ALIBABA CLOUD

Alibaba Cloud

Apsara Stack Enterprise

Realtime Compute

Developer Guide

Product Version: v3.16.2

Document Version: 20220915

Alibaba Cloud

Legal disclaimer

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

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

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

Table of Contents

1.ASAPI Developer Guide

1.1. Preparations

1.1.1. Log on to the API Tool console

You can log on to the API Tool console from the Apsara Uni-manager Management Console.

Prerequisites

- Before you log on to the Apsara Uni-manager Management Console, the endpoint of the console is obtained from the deployment staff.
- We recommend that you use Google Chrome.

Procedure

1. Enter the URL of the Apsara Uni-manager Management Console in the address bar and press the Enter key.
2. Enter your username and password.

   Obtain the username and password that you use to log on to the Apsara Uni-manager Management Console from the operations administrator.

   > **Note** The first time that you log on to the Apsara Uni-manager Management Console, you must change the password of your account. For security purposes, your password must meet the minimum complexity requirements. The password must be 10 to 32 characters in length and must contain at least two of the following character types:
   >
   > - Uppercase or lowercase letters
   > - Digits
   > - Special characters, including exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%)

3. Click **Log On**.
4. If your account has multi-factor authentication (MFA) enabled, perform corresponding operations in the following scenarios:
   - You log on to the Apsara Uni-manager Management Console for the first time after MFA is enabled by the administrator:
     a. On the Bind Virtual MFA Device page, bind an MFA device.
     b. Enter the username and password again as in Step 2 and click **Log On**.
     c. Enter a six-digit MFA verification code and click **Authenticate**.
   - You have enabled MFA and bound an MFA device:

     Enter a six-digit MFA verification code and click **Authenticate**.

     > **Note** For more information, see the *Bind a virtual MFA device to enable MFA* topic in *Apsara Uni-manager Management Console User Guide*.

5. In the top navigation bar, choose **Products > Others > API Tool**.

1.1.2. View information about API operations

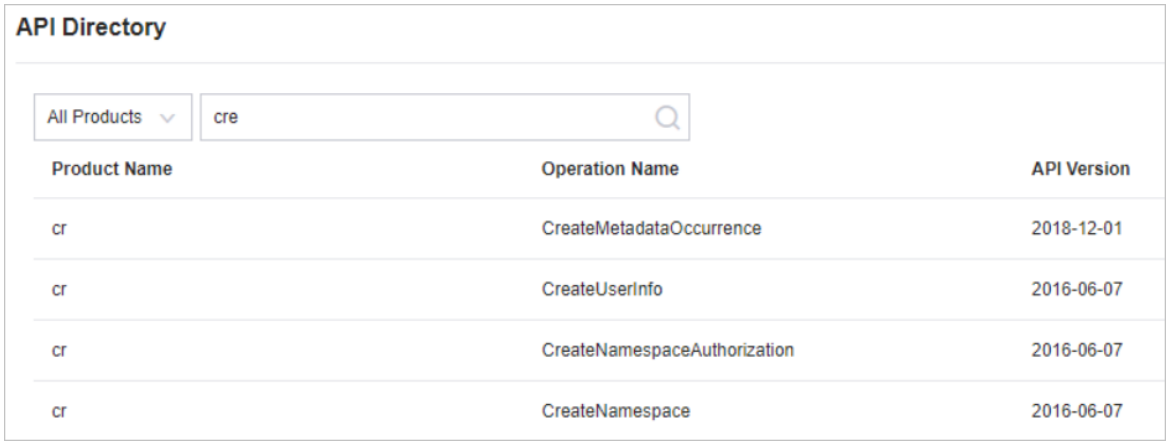
This topic describes how to view information about API operations. Before you use SDKs, obtain service names, API operation names, and API operation versions.

Context

The API operations on the API Directory page are only for reference. For information about how to call API operations, see developer guides of the corresponding cloud services. If the information on the API Directory page is inconsistent with that in the developer guide of the corresponding cloud service, the information in the developer guide prevails.

Procedure

1. Go to the API Tool console. For more information, see Log on to the API Tool console.
2. In the left-side navigation pane, click **API Directory**.
3. On the **API Directory** page, select the desired service from the drop-down list, enter a keyword of an API operation name in the search box, and then click the search icon.



4. Then, you can view the service name, API operation name, and API operation version.

1.1.3. Obtain an AccessKey pair

An AccessKey pair consists of an AccessKey ID and an AccessKey secret. AccessKey pairs are used to implement symmetric encryption to verify the identity of requesters. The AccessKey ID is used to identify a user. The AccessKey secret is used to encrypt a signature string. This topic describes how to obtain an AccessKey pair.

Prerequisites

An account that assumes the operations administrator or level-1 organization administrator role is obtained. Only an operations administrator or a level-1 organization administrator can obtain an AccessKey pair of the organization.

Context

AccessKey supports the following authorization methods:

- Use RAM to authorize the AccessKey ID and AccessKey secret to third-party requesters.

- Use STS to authorize the AccessKey ID, AccessKey secret, and security token to third-party requesters.

This topic describes how to obtain an AccessKey pair authorized by RAM. For more information about how to obtain an AccessKey pair authorized by STS, see *Obtain an STS AccessKey pair*.

We recommend that you use the AccessKey pair of a personal account to call the API operations of the Apsara Uni-manager Management Console and cloud services. If you use the AccessKey pair of a personal account, configure the request headers that are described in the following table for access control. Otherwise, an error that indicates insufficient permissions can occur.

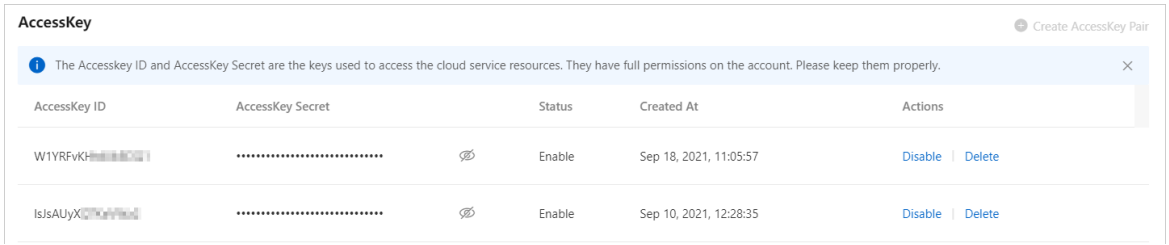
Parameter	Description
x-acs-regionid	The ID of the region, such as cn-hangzhou-*.
x-acs-organizationid	The ID of the organization in the Apsara Uni-manager Management Console. For information about how to obtain an organization ID, see the "GetOrganizationList" section in Apsara Uni-manager Management Console Developer Guide.
x-acs-resourcegroupid	The ID of the resource set in the Apsara Uni-manager Management Console. For information about how to obtain a resource set ID, see the "ListResourceGroup" section in Apsara Uni-manager Management Console Developer Guide.
x-acs-instanceid	The ID of the instance on which to perform operations. For example, you can query the ID of an Elastic Compute Service (ECS) instance from the ECS instance list.

Warning The AccessKey pair of a personal account is managed and controlled by the permission system of the Apsara Uni-manage Management Console. If you need to obtain the AccessKey pair of an account that has permissions on an organization, make sure that your operation is allowed by an administrator. This ensures security. The account that has permissions on an organization is a privileged account.

Obtain the AccessKey pair of a personal account

To obtain the AccessKey pair of a personal account, perform the following steps:

1. Log on to the Apsara Uni-manager Management Console.
2. In the upper-right corner of the homepage, move the pointer over the profile picture and choose **User Information** from the shortcut menu.
3. In the **Apsara Stack AccessKey Pair** section, obtain the AccessKey pair.



> **Note** The AccessKey pair consists of an AccessKey ID and an AccessKey secret. These credentials provide you with full permissions on Apsara Stack resources. You must keep the AccessKey pair confidential.

Obtain the AccessKey pair of an organization

To obtain the AccessKey pair of an organization, perform the following steps:

1. Log on to the Apsara Uni-manager Management Console as an administrator.
2. In the top navigation bar, click **Enterprise**.
3. In the left-side navigation pane, choose **Resources > Organizations**.
4. In the organization navigation tree, click the name of the level-1 organization for which you want to obtain AccessKey pairs.
5. Click **Management AccessKey** on the right side of the page.
6. In the AccessKey message, view the AccessKey pair of the organization.

1.1.4. Obtain common header parameters

You must provide multiple header parameters when you call API operations. This topic describes how to obtain common header parameters. Some response parameters are also explained.

Query a region ID

1. Log on to the Apsara Uni-manager Management Console as an administrator.
2. In the top navigation bar, click **Enterprise**.
3. In the left-side navigation pane, choose **Resources** > **Regions**.
4. Click the name of the organization that you want to view.
5. View the region ID in the **Regions** window on the right.

Obtain an organization ID

You can call the API operation to obtain the organization ID. For more information about how to obtain the organization ID, see *GetOrganizationList* in *Apsara Uni-manager Management Console Developer Guide*.

Obtain a resource set ID

You can call the API operation to obtain the resource set. For more information about how to obtain the resource set ID, see *ListResourceGroup* in *Apsara Uni-manager Management Console Developer Guide*.

Obtain an instance ID

You can obtain the ID of an instance in the cloud service console of the Apsara Uni-manager Management Console. This topic illustrates how to obtain the ID of an ECS instance. Perform the following steps:

1. Log on to the Apsara Uni-manager Management Console.
2. In the top navigation bar, choose **Products** > **Elastic Computing** > **Elastic Compute Service**.

3. In the left-side navigation pane, choose **Instances & Images > Instances**.
4. View the **instance ID** in the instance list.

Response parameters

The following response parameter are returned by default when you call list-type API operations for cloud product resource instances.

Parameter	Description
ResourceGroupName	The name of the resource set.
ResourceGroup	The ID of the resource set.
DepartmentName	The name of the organization.
Department	The ID of the organization.

1.1.5. Obtain an STS AccessKey pair

This topic describes how to use STS to make API calls.

Background information

Alibaba Cloud Security Token Service (STS) allows you to manage temporary credentials to your Alibaba Cloud resources. You can use STS to make API calls.

Procedure

1. Obtain the `RAM role`.
   i. Log on to the Apsara Uni-manager Management Console.
   ii. Move the pointer over the profile picture in the upper-right corner of the page and click **Personal Information**.
   iii. Click **View Current Role Policy**.

   In the **View Policies of Current Role** dialog box, view your `RAM role`.
2. Use the following code to obtain `stsAK`, `stsSK`, and `stsToken` by using your `RAM role`.

```
public static void main(String[] args) {
     ASClient asClient = new ASClient();
     // The call source.
        asClient.setSdkSource("asapi_test");
     // Pass in the ARN.
        String assumeRole = AssumeRole(asClient, "<RAM Role>");

        JSONObject assumeRoleJson = JSONObject.parseObject(assumeRole);

        JSONObject credentials = assumeRoleJson.getJSONObject("Credentials");

        // Obtain the STS triplet.
        String stsAK = credentials.getString("AccessKeyId");
        String stsSK = credentials.getString("AccessKeySecret");
        String stsToken = credentials.getString("SecurityToken");
}

public static String AssumeRole(ASClient asClient, String roleArn) {

        Map<String, String> headers = new HashMap<>();

        Map<String, Object> reqParams = new HashMap<>();

        reqParams.put("action", "AssumeRole");

        reqParams.put("Product", "Sts");

        reqParams.put("Version", "2015-04-01");

        reqParams.put("regionId", regionId);

     // Your AccessKey ID.
        reqParams.put("AccessKeyId", <Your AccessKey ID>);
     // Your AccessKey secret.
        reqParams.put("AccessKeySecret", <Your AccessKey secret>);

        reqParams.put("roleSessionName", "AsapiStsUser");

        reqParams.put("roleArn", roleArn);

        return asClient.doRequest(endpoint, headers, reqParams);
}
```

3. You can use STS to make API calls. The following sample code demonstrates how to call ECS API operations.

```
public static void main(String[] args) {
     ASClient asClient = new ASClient();
        asClient.setSdkSource("asapi_test");
  // Call ECS API operations.
        String ecsRes = DescribeZones(asClient,<stsAK>,<stsSK>,<stsToken>);
}


public static String DescribeZones(ASClient asClient, String stsAK, String stsSK, Strin
g stsToken) {

        Map<String, String> headers = new HashMap<>();

        Map<String, Object> reqParams = new HashMap<>();

        reqParams.put("action", "DescribeZones");

        reqParams.put("Product", "Ecs");

        reqParams.put("Version", "2014-05-26");

        reqParams.put("regionId", "<your-RegionID>");

        // The STS triplet of AccessKey ID, AccessKey secret, and security token.
        reqParams.put("AccessKeyId", <stsAK>);
        reqParams.put("AccessKeySecret", <stsSK>);
        reqParams.put("SecurityToken", <stsToken>);

        return asClient.doRequest(endpoint, headers, reqParams, HttpMethod.GET);

    }
```

1.2. Use POP to make API calls

1.2.1. Preparations

Like in the public cloud, you can use the POP gateway to call API operations in Apsara Stack. This topic describes the preparations before you use the POP gateway to call API operations.

