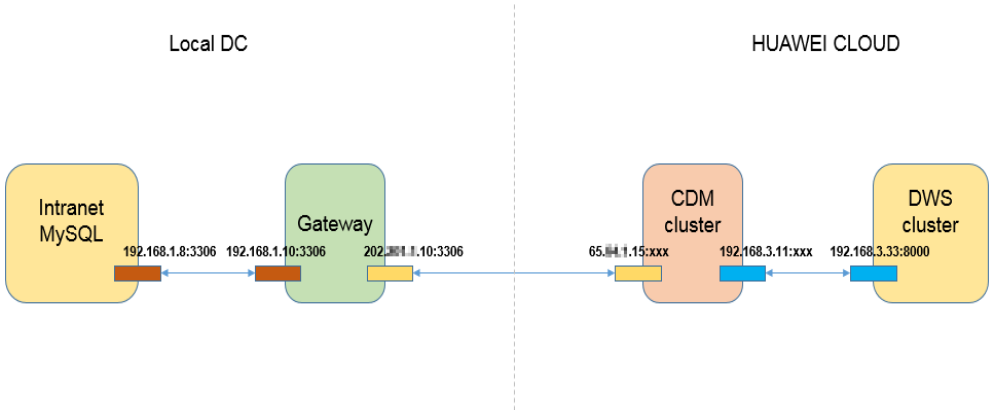
If the intranet database is exposed to the public network for a long time, security risks exist. Therefore, after data migration is complete, stop port mapping.

Scenario

Suppose that the MySQL database on the intranet is migrated to DWS. **Figure 2-3** shows the network topology.

In the figure, the intranet can be either an enterprise's data center or the intranet of the virtual data center on a third-party cloud.

Figure 2-3 Network topology example



Procedure

Step 1 Use a Windows computer as the gateway. Configure both the intranet and Internet IP addresses on the computer. Conduct the following test to check whether the gateway computer can fulfill service needs.

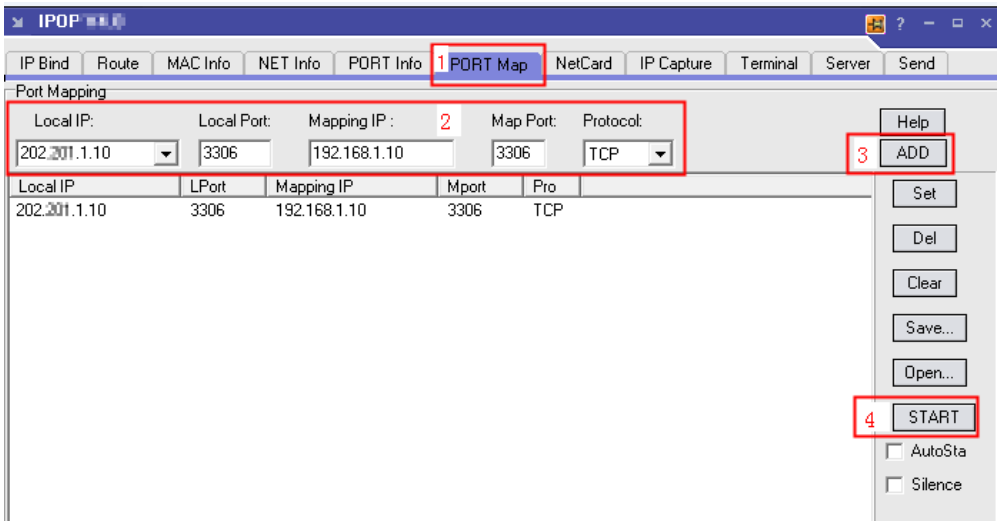
1. Run the **ping** command on the computer to check whether the intranet address of the MySQL database is pingable. For example, run **ping 192.168.1.8**.
2. Run the **ping** command on another computer that can access the Internet to check whether the public network address of the gateway computer is pingable. For example, run **ping 202.*xx.xx*.10**.

Step 2 Download the port mapping tool IPOP and install it on the gateway computer.

Step 3 Run the port mapping tool and select **PORT Map**. See **Figure 2-4**.

- **Local IP** and **Local Port**: Configure these two parameters to the public network address and port number of the gateway computer respectively, which must be entered when creating MySQL links on CDM.
- **Mapping IP** and **Map Port**: Configure these two parameters to the IP address and port number of the MySQL database on the intranet.