Background information

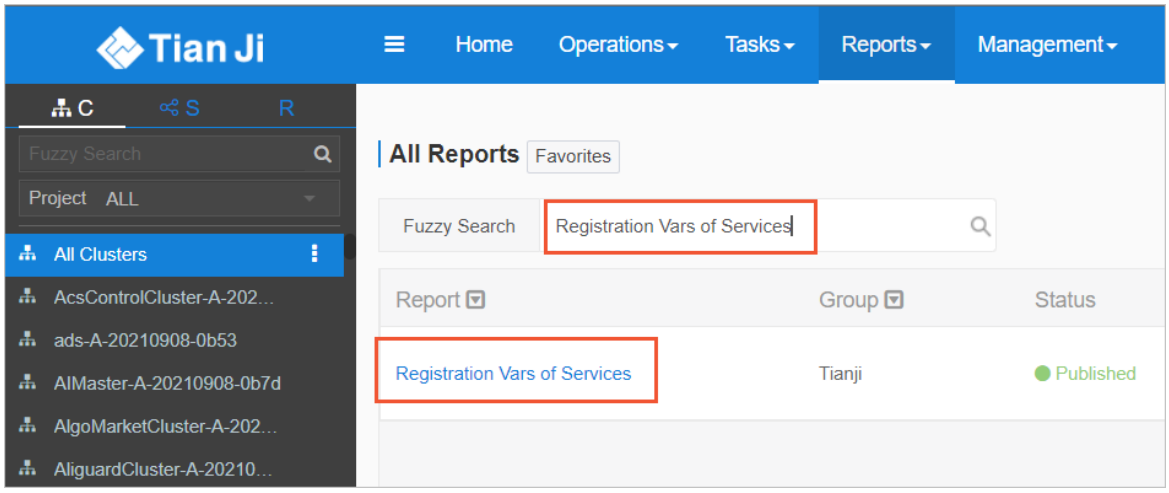
In Apsara Stack, you can use both ASAPI and POP gateways to call API operations. To use the same method and produce the same user experience as in the public cloud, we recommend that you use the POP gateway to call API operations in Apsara Stack.

Obtain service endpoints

To use the POP gateway to call API operations, you must use the service endpoints. You can obtain the service endpoints from Alibaba Cloud onsite O&M engineers.

You can also obtain the service endpoints by yourself. The following example describes how to obtain the ECS endpoints.

1. Log on to Apsara Infrastructure Management Framework.
   i. Log on to the Apsara Uni-manager Operations Console.
   ii. In the top navigation bar, click **O&M**.
   iii. In the left-side navigation pane, choose **Product Management** > **Products**.
   iv. In the Apsara Stack O&M section, click **Apsara Infrastructure Management Framework**.
2. In the left-side navigation pane, click **Reports**.
3. On the **All Reports** page, search for `Registration Vars of Services` and click the report name.



4. On the **Registration Vars of Services** page, click the icon next to **Service** and search for `ECS`.
5. Find the `ecs-yaochi` service, right-click the **Service Registration** column, and then select **Show More** from the shortcut menu.

   On the **Details** page, view the **internal.openapi.endpoint** value, which is the `endpoint`.

Obtain common parameters

You must obtain the parameters required when you use the POP gateway to call API operations, such as the organization ID, resource set ID, region ID, and instance ID. For more information, see Obtain common header parameters.

1.2.2. SDK for Java

Before you use POP SDK for Java to make API calls, you must first understand the RPC and ROA styles.

Prerequisites

- An Alibaba Cloud SDK package is downloaded for the service that you want to access. The required dependencies are installed.
- SSL certificate validation is ignored for requests over HTTPS.

Call examples

The following sample code provides an example on how to use the RPC style to make an API call. In this example, the DescribeInstances operation of Elastic Compute Service (ECS) is called, and the API version is 2014-05-26.

Sample code:

```
import com.aliyuncs.CommonRequest;
import com.aliyuncs.CommonResponse;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.profile.DefaultProfile;

public class DescribeInstances {

    public static void main(String[] args) {
        // Create and initialize a DefaultAcsClient instance.
        DefaultProfile profile = DefaultProfile.getProfile("<your-regionid>", "<yourAccessK
eyId>", "<yourAccessSecret>");

        // Ignore SSL certificate validation.
        // HttpClientConfig clientConfig = HttpClientConfig.getDefault();
        // clientConfig.setIgnoreSSLCerts(true);
        // Configure the time-out period for a connection.
        // clientConfig.setConnectionTimeoutMillis(3000L);
        // Set the read timeout period.
        // clientConfig.setReadTimeoutMillis(10000L);
        // profile.setHttpClientConfig(clientConfig);

        IAcsClient client = new DefaultAcsClient(profile);
        // Create an API request and configure the parameters.
        CommonRequest request = new CommonRequest();
        // Specify the request method.
        request.setSysMethod(MethodType.POST);
        // Set the service endpoint.
        request.setSysDomain("<ecsEndpoint>");
        // Specify the version of the API operation.
        request.setSysVersion("2014-05-26");
        // Specify the name of the service that you want to access and the region where the
service resides.
        request.setSysLocationProduct("Ecs");
        // Specify the name of the API operation.
        request.setSysAction("DescribeInstances");

        // When you use the STS AccessKey ID to call RPC API operations, you must add the S
ecurityToken parameter to the header.
        // request.putBodyParameter("SecurityToken", "<your-SecurityToken>");

        // Configure operation-specific parameters.
        request.putBodyParameter("PageSize", "10");
        request.putBodyParameter("PageNumber", "1");

        // Required. The call source.
```

```
        request.putHeadParameter("x-acs-caller-sdk-source", "ApiTestDemo");
        // Optional. For the historical versions before Enterprise Edition V3.15.0.
        request.putHeadParameter("x-acs-regionid", "<your-regionid>");
        // Optional. The resource set ID.
        request.putHeadParameter("x-acs-resourcegroupid", "<your-resourcegroupid>");
        // Required. The organization ID.
        request.putHeadParameter("x-acs-organizationid", "<your-organizationid>");
        // Optional. For the modify and delete operations.
        // request.putHeadParameter("x-acs-instanceid", "<your-instanceid>");

        try {
            // Initiate the request and handle the response or exceptions.
            CommonResponse response = client.getCommonResponse(request);
            System.out.println(response.getData());
        } catch (ServerException e) {
            e.printStackTrace();
        } catch (ClientException e) {
            System.out.println("ErrCode:" + e.getErrCode());
            System.out.println("ErrMsg:" + e.getErrMsg());
            System.out.println("RequestId:" + e.getRequestId());
        }

    }
}
```

The following sample code provides an example on how to use the ROA style to make an API call. In this example, the GetNamespaceList operation of Container Registry is called, and the API version is 2016-06-07.

Sample code:

```
import com.aliyuncs.CommonRequest;
import com.aliyuncs.CommonResponse;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.profile.DefaultProfile;

public class GetNamespaceList {

    public static void main(String[] args) {
        // Create and initialize a DefaultAcsClient instance.
        DefaultProfile profile = DefaultProfile.getProfile("<your-regionid>", "<yourAccessKeyId>", "<yourAccessSecret>");

        // Ignore SSL certificate validation.
        // HttpClientConfig clientConfig = HttpClientConfig.getDefault();
        // clientConfig.setIgnoreSSLCerts(true);
        // Configure the time-out period for a connection.
        // clientConfig.setConnectionTimeoutMillis(3000L);
        // Set the read timeout period.
        // clientConfig.setReadTimeoutMillis(10000L);
```

```
        // profile.setHttpClientConfig(clientConfig);

        IAcsClient client = new DefaultAcsClient(profile);
        // Create an API request and configure the parameters.
        CommonRequest request = new CommonRequest();

        // Set the request protocol to HTTPS.
        request.setSysProtocol(ProtocolType.HTTP);
        // Specify the request method.
        request.setSysMethod(MethodType.GET);
        // Set the service endpoint.
        request.setSysDomain("<crEndpoint>");
        // Specify the version of the API operation.
        request.setSysVersion("2016-06-07");
        // Configure the uriPattern parameter, which is required for the ROA style.
        request.setSysUriPattern("/namespace");
        // Specify the name of the service that you want to access and the region where the
service resides.
        request.setSysLocationProduct("cr");
        // Specify the name of the API operation.
        request.setSysAction("GetNamespaceList");

        // Required. The call source.
        request.putHeadParameter("x-acs-caller-sdk-source", "ApiTestDemo");
        // Optional. For the historical versions before Enterprise Edition V3.15.0.
        request.putHeadParameter("x-acs-regionid", "<your-regionid>");
        // Optional. The resource set ID.
        request.putHeadParameter("x-acs-resourcegroupid", "<your-resourcegroupid>");
        // Required. The organization ID.
        request.putHeadParameter("x-acs-organizationid", "<your-organizationid>");
        // Optional. For the modify and delete operations.
        // request.putHeadParameter("x-acs-instanceid", "<your-instanceid>");


        // When you use the STS AccessKey ID to call ROA API operations, you must add the x
-acs-security-token parameter to the header.
        // request.putHeadParameter("x-acs-security-token", "<your-securitytoken>");

        try {
            // Initiate the request and handle the response or exceptions.
            CommonResponse response = client.getCommonResponse(request);
            System.out.println(response.getData());
        } catch (ServerException e) {
            e.printStackTrace();
        } catch (ClientException e) {
            System.out.println("ErrCode:" + e.getErrCode());
            System.out.println("ErrMsg:" + e.getErrMsg());
            System.out.println("RequestId:" + e.getRequestId());
        }

    }
}
```

> **Notice**
>
> If you send a request over HTTPS, you must enable the request to ignore SSL certificate validation.

1.2.3. SDK for Go

This topic describes how to use POP SDK for Go.

Prerequisites

An Alibaba Cloud SDK package is downloaded for the service that you want to access. The required dependencies are installed.

Background information

Both RPC and ROA styles are supported for API calls.

Call examples

The following sample code provides an example on how to use the RPC style to make an API call. In this example, the DescribeInstances operation of Elastic Compute Service (ECS) is called, and the API version is 2014-05-26.

Sample code:

```
package main

import (
    "fmt"
    "github.com/aliyun/alibaba-cloud-sdk-go/sdk"
    "github.com/aliyun/alibaba-cloud-sdk-go/sdk/requests"

)

func main() {
    client, err := sdk.NewClientWithAccessKey("<your-RegionID>", "<AccessKey ID>", "<Access
Key Secret>")

    request := requests.NewCommonRequest()

    request.Method = "POST"
    request.Domain = "<ECSEndpoint>"
    request.Version = "2014-05-26"
    request.ApiName = "DescribeInstances"
    request.Product = "Ecs"
    // When you use the STS AccessKey ID to call RPC API operations, you must add the Secur
ityToken parameter to the header.
    // request.QueryParams["SecurityToken"]="<your-SecurityToken>"

    // Required. The call source.
    request.Headers["x-acs-caller-sdk-source"] = "ApiTestDemo"
    // Optional. For the historical versions before Enterprise Edition V3.15.0.
    request.Headers["x-acs-regionid"] = "<your-RegionID>"
    // Optional. The resource set ID.
    request.Headers["x-acs-resourcegroupid"] = "<your-ResourceGroupID>"
    // Required. The organization ID.
    request.Headers["x-acs-organizationid"] = "<your-OrganizationID>"
    // Optional. For the modify and delete operations.
    // request.Headers["x-acs-instanceid"] = "<your-InstanceID>"

    response, err := client.ProcessCommonRequest(request)
    if err != nil {
        panic(err)
    }
    fmt.Print(response.GetHttpContentString())
}
```

The following sample code provides an example on how to use the ROA style to make an API call. In this example, the GetNamespaceList operation of Container Registry is called, and the API version is 2016-06-07.

The following sample code shows how to call API operations:

```
package main

import (
 "fmt"
 "github.com/aliyun/alibaba-cloud-sdk-go/sdk"
 "github.com/aliyun/alibaba-cloud-sdk-go/sdk/requests"
 "os"
)



func main() {
    client, err := sdk.NewClientWithAccessKey("<your-regionid>", "<yourAccessKeyId>", "<you
rAccessSecret>")
 request := requests.NewCommonRequest()

 request.Method = "GET"
 request.Domain = "<crEndpoint>"
 request.Version = "2016-06-07"
 request.ApiName = "GetNamespaceList"
 request.Product = "cr"

 request.PathPattern = "/namespace"
 // When you use the STS AccessKey ID to call ROA API operations, you must add the x-acs-se
curity-token parameter to the header.
 // request.Headers["x-acs-security-token"] = "<your-SecurityToken>"
 // Required. The call source.
 request.Headers["x-acs-caller-sdk-source"] = "ApiTestDemo"
    // Optional. For the historical versions before Enterprise Edition V3.15.0.
    request.Headers["x-acs-regionid"] = "<your-RegionID>"
    // Optional. The resource set ID.
    request.Headers["x-acs-resourcegroupid"] = "<your-ResourceGroupID>"
    // Required. The organization ID.
    request.Headers["x-acs-organizationid"] = "<your-OrganizationID>"
 // Optional. For the modify and delete operations.
 // request.Headers["x-acs-instanceid"] = "<your-InstanceID>"

 response, err := client.ProcessCommonRequest(request)
 if err != nil {
  panic(err)
 }
 fmt.Print(response.GetHttpContentString())
}
```

Configure proxies

You can configure proxies in one of the following ways when you call API operations:

- Configure proxies as environment variables.

  Set the HTTP_PROXY and HTTPS_PROXY environment variables, or set the NO_PROXY environment variable.

  Sample code:

```
import (
    "os"
)
func main(){
 os.Setenv("HTTP_PROXY", "http://x.x.x.x:xxxx") // Configure the HTTP_PROXY environment variable.
 os.Setenv("HTTPS_PROXY", "https://x.x.x.x:xxxx") // Configure the HTTPS_PROXY environment variable.
}
```

- Configure proxies on the client.

  Sample code:

```
// The proxies that are configured on the client take precedence over those that are configured by using environment variables.
client.SetHttpProxy("http://x.x.x.x:xxxx")    // Configure the HTTP proxy.
client.GetHttpProxy()                           // Query the HTTP proxy.

client.SetHttpsProxy("https://x.x.x.x:xxxx")    // Configure the HTTPS proxy.
client.GetHttpsProxy()                            // Query the HTTPS proxy.

client.SetNoProxy("127.0.0.1,localhost")      // Configure the proxy whitelist.
client.GetNoProxy()                           // Query the proxy whitelist.
```

1.2.4. SDK for Python

This topic describes how to use POP SDK for Python.

Prerequisites

An Alibaba Cloud SDK package is downloaded for the service that you want to access. The required dependencies are installed.

Background information

Only the remote procedure call (RPC) style is supported for API calls.

Call examples

The following sample code provides an example on how to make an API call. In this example, the DescribeInstances operation of Elastic Compute Service (ECS) is called, and the API version is 2014-05-26.

Sample code:

```
from aliyunsdkcore.client import AcsClient
import os
from aliyunsdkcore.request import CommonRequest

# Create an AcsClient object.
client = AcsClient(
    "<AccessKey ID>",
    "<AccessKey Secret>",
    "<your-RegionID>",
    timeout=1000  # Set the timeout period.
```

```
    timeout=1000,  # Set the timeout period.
    verify=False, # Ignore certificate validation. or verify='<The location where the certificate is stored.>'
);
# Set the following parameters for the HTTPS request protocol:
# client.set_verify(False)
# os.environ["HTTPS_PROXY"] = "http://127.0.0.1:8080"

# os.environ["HTTP_PROXY"] = "http://127.0.0.1:8080"
def hello_ecs_instance():
    # Create a request and set the parameters.
    request = CommonRequest()

    # When you use the STS AccessKey ID to call RPC API operations, you must add the SecurityToken parameter to the header.
    # params = {"PageSize":"10","PageNumber":"1","SecurityToken":"<your-SecurityToken>"};
    params = {"PageSize":"10","PageNumber":"1"};
    request.set_body_params(params)

    headers = {
        # Required. The call source.
        "x-acs-caller-sdk-source":"ApiTestDemo",
        # Optional. For the historical versions before Enterprise Edition V3.15.0.
        "x-acs-regionid":"<your-RegionID>",
        # Optional. The resource set ID.
        "x-acs-resourcegroupid":"<your-ResourceGroupID>",
        # Required. The organization ID.
        "x-acs-organizationid":"<your-OrganizationID>"
        # Optional. For the modify and delete operations.
        # "x-acs-instanceid":"<your-InstanceID>"

    }
    request.set_headers(headers)
    request.set_version("2014-05-26")
    request.set_action_name("DescribeInstances")
    request.set_endpoint("<ecsEndpoint>") // Replace ecsEndpoint with the real endpoint.
    request.set_product("Ecs")
    request.set_method("POST")

    # Initiate the API request and obtain the response.
    response = client.do_action_with_exception(request)
    print(response)


if __name__ == '__main__':
    print("Hello Aliyun OpenApi!")
    hello_ecs_instance()
```

Configure proxies

You can configure HTTP_PROXY and HTTPS_PROXY environment variables when you call API operations.

Sample code:

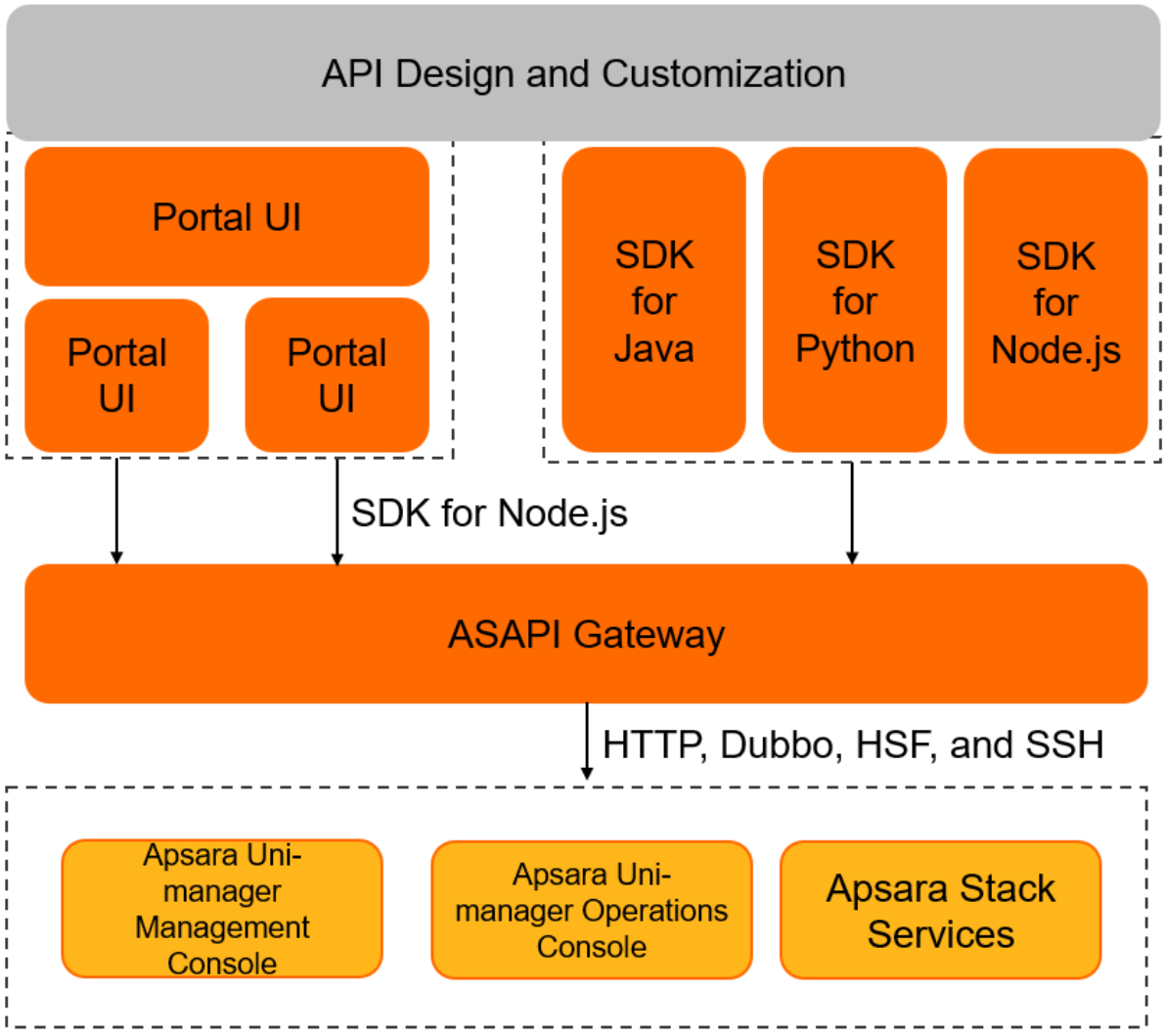
```
# Configure the HTTPS proxy.
os.environ["HTTPS_PROXY"] = "http://x.x.x.x:xxxx"

# Configure the HTTP proxy.
os.environ["HTTP_PROXY"] = "https://x.x.x.x:xxxx"
```

1.3. Use ASAPI to make API calls

1.3.1. ASAPI overview

This topic describes the architecture and features of the Apsara Stack API (ASAPI).

The ASAPI is a unified ingress gateway for Apsara Stack services and a main component of the Apsara Uni-manager Management Console. The ASAPI provides a unified method for you to manage and make API calls. You can use Apsara Stack SDK to centrally make API calls to Apsara Stack services, including Elastic Compute Service (ECS) and Virtual Private Cloud (VPC).



The ASAPI allows you to make API calls to manage and maintain Apsara Stack services.

- Make API calls to implement enterprise-grade management. For example, you can manage

organizations, users, roles, resource sets, and logon policies. These operations are provided by the Apsara Uni-manager Management Console.

- Make API calls to manage operations. For example, you can manage quotas, usage statistics, specifications, and password policies. These operations are provided by the Apsara Uni-manager Management Console.
- Make API calls to manage resources. For example, you can create, delete, query, or modify cloud resources. These operations are provided by Apsara Stack services. If you call an API operation to create an ECS instance or a VPC, the API operation is provided by ECS or VPC.
- Make API calls to query monitoring metrics of Apsara Stack services. These operations are provided by the Apsara Uni-manager Management Console.
- Make API calls to perform O&M. For example, you can manage monitoring and alerting, inventories, tasks, and physical servers. These operations are provided by the Apsara Uni-manager Operations Console.

For more information, see the developer guide of the corresponding Apsara Stack service. For information about API operations used to manage organization users, see *Apsara Uni-manager Management Console Developer Guide*. For information about the API operation used to create an ECS instance, see *ECS Developer Guide*.

> **Note** If the "Use Apsara Stack SDK to make API calls" section is included in the developer guide of an Apsara Stack service, the ASAPI is supported by the Apsara Stack service. In this case, you can make API calls based on the developer guide.

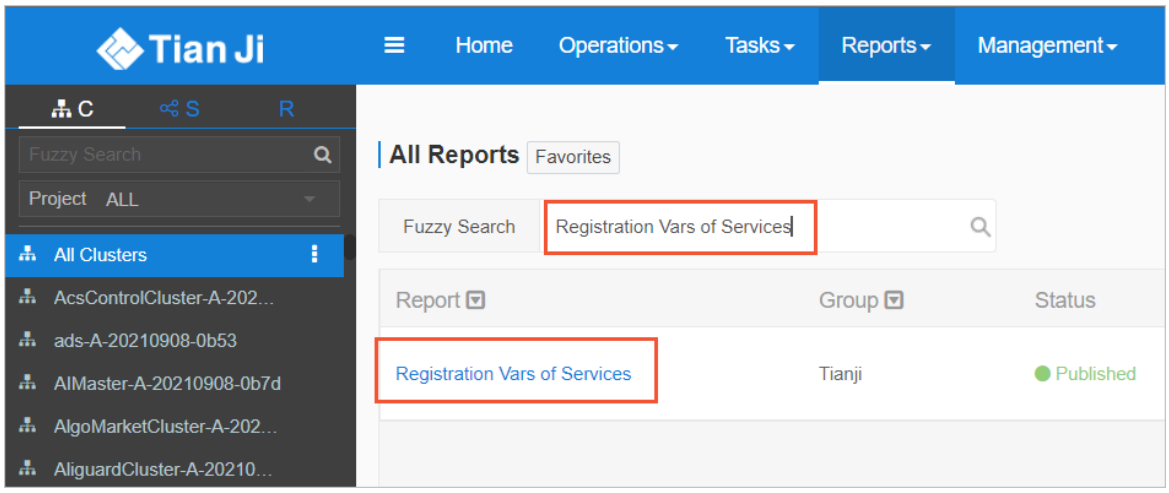
1.3.2. Generate an endpoint for ASAPI

This topic describes three methods used to generate an endpoint for Apsara Stack API (ASAPI).

Method 1: Use Registration Vars of Services to generate an endpoint

1. Log on to Apsara Infrastructure Management Framework.
   i. Log on to the Apsara Uni-manager Operations Console.
   ii. In the top navigation bar, click **O&M**.
   iii. In the left-side navigation pane, choose **Product Management** > **Products**.
   iv. In the Apsara Stack O&M section, click **Apsara Infrastructure Management Framework**.
2. In the left-side navigation pane, click **Reports**.

3. On the **All Reports** page, search for `Registration Vars of Services` and click the report name.



4. On the **Registration Vars of Services** page, click the icon next to **Service** and search for `asapi`.

5. Find the asapi service, right-click in the **Service Registration** column, and then select **Show More** from the shortcut menu.

   In the **Details** dialog box, obtain the value of **asapi.public.endpoint**.

6. Generate an endpoint for ASAPI based on the following format:

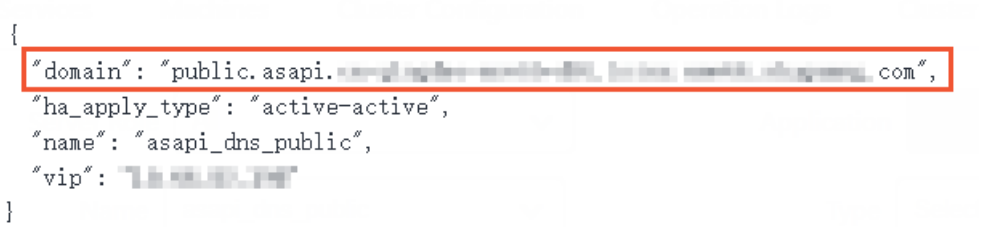
```
https://<asapi.public.endpoint>/asapi/v3
```

Method 2: Use cluster resources to generate an endpoint

1. Log on to Apsara Infrastructure Management Framework.
   i. Log on to the Apsara Uni-manager Operations Console.
   ii. In the top navigation bar, click **O&M**.
   iii. In the left-side navigation pane, choose **Product Management** > **Products**.
   iv. In the Apsara Stack O&M section, click **Apsara Infrastructure Management Framework**.
2. Go to the cluster operations page.
   i. In the left-side navigation pane, choose Operations > Cluster Operations.
   ii. In the **Cluster** search box, enter `ascm`.
   iii. Find the cluster that you want to manage and click **Operations** in the **Actions** column.
3. Obtain the value of the domain parameter.
   i. Click the Cluster Resources tab.
   ii. In the **Name** field, search for `asapi_dns_public`.

iii. Click **Details** in the **Application Parameter** column.

In the **Application Parameter** message, obtain the value of the **domain** parameter.



4. Generate an endpoint for ASAPI based on the following format:

```
https://<domain>/asapi/v3
```

Method 3: Concatenate strings to generate an endpoint

You can also concatenate strings to generate an endpoint based on the following format:

```
https://public.asapi.<region>.<internet-domain>/asapi/v3
```

> **Note**
>
> 1. Replace <region> with your actual region ID. For information about how to query a region ID, see Query a region ID.
> 2. Replace <internet-domain> with the root domain of your Apsara Stack service. To obtain the root domain, contact Apsara Stack O&M personnel. Take note of this point: Use the internet-domain parameter instead of the intranet-domain parameter.

If the **region** value is `cn-****-****-d01` and the **internet-domain** value is `inter.****.****.com`, the following endpoint can be generated:

An example of a complete endpoint:

```
https://public.asapi.cn-****-****-d01.inter.****.****.com/asapi/v3
```

1.3.3. Obtain Apsara Stack SDKs

Apsara Stack API (ASAPI) is a unified portal provided by Apsara Stack. You can use the ASAPI to connect to all Apsara Stack services. Apsara Stack provides SDKs for API calls.

You can go to Apsara Stack documentation and download the Apsara Stack SDKs in the **Other Resources** section at the lower part of the page.

1.3.4. Common request parameters

Common request parameters must be included in all API requests.

Sample common parameters

The following table describes the common parameters that are used when you call API operations supported by Apsara Stack API (ASAPI).