Figure 2-4 Configuring port mapping



Step 4 Click **ADD** to add a port mapping relationship.

Step 5 Click **START** to start mapping and receive data packets.

Then, you can use the EIP to read data from the MySQL database on the intranet on CDM and import the data to DWS.

NOTE

1. To access the on-premises data source, you must also bind an EIP to the CDM cluster.
2. Generally, DWS is accessible within the same VPC. When creating a CDM cluster, you must ensure that the VPC of the CDM cluster must be the same as that of DWS. In addition, it is recommended that CDM and DWS be in the same intranet and security group. If their security groups are different, you also need to enable data access between the security groups.
3. Port mapping can be used to migrate data between databases on the intranet or the SFTP servers.
4. For Linux computers, port mapping can also be implemented using IPTABLE.
5. When the FTP server on the intranet is mapped to the public network using port mapping, you need to check whether the PASV mode is enabled. In this case, the client and server are connected through a random port. Therefore, in addition to port 21 mapping, you also need to configure the port range mapping in PASV mode. For example, you can specify the **vsftp** port range by configuring **pasv_min_port** and **pasv_max_port**.

----End

2.10 What Is the Migration Performance in the Same VPC and Different VPCs?

The transmission rate depends on the bandwidth and file read/write speed.

3 Troubleshooting

3.1 What Do I Do If the Log Prompts that the Date Format Fails to Be Parsed?

Symptom

When CDM is used to migrate other data sources to CSS, the job fails to be executed and the error message "Unparseable date" is displayed in the log. See **Figure 3-1**.

Figure 3-1 Log output

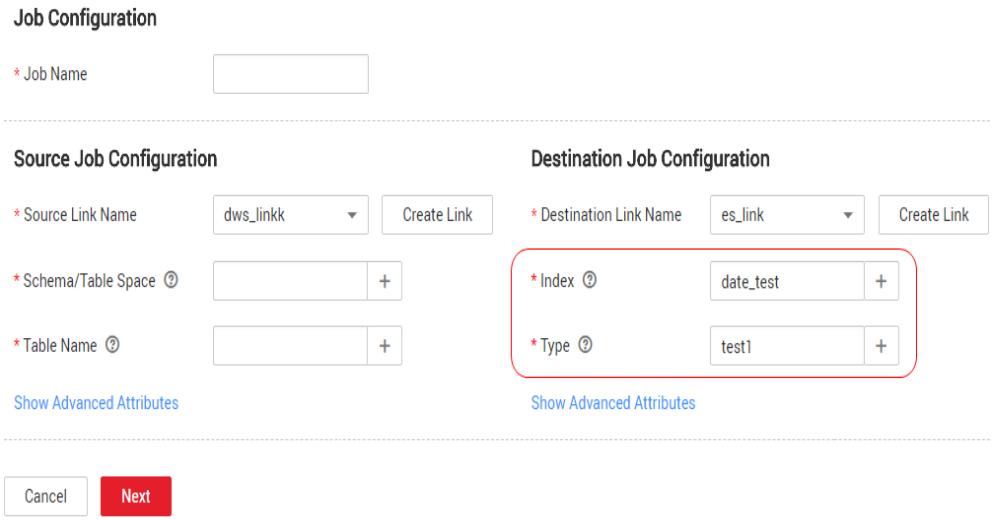
```
java.text.ParseException: Unparseable date: "2018/01/05 15:15:46"
        at java.text.DateFormat.parse(DateFormat.java:366) ~[na:1.8.0_112]
        at org.apache.sqoop.connector.common.DataTypeUtil.convertDateFormat
        at org.apache.sqoop.connector.elasticsearch.ElasticSearchLoader.toJ
        at org.apache.sqoop.connector.elasticsearch.ElasticSearchLoader.arr
7]
        at org.apache.sqoop.connector.elasticsearch.ElasticSearchLoader.loa
```

Possible Cause

CSS has a special processing mechanism on the time field. If the stored time data does not contain the time zone information, Kibana considers the time as the GMT.

In China, the displayed time is eight hours earlier than the actual time. Therefore, when CDM migrates data to CSS, if the index and type are automatically created by CDM (for example, if **date_test** and **test1** of the migration destination highlighted in **Figure 3-2** do not exist in CSS, CDM automatically creates the index and type in CSS), CDM, by default, sets the format of the time field to the standard format of **yyyy-MM-dd HH:mm:ss.SSS Z**, for example, **2018-01-08 08:08:08.666 +0800**.