Parameter	Type	Required	Description
Action	String	Yes	The operation that you want to perform. For more information, see View the information about API operations.
Product	String	Yes	The name of the service.
Version	String	Yes	The version of the API operation. For more information, see View information about API operations.
RegionId	String	Yes	The ID of the region. For more informatio, see Query a region ID.
AccessKeyId	String	Yes	The AccessKey ID used to connect to the service. For more information, see Obtain an AccessKey pair.
Signature	String	Yes	The signature string of the current request. For more information about how signatures are calculated, see Request signatures.
SignatureMethod	String	Yes	The encryption method of the signature string. Valid values: HMAC-SHA1 and HMAC-SHA256.

Parameter	Type	Required	Description
SignatureVersion	String	Yes	The version of the signature encryption algorithm. Valid values: 1.0 and 2.0.
SignatureNonce	String	Yes	A unique, random number used to prevent replay attacks. You must use different numbers for different requests.
Timestamp	String	Yes	The timestamp of the request. Specify the time in the ISO 8601 standard in the yyyy-MM-ddTHH:mm:ssZ format. The time must be in UTC. Example: 2018-01-01T12:00:00Z.
Format	String	Yes	The format in which to return the response. Set the value to JSON.

> **Note**
>
> The `Signature`, `SignatureMethod`, `SignatureVersion`, `SignatureNonce`, `Timestamp`, and `Format` parameters are encapsulated into Apsara Stack SDK. You do not need to configure these parameters.

1.3.5. SDK description

1.3.5.1. SDK for Java

This topic describes how to obtain and use Apsara Stack SDK for Java to call API operations.

Prerequisites

- An authorized account and an AccessKey pair are obtained. The AccessKey pair consists of an AccessKey ID and an AccessKey secret. You can obtain and view your AccessKey pair in the Apsara Uni-manager Management Console. For more information, see Obtain an AccessKey pair.
- The endpoint of the ASAPI is obtained. For more information, see Generate an endpoint for ASAPI.

- Apsara Stack SDK for Java 1.8 or later is installed.

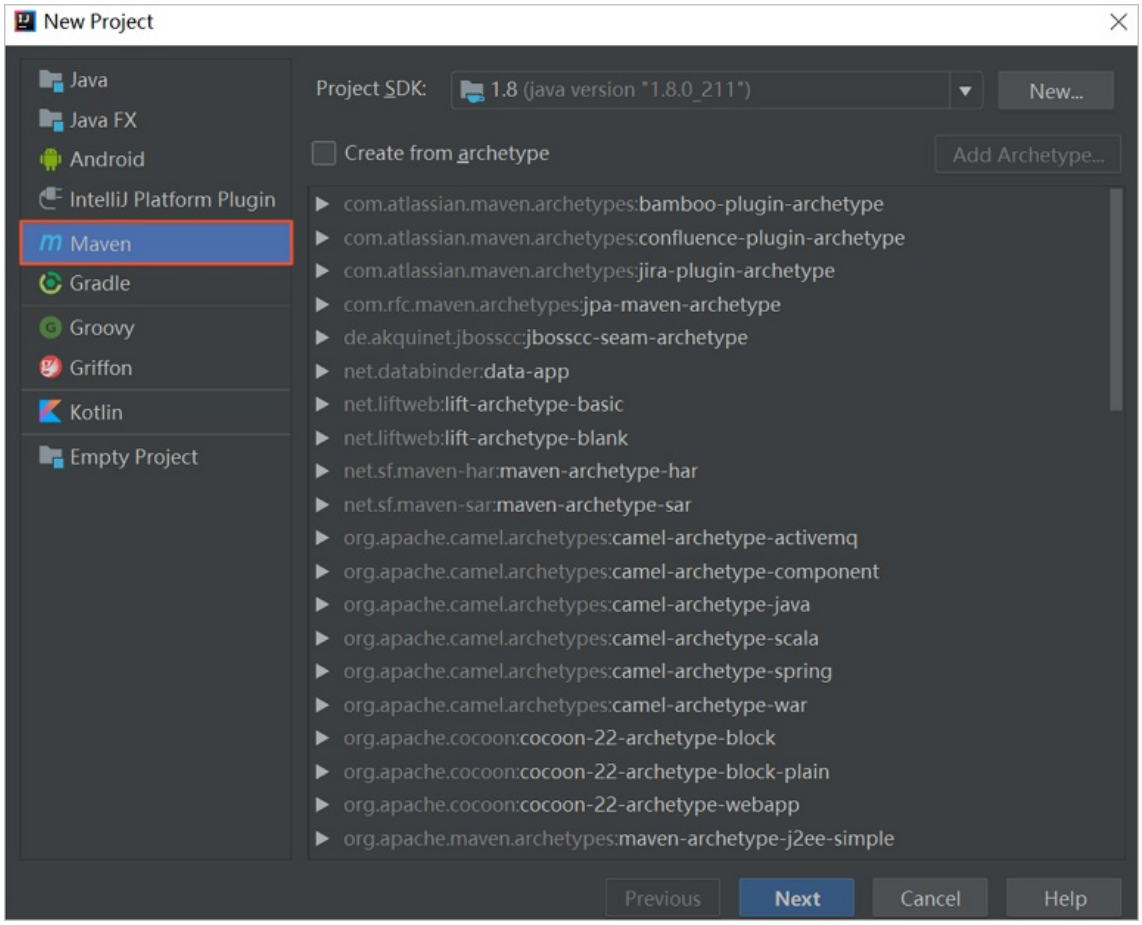
Background information

The help center of Apsara Stack provides Maven dependencies and JAR files for Apsara Stack SDK for Java. You can write code to use Apsara Stack SDK for Java to call API operations.

In this example, the Windows 10 64-bit OS and IntelliJ IDEA are used.

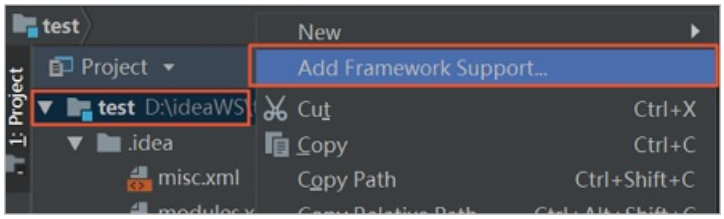
Procedure

1. Use one of the following methods to configure Maven in IntelliJ IDEA. Maven is a tool used for project management.
   - Use Maven that is integrated in IntelliJ IDEA.
   - Download Maven based on the OS that you are using from the official website and configure Maven. For more information, see Download Maven.
2. Obtain the Maven dependencies for Apsara Stack SDK for Java from the official website.
3. Use one of the following methods to create a Maven project:
   - Method 1: Create a Maven project in IntelliJ IDEA.

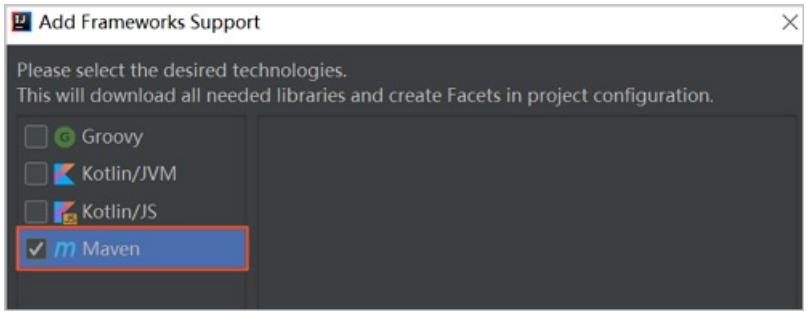


   - Method 2: Convert an existing project to a Maven project.

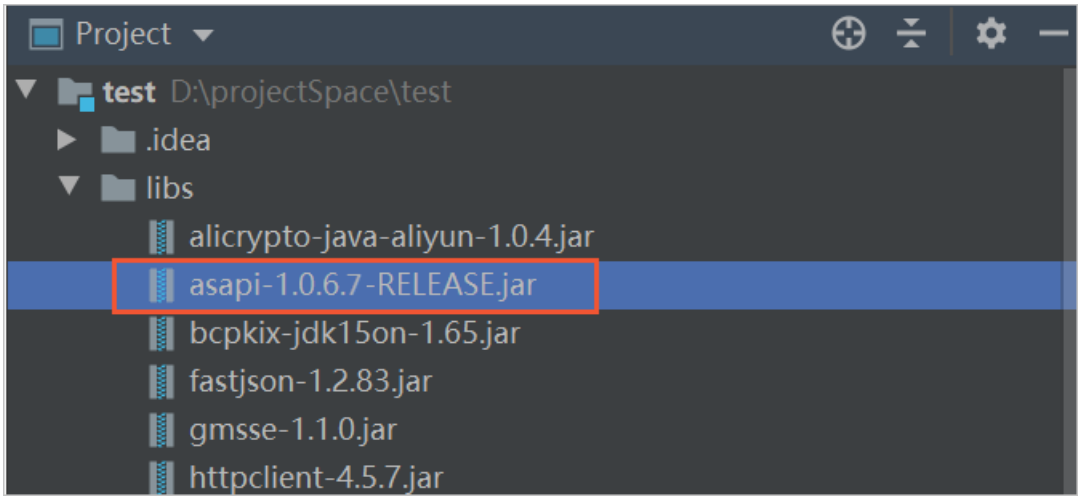
a. Right-click the project that you want to convert and select **Add Framework Support...**.



b. Select **Maven** and click **OK**.



4. Create a folder named libs in the root directory of the project. Then, add the JAR file of Apsara Stack SDK for Java to the libs folder, as shown in the following figure.



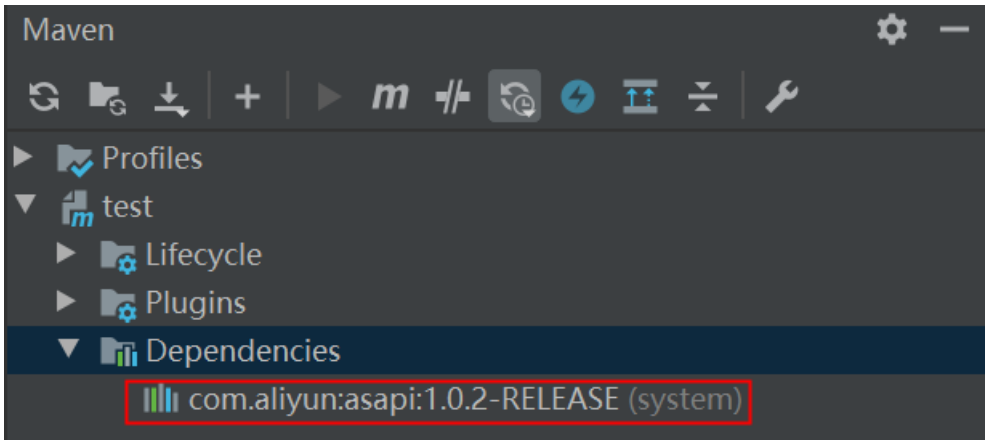
5. Add the dependencies of Apsara Stack SDK for Java to the pom.xml file in the project directory.

```
<dependencies>
    <dependency>
        <groupId>com.aliyun</groupId>
        <artifactId>asapi</artifactId>
        <version>1.0.6.4-RELEASE</version>
        <scope>system</scope>
        <systemPath>${project.basedir}/libs/asapi-1.0.6.4-RELEASE.jar</systemPath>
    </dependency>
  <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>alicrypto-java-aliyun</artifactId>
    <version>1.0.4</version>
    <systemPath>${project.basedir}/libs/alicrypto-java-aliyun-1.0.4.jar</systemPath>
  </dependency>
  <dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>gmsse</artifactId>
    <version>1.1.0</version>
    <systemPath>${project.basedir}/libs/gmsse-1.1.0.jar</systemPath>
  </dependency>
    <dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.7</version>
    <systemPath>${project.basedir}/libs/httpclient-4.5.7.jar</systemPath>
  </dependency>
</dependencies>
```

Note Replace `1.0.6.4-RELEASE` with the actual version of Apsara Stack SDK for Java.

6. On the right side of IntelliJ IDEA, view information in the Maven section.

   If the information in the following figure appears, the dependencies are added.



Use Apsara Stack SDK for Java

You can use the GET or POST method to call API operations provided by ASAPI. The following sample code shows how to call API operations:

```
// 1. Create a connection to an ASClient object.
ASClient client = new ASClient();
// 2. Configure the identifier of the requester. This parameter is required and used only f
or identification. You can set this parameter to a random value.
client.setSdkSource("<The identifier of the requester>");
Map<String, Object> parameters = new HashMap<String, Object>();
// 3. Configure the parameters for authorization. The parameter values vary based on your e
nvironment.
// The AccessKey ID of your Apsara Stack tenant account.
parameters.put("AccessKeyId","<AccessKey ID>");
// The AccessKey secret of your Apsara Stack tenant account.
parameters.put("AccessKeySecret", "<AccessKey Secret>");
// The ID of the region.
parameters.put("RegionId", "<your RegionID>");
// 4. Configure parameters for an API call. In this example, the Product, Action, and Versi
on parameters are configured.
parameters.put("Product", "<The name of the service>");
parameters.put("Action", "<The name of the API operation>");
parameters.put("Version", "<The version of the API operation>");
// When you use the STS AccessKey ID to call RPC API operations, you must add the SecurityT
oken parameter to the header.
// parameters.put("SecurityToken", "<your-securitytoken>");
// When you use the STS AccessKey ID to call ROA API operations, you must add the x-acs-sec
urity-token parameter to the header.
// headers.put("x-acs-security-token", "<your-securitytoken>");
// 5. Configure operation-specific parameters.
parameters.put("<The name of the operation-specific parameter>", "<The value of the operati
on-specific parameter>");
// 6. Configure request headers and the encryption method for the signature string. Valid v
alues of the SignatureMethod parameter: HMAC-SHA1 and HMAC-SHA256. By default, HMAC-SHA256
is used.
Map<String, String> headers = new HashMap<String, String>();
//headers.put("SignatureMethod", "HMAC-SHA1");
headers.put("x-acs-regionid", "<your-regionid>");
headers.put("x-acs-resourcegroupid", "<your-resourcegroupid>");
headers.put("x-acs-organizationid", "<your-organizationid>");
headers.put("x-acs-instanceid", "<your-instanceid>");
// 7. Make an API call. We recommend that you use the doPost() method.
// Configure the time-out period for a connection.
//client.setConnectTimeout(10000);
// Configure the time-out period for a request.
// client.setSocketTimeout(10000);
String result = client.doPost(endpoint, headers, parameters);
System.out.println(result);
```

Configure proxies

If you need to use a proxy to make API calls, use the following sample code to configure the proxy:

```
Cloud cloud = new Cloud();
// Configure the IP address of the proxy.
cloud.setConfig("proxy_host", "<IP>");
// Configure the port used by the proxy.
cloud.setConfig("proxy_port", "<Port>");
Create a connection to an ASClient object.
ASClient client = new ASClient(cloud);
```

Header parameters

Parameter	Description
x-acs-regionid	The ID of the region. For more information about how to obtain a region ID, see Query a region ID.
x-acs-organizationid	The ID of the organization in the Apsara Uni-manager Management Console.
x-acs-resourcegroupid	The ID of the resource set in the Apsara Uni-manager Management Console.
x-acs-instanceid	The ID of the instance to which you make API calls.