Figure 3-2 Job configuration

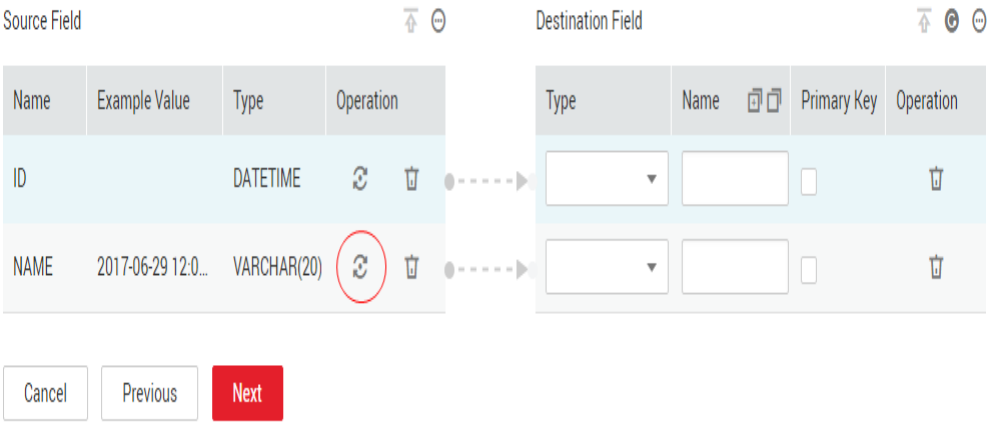


When data is imported from another data source to CSS, if the date format in the source data is not the standard format, for example, **2018/01/05 15:15:46**, the CDM job fails to be executed, and the log shows that the date format cannot be parsed. You need to configure a field converter on CDM to convert the format of the date field to the required format of CSS.

Solution

1. Edit the job and go to the **Map Field** tab page. Click the icon for creating a converter in the row of the source field to create a converter. See **Figure 3-3**.

   Figure 3-3 Creating a converter

2. Select **Expression conversion** as the converter. Currently, expression conversion supports functions of the character string and date types. The syntax is similar to the Java character string and time functions. For details about how to compile the expression, see **Expression Conversion**.
3. In this example, the source time format is *yyyy/MM/dd HH:mm:ss*. To convert the source time format to *yyyy-MM-dd HH:mm:ss.SSS Z*, perform the following operations:

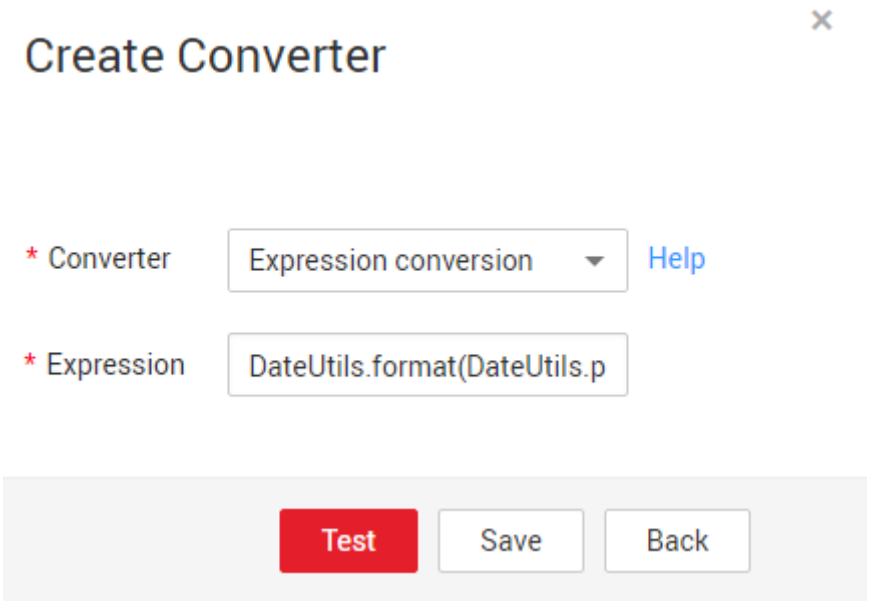
a. Add the time zone information **+0800** to the end of the original date character string. The corresponding expression is **value+" +0800"**.

b. Use the original date format to parse the string to a date object. You can use the DateUtils.parseDate function for parsing. The syntax is **DateUtils.parseDate(String value, String format)**.

c. Format the date object into a character string in target format by using the DateUtils.format function. The syntax is **DateUtils.format(Date date, String format)**.

In this example, the complete expression is **DateUtils.format(DateUtils.parseDate(value+" +0800","yyyy/MM/dd HH:mm:ss Z"),"yyyy-MM-dd HH:mm:ss.SSS Z")**. See **Figure 3-4**.

Figure 3-4 Configuring the expression



4. Save the converter configuration and save and run the job to solve the problem that Cloud Search Service fails to parse the date format.

3.2 What Can I Do If the Map Field Tab Page Cannot Display All Columns?

Symptom

When data is exported from HBase/CloudTable using CDM, fields in the HBase/CloudTable table on the **Map Field** tab page occasionally cannot be displayed completely and cannot match the fields on the migration destination. As a result, the data imported to the migration destination is incomplete.

Possible Cause

HBase/CloudTable are schema-less, and the number of columns in each data is not fixed. On the **Map Field** page, there is a high probability that all columns

cannot be obtained by obtaining example values. In this case, the data on the migration destination is incomplete after the job is executed.

To solve this problem, perform any of the following methods:

1. Add fields on the **Map Field** tab page.
2. Edit the JSON file of the job on the **Job Management** page (modify the **fromJobConfig.columns** and **toJobConfig.columnList** parameters).
3. Export the JSON file of the job to the local PC, modify the parameters in the JSON file (the principle is the same to that in **2**), and then import the JSON file back to CDM.

You are advised to perform **1**. The following uses data migration from HBase to DWS as an example.

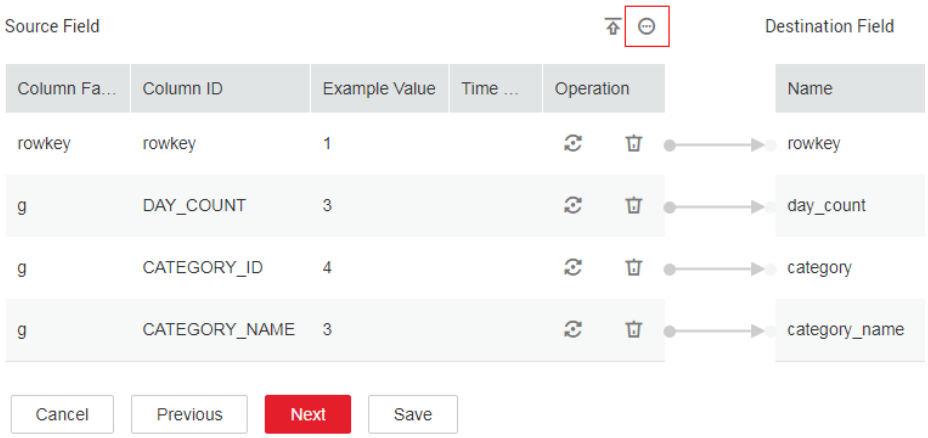
Solution 1: Adding Fields on the Map Field Tab Page

1. Obtain all fields in the tables to be migrated from source HBase. Use colons (:) to separate column families and columns. The following gives an example:

```
rowkey:rowkey
g:DAY_COUNT
g:CATEGORY_ID
g:CATEGORY_NAME
g:FIND_TIME
g:UPLOAD_PEOPLE
g:ID
g:INFOMATION_ID
g:TITLE
g:COORDINATE_X
g:COORDINATE_Y
g:COORDINATE_Z
g:CONTENT
g:IMAGES
g:STATE
```