1.3.5.2. SDK for Python

This topic describes how to obtain and use Apsara Stack SDK for Python to call API operations.

Preparations

- An authorized account and an AccessKey pair are obtained. The AccessKey pair consists of an AccessKey ID and an AccessKey secret. For more information about how to obtain an AccessKey pair in the Apsara Uni-manager Management Console, see Obtain an AccessKey pair.
- The endpoint of ASAPI is obtained. For more information, see Generate an endpoint for ASAPI.

Install Apsara Stack SDK for Python

1. Download the required version of Apsara Stack SDK for Python. For more information, see Obtain Apsara Stack SDKs.
2. Upload the compressed file of Apsara Stack SDK for Python to the host from which you want to send API requests and extract the file. Then, run the following code in the directory of the extracted files to install Apsara Stack SDK for Python:

```
# Install Apsara Stack SDK for Python 2.x.
pip install aliyun-python2-sdk-asapi-2.4.7.tar.gz
# Install Apsara Stack SDK for Python 3.x.
pip3 install aliyun-python3-sdk-asapi-2.4.7.tar.gz
```

> **Note**
>
> Replace `2.4.7` with the actual version of Apsara Stack SDK for Python.

Sample requests

You can use the GET or POST method to call API operations provided by ASAPI.

```
# 1. Import dependencies.
   from aliyunsdkasapi.ASClient import ASClient
   from aliyunsdkasapi.AsapiRequest import AsapiRequest

   if __name__ == '__main__':
    # 2. Create a request. The Product parameter specifies the name of the service. The Version parameter specifies the version of the API operation. The Action parameter specifies the operation that you want to perform. The asapi-endpoint parameter specifies the endpoint of ASAPI.
    req = AsapiRequest("<Product>", "<Version>", "<Action>", "<asapi-endpoint>")
    # 3. Configure operation-specific parameters.
    req.add_body_params("<key>", "<value>")
 # When you use the STS AccessKey ID to call RPC API operations, you must add the SecurityToken parameter to the header.
 # req.add_body_params("SecurityToken", "<your-securitytoken>");
 # When you use the STS AccessKey ID to call ROA API operations, you must add the x-acs-security-token parameter to the header.
 # req.add_header("x-acs-security-token", "<your-securitytoken>");
    # 4. Configure request headers.
    req.add_header("x-acs-resourcegroupid", "<your-resourcegroupid>");
    req.add_header("x-acs-organizationid", "<your-organizationid>");
    req.add_header("x-acs-regionid", "<your-regionid>");
    req.add_header("x-acs-instanceid", "<your-instanceid>");
    # 5. Configure a request method.
    req.set_method("POST")
    # 6. Configure initialization parameters for an ASClient object. The parameter values vary based on your environment. The timeout parameter specifies the time-out period for a request. The cert_file parameter specifies the certificate. The verify parameter specifies whether to verify the certificate.
    as_client = ASClient("<accesskeyid>", "<accesskeysecret>", "<regionid>",timeout=1000,
            cert_file=None,
            verify=False)
    # Configure the identifier of the requester. This parameter is required and used only for identification. You can set this parameter to a random value.
    as_client.setSdkSource("<The identifier of the requester>")
    response = as_client.do_request(req)
    print(response)
```

Header parameters

Parameter	Description
x-acs-resourcegroupid	The ID of the resource set in the Apsara Uni-manager Management Console.
x-acs-organizationid	The ID of the organization in the Apsara Uni-manager Management Console.
x-acs-regionid	The ID of the region. For more information about how to obtain a region ID, see Query a region ID.
x-acs-instanceid	The ID of the instance on which you want to perform operations.

Configure proxies

Perform the following steps to configure a proxy:

1. Import OS dependencies.

```
import os
```

2. Configure one or more proxies.

```
# Before you make an API call, configure one or more proxies.
os.environ['http_proxy'] = "http://x.x.x.x:xxxx"
os.environ['https_proxy'] = "https://x.x.x.x:xxxx"
```

1.3.5.3. SDK for Node.js

This topic describes how to obtain and use Apsara Stack SDK for Node.js to call API operations.

Preparations

- An authorized account and an AccessKey pair are obtained. The AccessKey pair consists of an AccessKey ID and an AccessKey secret. For more information about how to obtain an AccessKey pair in the Apsara Uni-manager Management Console, see Obtain an AccessKey pair.
- The endpoint of Apsara Stack API (ASAPI) is obtained. For more information, see Generate an endpoint for the ASAPI.
- For more information about how to obtain the region ID, see Obtain the endpoint of ASAPI.

Add dependencies

Run the following command to install Apsara Stack SDK for Node.js:

```
tnpm i @ali/asapi-nodejs-sdk@0.0.33 --save
```

Use Apsara Stack SDK for Node.js

The following sample code shows how to use Apsara Stack SDK for Node.js to make API calls:

```
// 1. Import the dependencies of Apsara Stack SDK for Node.js.
const Controller = require('egg').Controller;
const ASAPIClient = require('@ali/asapi-nodejs-sdk');
// 2. Configure the parameters.
const cnofig = {
    API_GATEWAY: '<The endpoint of ASAPI>',
    regionId: '<The ID of the region in which the Apsara Stack service is deployed>',
    accessKeyId: '<The AccessKey ID of your Apsara Stack tenant account>',
    accessKeySecret: '<The AccessKey secret of your Apsara Stack tenant account>'
};
// 3. Initialize a client for API calls.
const AsapiClient = new ASAPIClient(cnofig, this.ctx);
// 4. Configure request parameters.
const params = {
    "Action": "<The name of the API operation>",
    "Product": "<The name of the service that provides the API operation>",
    "Version": "<The version of the API operation>",
    "regionId": '<The ID of the region in which the Apsara Stack service is deployed>',
    // When you use the STS AccessKey ID to call RPC API operations, you must add the Securi
tyToken parameter to the header.
    // "SecurityToken": "<your-securitytoken>",
      "<The name of the operation-specific parameter required by the API operation>": "<The
value of the operation-specific parameter>"
}
// 5. Configure request headers.
// When you use the STS AccessKey ID to call ROA API operations, you must add the x-acs-secu
rity-token parameter to the header.
// AsapiClient.addHeader("x-acs-security-token", "<your-securitytoken>")
AsapiClient.addHeader('x-acs-caller-sdk-source', '<The identifier of the requester>')
AsapiClient.addHeader('x-acs-roleid', '<The ID of the role>')
AsapiClient.addHeader('x-acs-userid', '<The ID of the user>')
AsapiClient.addHeader('x-acs-organizationid', '<your-organizationid>')
AsapiClient.addHeadert("x-acs-regionid", "<your-regionid>");
AsapiClient.addHeader("x-acs-resourcegroupid", "<your-resourcegroupid>");
AsapiClient.addHeader("x-acs-instanceid", "<your-instanceid>");
// 6. Make an API call and obtain the response.
try {
  const res = await AsapiClient.doRequest(cnofig.API_GATEWAY, params, {
    method: 'POST', // The request method.
    formatParams: false, // Specifies whether to format parameters and capitalize the first
letter of each parameter.
    enableProxy: true, // Specifies whether to configure a proxy.
    proxy: 'http://x.x.x.x:xxxx',
    timeout: 10000, // The time-out period.
  // ca: , // The certificate issued by a certificate authority (CA). For more information,
visit https://nodejs.org/api/https.html#https_https_request_options_callback.
  })
  console.log(res)
} catch (e) {
  console.log(e)
}
```

Disable certificate validation

The following sample code shows how to disable certificate validation:

```
Configure the environment variable.
NODE_TLS_REJECT_UNAUTHORIZED=0
```

Request headers

Parameter	Description
x-acs-regionid	The ID of the region. For more information about how to obtain a region ID, see Query a region ID.
x-acs-organizationid	The ID of the organization in the Apsara Uni-manager Management Console.
x-acs-resourcegroupid	The ID of the resource set in the Apsara Uni-manager Management Console.
x-acs-instanceid	The ID of the instance on which you want to perform operations.
x-acs-caller-sdk-source	The custom name of the requester or the system. This parameter is used to identify the source of the API request.
x-acs-roleid	The ID of the role.
x-acs-userid	The ID of the user.

1.3.6. HTTP requests

1.3.6.1. Overview

This topic describes how to make API requests and is intended for users who use API URLs to initiate HTTP GET requests.

If you are using an SDK to initiate requests, you can skip this topic.

The request URL consists of different parameters and has a fixed syntax. The URL contains common request parameters, your signature, and operation-specific parameters. Sample request URLs are provided for each API operation. These URLs are not encoded to make them easy to read. Before you make a request, you must encode the request URL. After the authentication process is complete based on your signature, the results are returned to you. Response parameters are returned if the call is successful, whereas an error message is returned if the call fails. You can troubleshoot issues based on the error message.

1.3.6.2. Request structure

You can use API URLs to initiate HTTP GET requests. Each URL must include request parameters. This topic describes the syntax of HTTP GET requests and provides the endpoint of ASAPI.

Sample request

GetOrganizationList is used in the following unencoded URL request:

```
https://public.asapi.aliyuns.com/asapi/v3?Action=GetOrganizationList&Product=ascm&Version=2019-05-10&Id=<Organization ID>&<Common parameters>
```

- `https` : the protocol used to transmit the request.
- `public.asapi.aliyuns.com/asapi/v3` : the endpoint of ASAPI.
- `Action=GetOrganizationList Product=ascm Version=2019-05-10` : the API operation.
- `Id` : the ID of the organization.
- `<Common request parameters>` : common parameters specified in the system.

Protocols

ASAPI supports only the HTTP request protocol.

Endpoints

For more information about how to obtain the endpoint of ASAPI, see Generate an endpoint for ASAPI.

You must use the `Action, Product, and Version` parameters to specify an API operation. For example, you can set `Action to GetOrganizationList, Product to ascm, and Version to 2019-05-10`. You must also specify other operation-specific request parameters and common request parameters. For more information, see Common request parameters.

1.3.6.3. Request signatures

This topic describes how to sign HTTP API requests. ASAPI uses the request signature to verify the identity of the request sender. ASAPI implements symmetric encryption by using an AccessKey pair to verify the identity of the request sender.

> **Note**
>
> - An AccessKey pair consists of an AccessKey ID and an AccessKey secret, which are issued to each user or organization in the Apsara Uni-manager Management Console.
> - The AccessKey ID is used to verify identities of users, and the AccessKey secret is used to encrypt and verify signature strings. You must keep your AccessKey secret confidential.
> - ASAPI provides SDKs for Node.js, Python, and Java to facilitate complex signature calculation. For more information, see Obtain the core package for ASAPI.

Step 1: Create a canonicalized query string

1. Arrange the request parameters in alphabetical order, including all common and operation-specific parameters except the `Signature` parameter.

   > **Note**
   >
   > If you use the GET method to send a request,these parameters are connected by ampersands (&) and included in the request URL. The first request parameter follows a question mark (?).

2. Encode the request parameters and their values in UTF-8 based on RFC 3986. The following rules apply in the encoding process:
   - Letters, digits, and special characters such as hyphens (`-`), underscores (`_`), and periods (`.`) do not need to be encoded.
   - Other characters must be percent encoded in the `%XY` format. `XY` represents the ASCII code of the characters in hexadecimal notation. For example, double quotation marks (`"`) are encoded as `%22` .
   - Extended UTF-8 characters are encoded in the `%XY%ZA…` format.
   - Spaces must be encoded as `%20` . Do not encode spaces as plus signs (`+`).

     This encoding method is similar to but different from the `application/x-www-form-urlencoded` MIME encoding algorithm.

     If you use `java.net.URLEncoder` from the Java standard library, use `percentEncode` to encode request parameters and their values. In the encoded query string, replace the plus signs (`+`) with `%20` , asterisks (`*`) with `%2A` , and `%7E` with tildes (`~`). This way, you can obtain an encoded string that matches the preceding encoding rules.

     ```
     private static final String ENCODING = "UTF-8";
     private static String percentEncode(String value) throws UnsupportedEncodingException
     {
         return value != null ? URLEncoder.encode(value, ENCODING).replace("+", "%20").rep
     lace("*", "%2A").replace("%7E", "~") : null;
     }
     ```

3. Use equal signs (`=`) to connect the encoded request parameters and their values.
4. Use ampersands (`&`) to connect the encoded request parameters. These parameters must be arranged in the same order as those in Step 1.

Then, a canonicalized query string that follows the request structure is obtained. For more information, see Request structure.

Step 2: Create a string-to-sign

1. Create a `string-to-sign`. You can also use `percentEncode` to encode the canonicalized query string created in Step 1 based on the following rules:

```
StringToSign=
    HTTPMethod + "&" + //HTTPMethod: the HTTP method that is used to send a request, such as GET.
    percentEncode("/") + "&" + //percentEncode("/"): Encode the forward slash (/) in UTF-8 as %2F.
    percentEncode(CanonicalizedQueryString) // Encode the canonicalized query string created in Step 1.
```

2. Calculate the RFC 2104-compliant HMAC-SHA1 or HMAC-256 value of `StringToSign`. In the following example, Java Base64 is used:

```
String signature = Base64.getEncoder().encodeToString(signData);
```

> **Note**
>
> When you calculate the signature, the key value specified by RFC 2104 is your `AccessKey secret` plus an ampersand (&), which has an ASCII value of 38. For more information, see Obtain an AccessKey pair.

3. Encode the `Signature` parameter based on RFC 3986 and add it to the canonicalized query string.

Example 1: Concatenate parameters

`GetOrganizationList` is used in this example to query regions. For example, `AccessKeyID` is set to testid and `AccessKeySecret` is set to testsecret. Perform the following operations:

1. Construct a canonicalized query string.

```
https://public.asapi.aliyuns.com/asapi/v3/?AccessKeyId=testid&Action=GetOrganizationList&Format=JSON&Id=1&Product=ascm&RegionId=cn-region-test&SignatureMethod=HMAC-SHA1&SignatureNonce=43967c6e-e3ad-44fa-b06b-fcf3db5f39d2&SignatureVersion=1.0&Timestamp=2020-09-15T08%3A50%3A40Z&Version=2019-05-10
```

2. Create a `string-to-sign`.

```
GET&%2F&AccessKeyId%3Dtestid%26Action%3DGetOrganizationList%26Format%3DJSON%26Id%3D1%26Product%3Dascm%26RegionId%3Dcn-region-test%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3D43967c6e-e3ad-44fa-b06b-fcf3db5f39d2%26SignatureVersion%3D1.0%26Timestamp%3D2020-09-15T08%253A50%253A40Z%26Version%3D2019-05-10
```

3. Calculate the signature.

Calculate the signature. The key value used for calculation is `testsecret&` because `AccessKeySecret` is set to testsecret. The calculated signature is `OLeaidS1JvxuMvnyHOwuJ+uX5qY=`. Java Base64 is used in the following sample code:

`String signature = Base64.getEncoder().encodeToString(signData);` Encode `Signature=hvqv9Ll5RE6kQ4YsQXFAf9VuRIY=` based on RFC 3986 and add it to the canonicalized query string.

```
https://public.asapi.aliyuns.com/asapi/v3/?AccessKeyId=testid&Action=GetOrganizationList&Format=JSON&Id=1&Product=ascm&RegionId=cn-region-test&SignatureMethod=HMAC-SHA1&SignatureNonce=43967c6e-e3ad-44fa-b06b-fcf3db5f39d2&SignatureVersion=1.0&Timestamp=2020-09-15T08%3A50%3A40Z&Version=2019-05-10
```

You can use browsers or tools such as cURL and Wget to initiate an HTTP request for calling the `DescribeRegions` operation to query the Alibaba Cloud region list.

Example 2: Use the standard library of the programming language

`GetOrganizationList` is used in this example to query regions. For example, `AccessKeyID` is set to testid, `AccessKeySecret` is set to testsecret, and all request parameters are stored in a Java `Map<String, String>` object.

1. Predefine an encoding method.

```
private static final String ENCODING = "UTF-8";
private static String percentEncode(String value) throws UnsupportedEncodingException {
        return value != null ? URLEncoder.encode(value, ENCODING).replace("+", "%20").replace("*", "%2A").replace("%7E", "~") : null;
}
```

2. Predefine the time format for the `Timestamp` parameter. Specify the `Timestamp` parameter in the ISO 8601 format. The time must be in UTC+0.

```
private static final String ISO8601_DATE_FORMAT = "yyyy-MM-dd'T'HH:mm:ss'Z'";
private static String formatIso8601Date(Date date) {
    SimpleDateFormat df = new SimpleDateFormat(ISO8601_DATE_FORMAT);
    df.setTimeZone(new SimpleTimeZone(0, "GMT"));
    return df.format(date);
}
```

3. Create a request string.

```
final String HTTP_METHOD = "GET";
Map parameters = new HashMap();
// Specify request parameters.
parameters.put("Product", "ascm");
parameters.put("Action", "GetOrganizationList");
parameters.put("Version", "2019-05-10");
parameters.put("AccessKeyId", "testid");
parameters.put("RegionId", "cn-region-test");
parameters.put("Timestamp", formatIso8601Date(new Date()));
parameters.put("SignatureMethod", "HMAC-SHA1");
parameters.put("SignatureVersion", "1.0");
parameters.put("SignatureNonce", UUID.randomUUID().toString());
parameters.put("Format", "JSON");
parameters.put("Id", "1");
// Arrange request parameters.
String[] sortedKeys = parameters.keySet().toArray(new String[]{});
Arrays.sort(sortedKeys);
final String SEPARATOR = "&";
// Create a string-to-sign.
StringBuilder stringToSign = new StringBuilder();
stringToSign.append(HTTP_METHOD).append(SEPARATOR);
stringToSign.append(percentEncode("/")).append(SEPARATOR);
StringBuilder canonicalizedQueryString = new StringBuilder();
for(String key : sortedKeys) {
    // Encode the key and the value.
    canonicalizedQueryString.append("&")
    .append(percentEncode(key)).append("=")
    .append(percentEncode(parameters.get(key)));
}
//Encode the canonicalized query string.
stringToSign.append(percentEncode(
canonicalizedQueryString.toString().substring(1)));
```

4. The signature. The key value used for calculation is `testsecret&` because `AccessKeySecret` is set to testsecret. The calculated signature is `OLeaidS1JvxuMvnyHOwuJ%2BuX5qY%3D` .

```
// The following sample code shows how to calculate the signature:
final String ALGORITHM = "HmacSHA1";
final String ENCODING = "UTF-8";
key = "testsecret&";
Mac mac = Mac.getInstance(ALGORITHM);
mac.init(new SecretKeySpec(key.getBytes(ENCODING), ALGORITHM));
byte[] signData = mac.doFinal(stringToSign.getBytes(ENCODING));
String signature = Base64.getEncoder().encodeToString(signData);
```

Encode the Signature parameter based on RFC 3986 and add it to the canonicalized query string. The following URL is obtained:

```
https://public.asapi.aliyuns.com/asapi/v3/?AccessKeyId=testid&Action=GetOrganizationLis
t&Format=JSON&Id=1&Product=ascm&RegionId=cn-region-test&SignatureMethod=HMAC-SHA1&Signa
tureNonce=43967c6e-e3ad-44fa-b06b-fcf3db5f39d2&SignatureVersion=1.0&Timestamp=2020-09-1
5T08%3A50%3A40Z&Version=2019-05-10
```

5. Use browsers or tools such as cURL and Wget to initiate an HTTP request.

1.3.6.4. Responses

Responses are returned only in the JSON format. If a return code is greater than or equal to 200 and less than 300, a success response is returned. Otherwise, an error response is returned.

Sample success responses

If a request is successful, the request ID and operation-specific parameter values are returned. For more information, see the developer guide of the corresponding Apsara Stack service.

```
   "successResponse":true,
  "eagleEyeTraceId":"xxxxxxxxxxxxx",
  "asapiSuccess":true,
  "code":"200",
  "cost":12,
  "asapiRequestId":"xxxxxxxxxxxxx",
  "data":[
    {
      "muserId":"aliyuntest",
      "internal":false,
      "level":"0",
      "supportRegions":"cn-*",
      "personNum":100,
      "mtime":1553247962000,
      "uuid":"1",
      "parentId":0,
      "supportRegionList":[
        "cn-*"
      ],
      "name":"root",
      "alias":"root",
      "ctime":1553247962000,
      "id":1,
      "resourceGroupNum":100,
      "cuserId":"aliyuntest"
    }
  ],
  "requestId":"xxxxxxxxxxxxx",
  "success":true,
  "message":"success"
}
```

Sample error responses

If a request fails, the following information is returned. For more information, see the developer guide of the corresponding Apsara Stack service.

```
{
    "code": "asapi.server.api.notfound",// The error code.
    "eagleEyeTraceId": "xxx-xxx-xxx-xxx", // The trace ID provided by EagleEye.
    "asapiSuccess": false, // Indicates whether the request is successful.
    "hostId": "xxxx", // The ID of the host.
    "message": "Failed to ....", // The message returned.
    "asapiErrorMessage": "Failed to ....", // The error message provided by ASAPI.
    "asapiErrorCode": "asapi.server.api.notfound" // The error code provided by ASAPI.
}
```

2.Flink SQL

2.1. Overview

As a programming language that conforms to standard SQL semantics, Flink SQL is designed to simplify the computational model, making it easy for users to use Realtime Compute. Flink SQL has the following advantages:

- Low development cost. Flink SQL enables you to implement complex and varied business logic by using simple SQL statements instead of thousands of lines of Apache Storm code.
- Low maintenance cost. Flink SQL eliminates the need for you to maintain thousands of lines of Storm code.

When you use Apache Storm code, system exceptions, errors, or even breakdown may occur and cause data inaccuracies.

Flink SQL features specified semantics to resolve the preceding issues. Realtime Compute has the following advantages:

- Data accuracy. Realtime Compute provides unique exactly-once semantics to ensure data accuracy in various failover scenarios.
- Performance optimization. Realtime Compute integrates SQL optimization logic to translate SQL code into underlying Apache Storm code that can be used by universal developers. This helps to improve SQL code compilation efficiency.
- Programming extensibility. Realtime Compute supports user-defined functions (UDFs). You can customize functions to implement a custom business logic or a logic that cannot be achieved by SQL statements.
- System scalability. Realtime Compute supports auto scaling upon data surges (for example, during the Double 11) while ensuring job and system stability.

2.2. Keywords

This topic describes the keywords in Realtime Compute.

Realtime Compute keywords

Common keywords

Keyword type	Keywords
Data type	VARCHAR, INT, BIGINT, DOUBLE, DATE, BOOLEAN, TINYINT, SMALLINT, FLOAT, DECIMAL, and VARBINARY
DDL statement	CREATE TABLE, CREATE FUNCTION, and CREATE VIEW
DML statement	INSERT INTO
SELECT clause	SELECT FROM, WHERE, GROUP BY, and JOIN

Naming conventions

Names of source tables, result tables, views, and aliases must follow the standard database naming conventions. The names must start with a letter, and can only contain letters, numbers, and underscores (_).

Reserved keywords

The following strings are reserved as keywords for later use. If you want to use any of the following keywords as a field name, enclose the keyword in backticks (`), for example, `` `value` ``.

```
A, ABS, ABSOLUTE, ACTION, ADA, ADD, ADMIN, AFTER, ALL, ALLOCATE, ALLOW, ALTER, ALWAYS, AND,
ANY, ARE, ARRAY, AS, ASC, ASENSITIVE, ASSERTION, ASSIGNMENT, ASYMMETRIC, AT, ATOMIC,
ATTRIBUTE, ATTRIBUTES, AUTHORIZATION, AVG, BEFORE, BEGIN, BERNOULLI, BETWEEN, BIGINT, BINARY,
BIT, BLOB, BOOLEAN, BOTH, BREADTH, BY, C, CALL, CALLED, CARDINALITY, CASCADE, CASCADED, CASE,
CAST, CATALOG, CATALOG_NAME, CEIL, CEILING, CENTURY, CHAIN, CHAR, CHARACTER, CHARACTERISTICTS,
CHARACTERS, CHARACTER_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_NAME, CHARACTER_SET_SCHEMA,
CHAR_LENGTH, CHECK, CLASS_ORIGIN, CLOB, CLOSE, COALESCE, COBOL, COLLATE, COLLATION,
COLLATION_CATALOG, COLLATION_NAME, COLLATION_SCHEMA, COLLECT, COLUMN, COLUMN_NAME,
COMMAND_FUNCTION, COMMAND_FUNCTION_CODE, COMMIT, COMMITTED, CONDITION, CONDITION_NUMBER,
CONNECT, CONNECTION, CONNECTION_NAME, CONSTRAINT, CONSTRAINTS, CONSTRAINT_CATALOG,
CONSTRAINT_NAME, CONSTRAINT_SCHEMA, CONSTRUCTOR, CONTAINS, CONTINUE, CONVERT, CORR,
CORRESPONDING, COUNT, COVAR_POP, COVAR_SAMP, CREATE, CROSS, CUBE, CUME_DIST, CURRENT,
CURRENT_CATALOG, CURRENT_DATE, CURRENT_DEFAULT_TRANSFORM_GROUP, CURRENT_PATH, CURRENT_ROLE,
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_TRANSFORM_GROUP_FOR_TYPE,
CURRENT_USER, CURSOR, CURSOR_NAME, CYCLE, DATA, DATABASE, DATE, DATETIME_INTERVAL_CODE,
DATETIME_INTERVAL_PRECISION, DAY, DEALLOCATE, DEC, DECADE, DECIMAL, DECLARE, DEFAULT,
DEFAULTS, DEFERRABLE, DEFERRED, DEFINED, DEFINER, DEGREE, DELETE, DENSE_RANK, DEPTH, DEREF,
DERIVED, DESC, DESCRIBE, DESCRIPTION, DESCRIPTOR, DETERMINISTIC, DIAGNOSTICS, DISALLOW,
DISCONNECT, DISPATCH, DISTINCT, DOMAIN, DOUBLE, DOW, DOY, DROP, DYNAMIC, DYNAMIC_FUNCTION,
DYNAMIC_FUNCTION_CODE, EACH, ELEMENT, ELSE, END, END-EXEC, EPOCH, EQUALS, ESCAPE, EVERY,
EXCEPT, EXCEPTION, EXCLUDE, EXCLUDING, EXEC, EXECUTE, EXISTS, EXP, EXPLAIN, EXTEND, EXTERNAL,
EXTRACT, FALSE, FETCH, FILTER, FINAL, FIRST, FIRST_VALUE, FLOAT, FLOOR, FOLLOWING, FOR,
FOREIGN, FORTRAN, FOUND, FRAC_SECOND, FREE, FROM, FULL, FUNCTION, FUSION, G, GENERAL,
GENERATED, GET, GLOBAL, GO, GOTO, GRANT, GRANTED, GROUP, GROUPING, HAVING, HIERARCHY, HOLD,
HOUR, IDENTITY, IMMEDIATE, IMPLEMENTATION, IMPORT, IN, INCLUDING, INCREMENT, INDICATOR,
INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INSTANCE, INSTANTIABLE, INT, INTEGER,
INTERSECT, INTERSECTION, INTERVAL, INTO, INVOKER, IS, ISOLATION, JAVA, JOIN, K, KEY,
KEY_MEMBER, KEY_TYPE, LABEL, LANGUAGE, LARGE, LAST, LAST_VALUE, LATERAL, LEADING, LEFT,
LENGTH, LEVEL, LIBRARY, LIKE, LIMIT, LN, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR, LOWER, M,
MAP, MATCH, MATCHED, MAX, MAXVALUE, MEMBER, MERGE, MESSAGE_LENGTH, MESSAGE_OCTET_LENGTH,
MESSAGE_TEXT, METHOD, MICROSECOND, MILLENNIUM, MIN, MINUTE, MINVALUE, MOD, MODIFIES, MODULE,
MONTH, MORE, MULTISET, MUMPS, NAME, NAMES, NATIONAL, NATURAL, NCHAR, NCLOB, NESTING, NEW,
NEXT, NO, NONE, NORMALIZE, NORMALIZED, NOT, NULL, NULLABLE, NULLIF, NULLS, NUMBER, NUMERIC,
OBJECT, OCTETS, OCTET_LENGTH, OF, OFFSET, OLD, ON, ONLY, OPEN, OPTION, OPTIONS, OR, ORDER,
ORDERING, ORDINALITY, OTHERS, OUT, OUTER, OUTPUT, OVER, OVERLAPS, OVERLAY, OVERRIDING, PAD,
PARAMETER, PARAMETER_MODE, PARAMETER_NAME, PARAMETER_ORDINAL_POSITION,
PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_NAME, PARAMETER_SPECIFIC_SCHEMA, PARTIAL,
PARTITION, PASCAL, PASSTHROUGH, PATH, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK, PLACING,
PLAN, PLI, POSITION, POWER, PRECEDING, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR,
PRIVILEGES, PROCEDURE, PUBLIC, QUARTER, RANGE, RANK, READ, READS, REAL, RECURSIVE, REF,
REFERENCES, REFERENCING, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2,
REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY, RELATIVE, RELEASE, REPEATABLE, RESET, RESTART,
RESTRICT, RESULT, RETURN, RETURNED_CARDINALITY, RETURNED_LENGTH, RETURNED_OCTET_LENGTH,
RETURNED_SQLSTATE, RETURNS, REVOKE, RIGHT, ROLE, ROLLBACK, ROLLUP, ROUTINE, ROUTINE_CATALOG,
ROUTINE_NAME, ROUTINE_SCHEMA, ROW, ROWS, ROW_COUNT, ROW_NUMBER, SAVEPOINT, SCALE, SCHEMA,
```