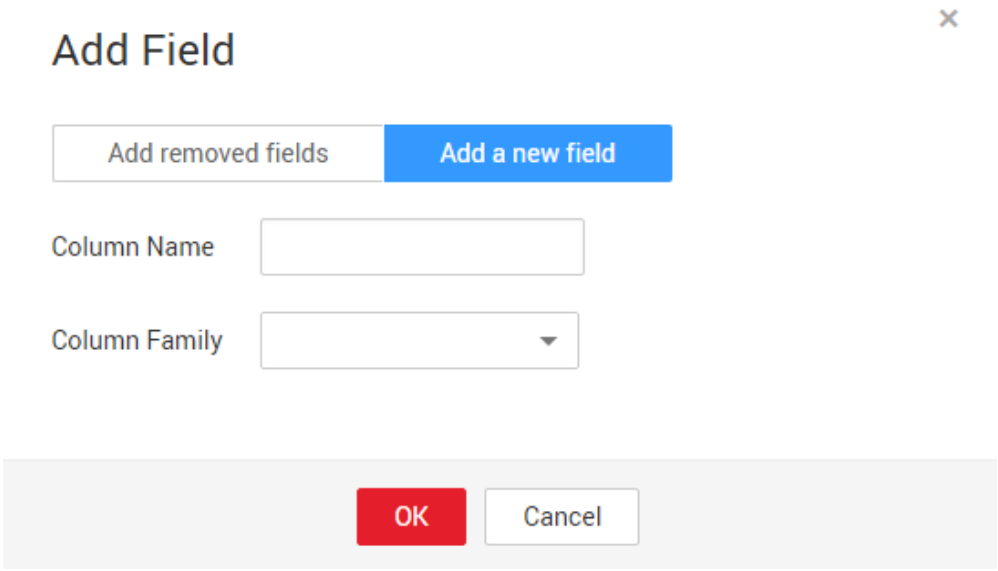
2. On the **Job Management** page, locate the job for exporting data from HBase to DWS, click **Edit** in the row where the job resides, and go to the **Map Field** tab page. See **Figure 3-5**.

Figure 3-5 Field mapping 03



3. Click . In the dialog box that is displayed, select **Add a new field**. See **Figure 3-6**.

Figure 3-6 Adding a field 04



NOTE

- After a field is added, the example value of the new field is not displayed on the console. This does not affect the transmission of field values. CDM directly writes the field values to the migration destination.
- To add new fields, the migration source must be MongoDB, HBase, relational databases, or Redis (data in Redis must be in the Hash format).

4. After all fields are added, check whether the mapping between the migration source and destination is correct. If the mapping is incorrect, drag the fields to adjust the field mapping.
5. Click **Next** and **Save**.

Solution 2: Modifying a JSON File

1. Obtain all fields in the tables to be migrated from source HBase. Use colons (:) to separate column families and columns. The following gives an example:

```
rowkey:rowkey
g:DAY_COUNT
g:CATEGORY_ID
g:CATEGORY_NAME
g:FIND_TIME
g:UPLOAD_PEOPLE
g:ID
g:INFOMATION_ID
g:TITLE
g:COORDINATE_X
g:COORDINATE_Y
g:COORDINATE_Z
g:CONTENT
g:IMAGES
g:STATE
```

2. In the DWS destination table, obtain the fields corresponding to the HBase table fields.

   If any field name corresponding to the HBase field does not exist in the DWS destination table, add it to the DWS table schema. Suppose that the fields in the DWS table are complete and are displayed as follows:


```
rowkey
day_count
category
category_name
find_time
upload_people
id
infomation_id
title
coordinate_x
coordinate_y
coordinate_z
content
images
state
```

3. On the **Job Management** page, locate the job for exporting data from HBase to DWS, and choose **More** > **Edit Job JSON** in the row where the job resides.
4. On the page that is displayed, edit the JSON file of the job.
   a. Modify the **fromJobConfig.columns** parameter of the migration source to the HBase fields obtained in **1**. Use & to separate column numbers and colons (:) to separate column families and columns. The following gives an example:

```
"from-config-values": {
            "configs": [
               {
                  "inputs": [
                     {
                        "name": "fromJobConfig.table",
                        "value": "HBase"
                     },
                     {
                        "name": "fromJobConfig.columns",
                        "value":
"rowkey:rowkey&g:DAY_COUNT&g:CATEGORY_ID&g:CATEGORY_NAME&g:FIND_TIME&g:UPLOAD_PEOPLE&g:ID&g:INFOMATION_ID&g:TITLE&g:COORDINATE_X&g:COORDINATE_Y&g:COORDINATE_Z&g:CONTENT&g:IMAGES&g:STATE"
                     },
                     {
                        "name": "fromJobConfig.formats",
                        "value": {
                           "2": "yyyy-MM-dd",
                           "undefined": "yyyy-MM-dd"
                        }
                     }
                  ],
                  "name": "fromJobConfig"
               }
            ]
         }
```

   b. Modify the **toJobConfig.columnList** parameter of the migration source to the field list of DWS obtained in **2**.