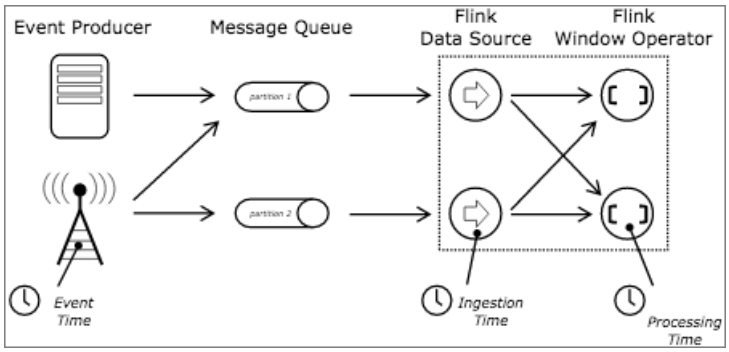
```
SCHEMA_NAME, SCOPE, SCOPE_CATALOGS, SCOPE_NAME, SCOPE_SCHEMA, SCROLL, SEARCH, SECOND, SECTION,
SECURITY, SELECT, SELF, SENSITIVE, SEQUENCE, SERIALIZABLE, SERVER, SERVER_NAME, SESSION,
SESSION_USER, SET, SETS, SIMILAR, SIMPLE, SIZE, SMALLINT, SOME, SOURCE, SPACE, SPECIFIC,
SPECIFICTYPE, SPECIFIC_NAME, SQL, SQLEXCEPTION, SQLSTATE, SQLWARNING, SQL_TSI_DAY,
SQL_TSI_FRAC_SECOND, SQL_TSI_HOUR, SQL_TSI_MICROSECOND, SQL_TSI_MINUTE, SQL_TSI_MONTH,
SQL_TSI_QUARTER, SQL_TSI_SECOND, SQL_TSI_WEEK, SQL_TSI_YEAR, SQRT, START, STATE, STATEMENT,
STATIC, STDDEV_POP, STDDEV_SAMP, STREAM, STRUCTURE, STYLE, SUBCLASS_ORIGIN, SUBMULTISET,
SUBSTITUTE, SUBSTRING, SUM, SYMMETRIC, SYSTEM, SYSTEM_USER, TABLE, TABLESAMPLE, TABLE_NAME,
TEMPORARY, THEN, TIES, TIME, TIMESTAMP, TIMESTAMPADD, TIMESTAMPDIFF, TIMEZONE_HOUR,
TIMEZONE_MINUTE, TINYINT, TO, TOP_LEVEL_COUNT, TRAILING, TRANSACTION, TRANSACTIONS_ACTIVE,
TRANSACTIONS_COMMITTED, TRANSACTIONS_ROLLED_BACK, TRANSFORM, TRANSFORMS, TRANSLATE,
TRANSLATION, TREAT, TRIGGER, TRIGGER_CATALOG, TRIGGER_NAME, TRIGGER_SCHEMA, TRIM, TRUE, TYPE,
UESCAPE, UNBOUNDED, UNCOMMITTED, UNDER, UNION, UNIQUE, UNKNOWN, UNNAMED, UNNEST, UPDATE,
UPPER, UPSERT, USAGE, USER, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_CODE,
USER_DEFINED_TYPE_NAME, USER_DEFINED_TYPE_SCHEMA, USING, VALUE, VALUES, VARBINARY, VARCHAR,
VARYING, VAR_POP, VAR_SAMP, VERSION, VIEW, WEEK, WHEN, WHENEVER, WHERE, WIDTH_BUCKET, WINDOW,
WITH, WITHIN, WITHOUT, WORK, WRAPPER, WRITE, XML, YEAR, ZONE
```

2.3. Basic concepts

2.3.1. Time attributes

This topic describes the following time attributes that are supported by Blink SQL: event time and processing time.

Apache Flink supports three time attributes for the processing of streaming data: processing time, event time, and ingestion time.



Blink SQL supports only two of the three time attributes:

- Event time: the event time that you provide in the data store. In most cases, the event time is the original time when the data is created.
- Processing time: the local system time when the system processes an event. The unit is milliseconds.

Event Time

The event time is also known as rowtime. The event time attribute must be declared in the data definition language (DDL) statement that you execute to create a source table. You can declare a field in the source table as the rowtime field. Note that you can declare a field of only the TIMESTAMP type as the rowtime field. In the future, you can declare a field of the LONG type as the rowtime field. If the source table does not contain a TIMESTAMP column, you can use a computed column to create a TIMESTAMP column based on an existing column. For more information, see Computed column.

In some scenarios, the order in which data records are received may be different from the order in which they are processed. The possible causes include out-of-order input data and network jitters. The network jitters may be caused by network congestions and transmission latencies. Before you define a rowtime field, define a computing method for watermarks in an explicit way. For more information, see Watermark.

In the following example, data is aggregated by using event time-based window functions:

```
CREATE TABLE tt_stream (
  a VARCHAR,
  b VARCHAR,
  ts TIMESTAMP,
  WATERMARK wk1 FOR ts as withOffset (ts, 1000) --Define a computing method for watermarks.
) WITH (
  type = 'sls',
  topic = '<yourTopicName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>'
);
CREATE TABLE rds_output (
  id VARCHAR,
  win_start TIMESTAMP,
  win_end TIMESTAMP,
  cnt BIGINT
) WITH (
  type = 'rds',
  url = 'jdbc:mysql://****3306/test',
  tableName = '<yourTableName>',
  userName = '<yourUserName>',
  password = '<yourPassword>'
);
INSERT
  INTO rds_output
SELECT
  a AS id,
  SESSION_START (ts, INTERVAL '1' SECOND) AS win_start,
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,
  COUNT (a) AS cnt
FROM
  tt_stream
GROUP
  BY SESSION (ts, INTERVAL '1' SECOND),
  a
```

Processing Time

The processing time is generated by the system and is not included in the raw data. Therefore, you must explicitly define a processing time column when you declare the source table.

```
filedName as PROCTIME()
```

In the following example, processing time-based window functions are used to aggregate data:

```
CREATE TABLE mq_stream (
    a VARCHAR,
    b VARCHAR,
    c BIGINT,
    ts AS PROCTIME () --Explicitly define a processing time column when you declare the sou
rce table.
  ) WITH (
    type = 'mq',
    topic = '<yourTopic>',
    accessId = '<yourAccessId>',
    accessKey = '<yourAccessSecret>'
  );
CREATE TABLE rds_output (
  id VARCHAR,
  win_start TIMESTAMP,
  win_end TIMESTAMP,
  cnt BIGINT
) with (
  type = 'rds',
  url = '<yourDatebaseURL>',
  tableName = '<yourDatabasTableName>',
  userName = '<yourUserName>',
  password = '<yourPassword>'
);
INSERT
  INTO rds_output
SELECT
  a AS id,
  SESSION_START (ts, INTERVAL '1' SECOND) AS win_start,
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,
  COUNT (a) AS cnt
FROM
  mq_stream
GROUP
  BY SESSION (ts, INTERVAL '1' SECOND),
  a
```

Expiration of time attributes

The time attribute of fields no longer takes effect after one of the following operations is completed:

- GROUP BY operations on the fields that are not defined as time attribute fields, except the GROUP BY operations in tumbling, sliding, and session windows. For more information, see TUMBLE, HOP, and 会话窗口.
- JOIN operations on two data streams.

  > **Note** For more information, see JOIN statements.

- MATCH_RECOGNIZE operations in complex event processing (CEP) statements. For more information, see CEP statements.
- PARTITION BY operations in OVER windows. For more information, see OVER window.
- UNION operations. UNION is equivalent to the combination of RETRACT and UNION ALL.

If you use time-based window functions for computing after the preceding operations, errors are returned, such as `org.apache.flink.table.api.ValidationException: Window can only be defined over a time attribute column.`

2.3.2. Watermark

Realtime Compute for Apache Flink aggregates data by using window functions based on the time attribute. To use window functions based on the event time of a job, you must define a watermark when you declare a source table.

A watermark is used to measure the progress of Event Time. It is a hidden data attribute. A watermark is defined in the DDL statement of the source table. Flink provides the following statement to define a watermark:

```
WATERMARK [watermarkName] FOR <rowtime_field> AS withOffset(<rowtime_field>, offset)
```

> **Note** For more information about the time attributes of Realtime Compute for Apache Flink, see Time attributes.

Parameter	Required	Description
watermarkName	No	The name of the watermark.
<rowtime_field>	Yes	The column used to generate the watermark. <rowtime_field> is identified as the event time column and must be a column of the TIMESTAMP type defined in the table. You can use <rowtime_field> to define a window in the job code.
withOffset	Yes	The policy to generate a watermark. In this example, the watermark value is generated by using the following formula: `<rowtime_field> - offset`. The first parameter in `withOffset` must be <rowtime_field>.
offset	Yes	The offset between the watermark value and event time, in milliseconds.

A specific field in a data record indicates the time when the record was generated. For example, a table contains a `rowtime` field whose data type is TIMESTAMP, and one of its values is `1501750584000 (2017-08-03 08:56:24.000)`. If you want to define a watermark based on the `rowtime` field and configure a 4-second offset, add the following definition:

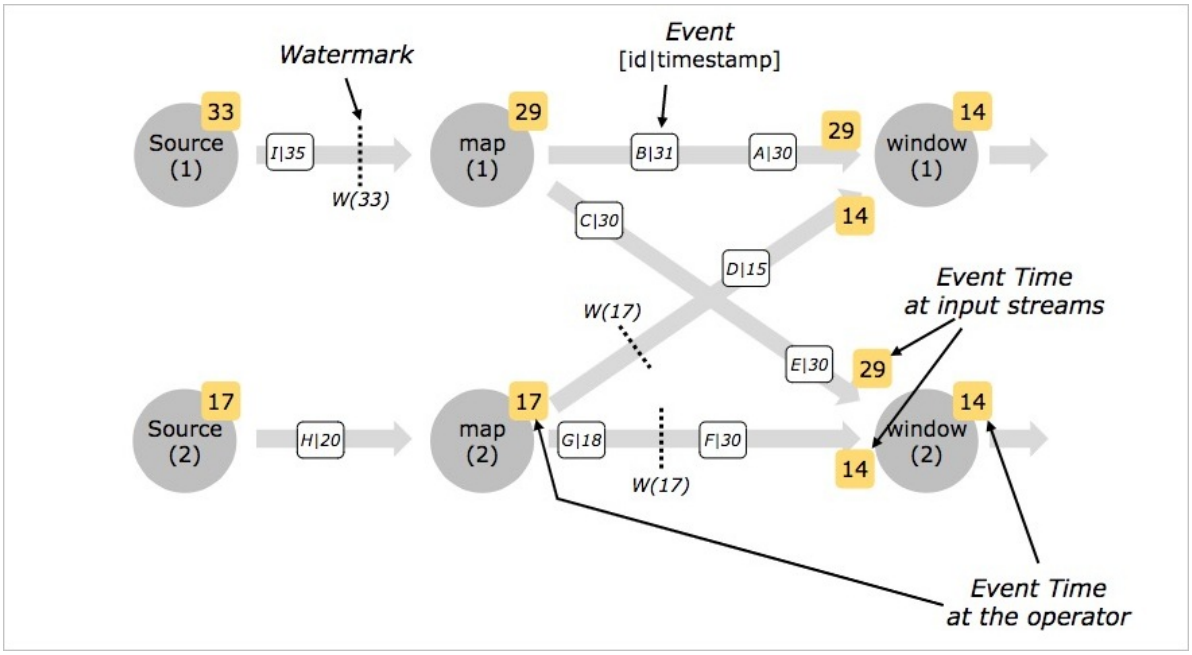
```
WATERMARK FOR rowtime AS withOffset(rowtime, 4000)
```

In this example, the watermark time of the data record is `1501750584000 - 4000 = 1501750580000 (2017-08-03 08:56:20.000)`. This means that all data whose timestamp is earlier than `1501750580000 (2017-08-03 08:56:20.000)` has arrived.