      The sequence must be the same as that of HBase to ensure correct field mapping. Use & to separate field names. The following gives an example:

```
"to-config-values": {
            "configs": [
               {
                  "inputs": [
                     {
                        "name": "toJobConfig.schemaName",
                        "value": "dbadmin"
                     },
                     {
                        "name": "toJobConfig.tablePreparation",
                        "value": "DO_NOTHING"
```



```
                    },
                    {
                        "name": "toJobConfig.tableName",
                        "value": "DWS "
                    },
                    {
                        "name": "toJobConfig.columnList",
                        "value":
"rowkey&day_count&category&category_name&find_time&upload_people&id&infomation_
id&title&coordinate_x&coordinate_y&coordinate_z&content&images&state"
                    },
                    {
                        "name": "toJobConfig.shouldClearTable",
                        "value": "true"
                    }
                ],
                "name": "toJobConfig"
            }
        ]
    }
```

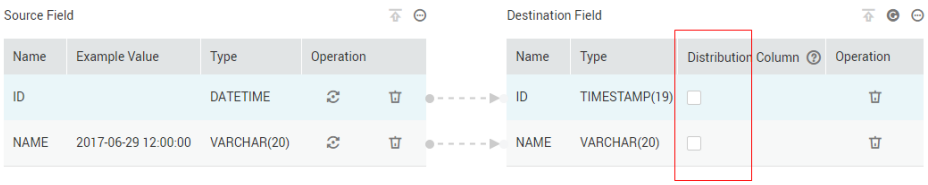
c. Retain the settings of other parameters, and then click **Save and Run**.

5. After the job is completed, check whether the data in the DWS table matches the data in HBase. If the mapping is incorrect, check whether the sequences of the HBase and DWS fields in the JSON file are the same.

3.3 How Do I Select Distribution Columns When Using CDM to Migrate Data to DWS?

When using CDM to migrate data to DWS or FusionInsight LibrA and create a table on DWS, select the distribution columns on the **Map Field** tab page. See **Figure 3-7**.

Figure 3-7 Selecting distribution columns



Selecting the distribution column is very important for the running of DWS/FusionInsight LibrA. When migrating data to DWS/FusionInsight LibrA, you are advised to specify the distribution column according to the following principles:

1. Use the primary key as the distribution column.
2. If multiple data segments are combined as primary keys, specify all primary keys as the distribution column.
3. In the scenario where no primary key is available, if no distribution column is selected, DWS uses the first column as the distribution column by default. As a result, data skew risks exist.

Therefore, when a single table or entire database is imported to DWS/FusionInsight LibrA, you are advised to manually select a distribution column; otherwise, CDM automatically selects one. For more information about the distribution column, see **Selecting a Distribution Column**.

If the DWS primary key or table contains only one field, the field type must be a common character string, value, or date. When data is migrated from another database to DWS, if automatic table creation is selected, the primary key must be of the following types. If no primary key is set, at least one of the following fields must be set. Otherwise, the table cannot be created and the CDM job fails.

- INTEGER TYPES: TINYINT, SMALLINT, INT, BIGINT, NUMERIC/DECIMAL
- CHARACTER TYPES: CHAR, BPCHAR, VARCHAR, VARCHAR2, NVARCHAR2, TEXT
- DATA/TIME TYPES: DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, INTERVAL, SMALLDATETIME

3.4 What Do I Do If the Error Message "value too long for type character varying" Is Displayed When I Migrate Data to DWS?

Symptom

When you use CDM to migrate data to DWS/FusionInsight LibrA, the migration fails and the error message "**value too long for type character varying**" is displayed in the log. See **Figure 3-8**.

Figure 3-8 Log output

```
Caused by: org.postgresql.util.PSQLException: ERROR: value too long for type character varying(50)
  Where: COPY fl_behavior_module, line 72, column MODULE_NAME: "[redacted]"
        at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2477)
        at org.postgresql.core.v3.QueryExecutorImpl.processCopyResults(QueryExecutorImpl.java:1107)
        at org.postgresql.core.v3.QueryExecutorImpl.writeToCopy(QueryExecutorImpl.java:989)
        at org.postgresql.core.v3.CopyInImpl.writeToCopy(CopyInImpl.java:35)
        ... 16 common frames omitted
```

Possible Cause

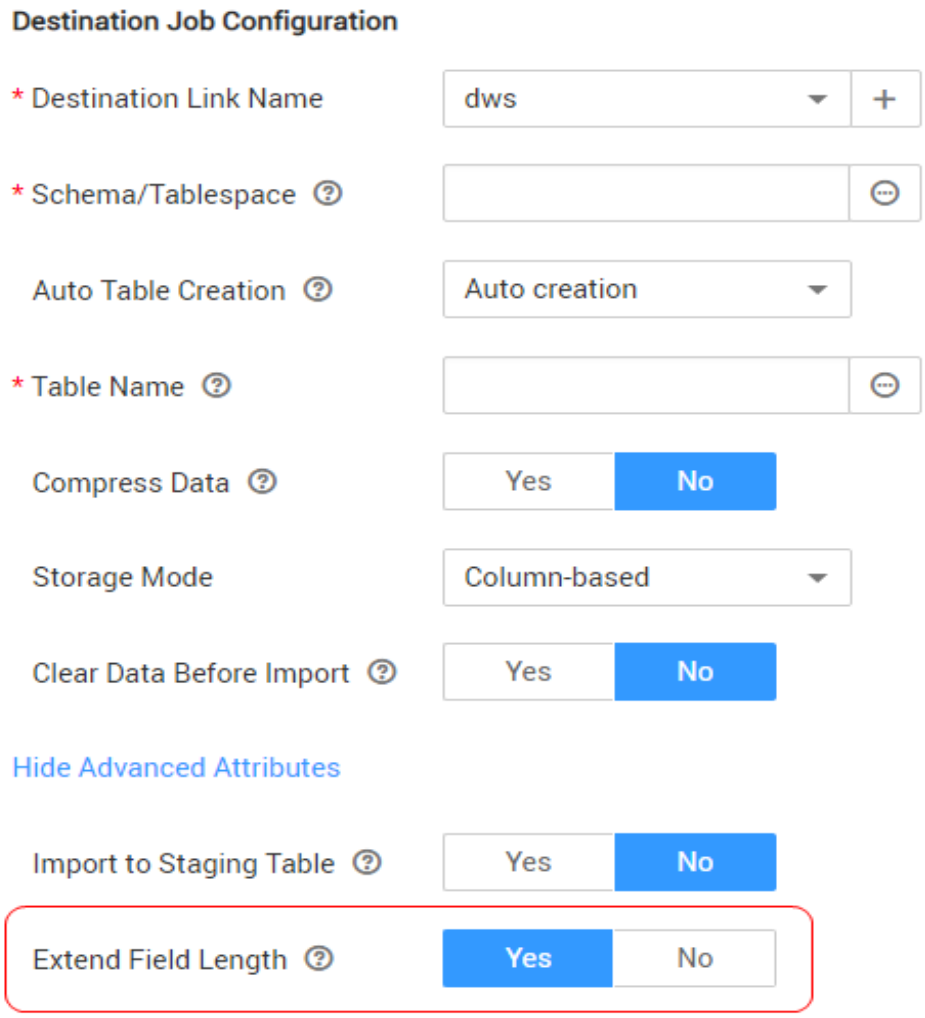
The data migrated to DWS is in Chinese, and the table is automatically created at the migration destination. The length of the varchar field of DWS is calculated by byte, and a Chinese character may occupy three bytes in UTF-8 encoding. If the length of a Chinese character exceeds that of the varchar field of DWS, an error occurs and the error message "**value too long for type character varying**" is displayed.

Solution

To solve this problem, you can select **Extend Field Length** to **Yes**, so that the length of the **varchar** field is automatically increased by three times when the destination table is created.

Edit the table/file migration job on CDM. In **Destination Job Configuration**, set **Auto Table Creation** to **Auto creation**, **Extend Field Length** is displayed in **Show Advanced Attributes**. Set **Extend Field Length** to **Yes**. See **Figure 3-9**.

Figure 3-9 Extending field length



3.5 What Can I Do If Error Message "Unable to execute the SQL statement" Is Displayed When I Import Data from OBS to SQL Server?

Symptom

When CDM is used to import data from OBS to SQL Server, the job fails to be executed and error message "Unable to execute the SQL statement. Cause: "String or binary data truncated" is displayed.

Possible Cause

The data in OBS exceeds the length limit of the SQL Server database.

Solution

When creating a table in the SQL Server database, increase the length of the database field. The length of the database field must be greater than that of the data in OBS.