> **Note**
>
> - If you use an event time-based watermark, the `rowtime` field must be of the TIMESTAMP type. Realtime Compute for Apache Flink supports 13-digit UNIX timestamps in milliseconds. If the rowtime field is of another type or the UNIX timestamp is not 13 digits in length, we recommend that you use a computed column to convert the time. For more information, see Computed column.
> - The event time and processing time can only be declared in the source table. For more information, see Event Time and Processing Time.

Summary

- A watermark indicates that all the events whose timestamp t' is earlier than the watermark time t (`t '< t`) have occurred. After the watermark time `t` takes effect, all subsequently received data records whose event time is earlier than `t` are discarded. Realtime Compute for Apache Flink will allow you to change the configuration and update the subsequent data.
- Watermarks are important for data streams that arrive out of order because the watermarks help ensure that the computing in a window is correct even if some events arrive late.
- If an operator has multiple input data streams for parallel processing, the event time of the data stream with the shortest time is used as the event time of the operator.



2.3.3. Computed column

You can calculate a value for a computed column by using data from other columns. If your source table does not have a column of the TIMESTAMP type, you can use a computed column to convert a field of another type to the TIMESTAMP type.

Concept

A computed column is a virtual column that is not stored in a physical table. You can create computed columns by using expressions, built-in functions, or user-defined extensions (UDXs). In Flink SQL, a computed column can be used the same as columns that are stored in a physical table.

Usage

Currently, the event time (also known as rowtime) column in a Watermark must be of the TIMESTAMP type. The LONG data type will be supported in the future. You can only define a watermark in the DDL statement of a source table. If a source table does not have a column of the TIMESTAMP type, you can use a computed column to convert a field of another type to the TIMESTAMP type.

Syntax

```
column_name AS computed_column_expression
```

Example

The rowtime column in a watermark must be of the TIMESTAMP type. Currently, Realtime Compute only supports 13-bit UNIX timestamps measured in milliseconds. If the TIME column in a DataHub source table is defined as a 16-bit UNIX timestamp measured in microseconds, you can use a computed column to convert the 16-bit UNIX timestamp to a 13-bit UNIX timestamp. The sample code is as follows:

```
CREATE TABLE test_stream(
  a INT,
  b BIGINT,
  `TIME` BIGINT,
  ts AS TO_TIMESTAMP(`TIME`/1000), -- Use a computed column to convert 16-bit timestamps to
13-bit timestamps.
  WATERMARK FOR ts AS WITHOFFSET(ts, 1000)
) WITH (
  type = 'datahub',
  ...
);
```

The `` `TIME` `` field in the source table contains the date and time information. The value of this field is of the BIGINT type. A computed column is created to convert the `TIME` column of the BIGINT type to the `ts` column of the TIMESTAMP type. The `ts` column is used as the rowtime of a watermark.

2.4. Data types

2.4.1. Data types

This topic describes data types supported by Realtime Compute.

Supported data types

Data type	Description	Value range
VARCHAR	Character string with a changeable length	Maximum storage capacity: 4 MB

Data type	Description	Value range
BOOLEAN	Logical value	Valid values: TRUE, FALSE, and UNKNOWN
TINYINT	Tiny integer (one byte)	-128 to 127
SMALLINT	Small integer (two bytes)	-32768 to 32767
INT	Integer (four bytes)	-2147483648 to 2147483647
BIGINT	Big integer (eight bytes)	-9223372036854775808 to 9223372036854775807
FLOAT	Four-byte floating point number	Six digits of precision
DECIMAL	Decimal	An example value for `DECIMAL(5,2)` is `123.45`.
DOUBLE	Eight-byte floating point number	15 decimal digits of precision
DATE	Date	Example: `DATE'1969-07-20'`
TIME	Time	Example: `TIME '20:17:40'`
TIMESTAMP	Timestamp containing both date and time parts	Example: `TIMESTAMP '1969-07-20 20:17:40'`
VARBINARY	Binary data	Byte array

Type conversion

Convertible	Data Type										
	VARCHAR	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	DATE	TIMESTAMP	TIME
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
BOOLEAN	Y	Y	N	N	N	N	N	N	N	N	N
TINYINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
SMALLINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
INT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
BIGINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
FLOAT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DOUBLE	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DATE	Y	N	N	N	N	N	N	N	Y	Y	N
TIMESTAMP	Y	N	N	N	N	N	N	N	Y	Y	Y
TIME	Y	N	N	N	N	N	N	N	N	Y	Y

Type conversion example

Input example

var1 (VARCHAR)	big1 (BIGINT)
1000	323

Sample code

```
CAST(var1 AS BIGINT) AS AA;
CAST(big1 AS VARCHAR) AS BB;
```

Output example

AA (BIGINT)	BB (VARCHAR)
1000	323

2.4.2. Data types and operators

This topic describes data types and operators in Realtime Compute.

You can perform mathematical and logical operations for different data types. Examples are provided in the following table.

Operator	Description	Data types supported by numeric1 and numeric2	Example
numeric1 + numeric2	Addition	INT, DOUBLE, DECIMAL, BIGINT	`2 + 4.2`
numeric1 - numeric2	Subtraction		`3 - 5.3`
numeric1 * numeric2	Multiplication		`2 * 4`
numeric1/numeric2	Division		2.4/5
numeric1 > numeric2	>		`2.4 > 5`
numeric1 < numeric2	<		`2.4 < 5`
numeric1 >= numeric2	>=		`2.4 >= 5`
numeric1 <= numeric2	<=		`2.4 <= 5`
numeric1 = numeric2	=	INT, DOUBLE, DECIMAL,	`'iphone' = 5`

Operator	Description	BIGINT, VARCHAR Data types supported by numeric1 and numeric2	Example
numeric1 <> numeric2	<>		'iphone' <> 5

2.5. DDL statements

2.5.1. Overview

Syntax

```
CREATE TABLE tableName
     (columnName dataType [, columnName dataType ]*)
     [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

Description

Realtime Compute does not store data from source and result tables. You can only perform operations on source and result tables by using DDL statements to declare references to data stores. Example:

```
create table tt_stream(
    a varchar,
    b varchar,
    c varchar
    ) with (
    type='DataHub',
    topic='blink_tt_test',
    accessId='yourAccessId',
    accessKey='yourAccessSecret'
    );
```

This Flink SQL statement is not used to create a DataHub topic, but to create a table named tt_stream in Realtime Compute and use the table to declare a reference to a DataHub topic. You can use tt_stream as an alias in subsequent DML statements to perform operations on the table.

- A declaration of a table (tt_stream in this example) is valid only for the jobs generated from the current SQL file. A job is generated after an SQL file is published. The table can also be declared for reference in other SQL files under the same project.
- Since Realtime Compute adopts standard SQL, keywords, table names, and field names in DDL statements are not case-sensitive.
- Table and field names must start with a letter or digit and can contain only letters, digits, and underscores (_).
- The method of mapping fields between a source table and an upstream data store depends on the feature of the data store. If the data store such as DataHub supports mapping based on key values, the field names in the source table must be the same as those in the data store. However, the number of fields can be different. If a data store does not support mapping based on key values, both the number and sequence of fields defined in the source table must the same as those in the data store. We recommend that you use the same field names and the same number of fields in DDL

statements as those in external data stores. This can prevent data errors caused by inaccurate definitions.

Mappings

Data stores are classified into two types based on whether the data store has a schema.

- Sequence mapping

  Systems such as Message Queue (MQ) are non-structured storage systems that do not have a schema. You can customize field names in DDL statements. However, the data types and the number of fields must be the same as those in the data store.

  The following example shows an MQ record:

  ```
  asavfa,sddd32,sdfdsv
  ```

  You must set field names of the tt_stream table based on naming conventions.

  ```
  create table tt_stream(
        a varchar,
        b varchar,
        c varchar
        ) with (
        type='mq',
        topic='blink_mq_test',
        accessId='yourAccessId',
        accessKey='yourAccessSecret'
        );
  ```

- Name mapping

  For structured storage systems such as DataHub, field names and data types are defined based on a table schema. We recommend that you define fields by using Flink SQL based on the table schema. The names, number, and sequence of fields must be the same as those in the data store.

  > **Note** If field names in the data store (such as Tablestore) are case-sensitive, you must enclose field names in backticks (`) in Flink SQL statements. DataHub is used in this example. In DDL statements, field names of the declared source table must apply the same casing that DataHub uses.

  The following table describes a sample schema of DataHub.

Sample schema of DataHub

Field	Data type
name	varchar
age	bigint
value	varchar

We recommend that you declare references for all fields. Note that you can reference only the fields that are included in the data store. The following code shows a sample of the DDL statement:

```
create  table stream_result (
     name varchar,
     age bigint,
     value varchar
     ) with (
     type='DataHub',
     endpoint='http://dh-cn-hangzhou.aliyuncs.com',
     accessId='yourAccessId',
     accessKey='yourAccessSecret',
     project='project',
     topic='topic'
     );
```

Case insensitivity

Realtime Compute adopts standard SQL. Therefore, fields are not case-sensitive. For example, the following two statements produce the same effect. Statement 1:

```
create  table stream_result (
    name            varchar,
    value           varchar
    );
```

Statement 2:

```
create  table STREAM_RESULT (
     NAME            varchar,
     VALUE           varchar
     );
```

Realtime Compute references data from multiple data stores. Some data stores, such as Tablestore, are case-sensitive in field names. If a NAME parameter is defined in Tablestore, use the following DDL statement:

```
create  table STREAM_RESULT (
    `NAME`          varchar,
    `VALUE`         varchar
    );
```

In all subsequent DML statements, the parameter must be enclosed in backticks (`) if it is referenced. The following code shows an example:

```
INSERT INTO Table_A
    SELECT
    `NAME`,
    `VALUE`
    FROM
    Table_B;
```

2.5.2. Create a data source table

2.5.2.1. Overview

Source tables for Realtime Compute store streaming data. Streaming data triggers stream processing jobs in Realtime Compute. To process streaming data, you must create at least one source table that provides streaming data for Realtime Compute jobs.

Syntax

```
  CREATE TABLE tableName
      (columnName dataType [, columnName dataType ]*)
      [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

Example

```
CREATE TABLE  datahub_stream(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
) WITH (
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessKey>',
  startTime='2017-07-21 00:00:00'
);
```

```
CREATE TABLE metaq_stream(
 x VARCHAR,
 y VARCHAR,
 z VARCHAR
) WITH (
 type='mq',
 topic='<yourTopicName>',
 endpoint='<yourEndpointName>',
 pullIntervalMs='1000',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 startMessageOffset='1000',
 consumerGroup='<yourConsumerGroup>',
 fieldDelimiter='|'
);
```

Retrieve attribute fields from a source table

- Syntax

  Realtime Compute provides the HEADER keyword in the data definition language (DDL) statement of a source table for you to retrieve the attribute fields from the source table.

```
CREATE TABLE sourcetable
(
 `timestamp`  VARCHAR HEADER,
  name        VARCHAR,
  MsgID       VARCHAR
)WITH(
     type='RDS'
);
```

In this example, the `'timestamp'` field is defined as `HEADER`. Realtime Compute reads the values of attribute fields from the source table. Then, the 'timestamp' field is used as a common field.

> **Note** The default attribute fields vary based on the source table type, such as DataHub, Log Service, and Message Queue (MQ). You can customize attribute fields for certain types of source tables. For more information, see the topics about source tables of the related type.

- Example

  A Log Service source table is used as an example to explain how to retrieve attribute fields. A Log Service source table has three attribute fields listed in the following table.

Field	Description
`source`	The message source.
`topic`	The message topic.
`timestamp`	The time when a data record enters Log Service.

> **Note** To obtain attribute fields of a source table, you must add the `HEADER` keyword to the end of a field declaration.

  Example:

  - Test data

```
__topic__:  ens_altar_flow
        result:  {"MsgID":"ems0a","Version":"0.0.1"}
```

- Test statements

```
CREATE TABLE sls_log (
  __topic__  VARCHAR HEADER,
  result     VARCHAR
)WITH(
  type ='sls'
);
CREATE TABLE sls_out (
  name     varchar,
  MsgID    varchar,
  Version  varchar
)WITH(
  type ='RDS'
);
INSERT INTO sls_out
SELECT
__topic__,
JSON_VALUE(result,'$.MsgID'),
JSON_VALUE(result,'$.Version')
FROM
sls_log
```

- Test results

name(VARCHAT)	MsgID(VARCHAT)	Version(VARCHAT)
ens_altar_flow	ems0a	0.0.1

Source tables for window function-based computing

Realtime Compute can aggregate data by using window functions based on the event time or processing time. If window functions are used for a stream processing job, you must define the watermark and computed column in the DDL statement of a source table. For more information about the watermark and computed column, see Watermark and Computed column. For more information about data aggregation based on time attributes, see Time attributes.

2.5.2.2. Create a DataHub source table

This topic describes how to create a DataHub source table in Realtime Compute for Apache Flink. It also describes the attribute fields, parameters in the WITH clause, and data type mapping involved when you create a DataHub source table.

> **Notice** This topic applies only to Blink 1.4.5 and later.

Introduction to DataHub

DataHub is a data distribution platform designed to process streaming data. You can publish and subscribe to applications for streaming data in DataHub and distribute the data to other platforms. DataHub allows you to analyze streaming data and build applications based on the streaming data. Realtime Compute for Apache Flink uses DataHub to store input and output data for the processing of streaming data.

Syntax

In Realtime Compute for Apache Flink, you can use DataHub to store input data. The following code shows an example:

```
CREATE TABLE datahub_stream(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2017-07-21 00:00:00'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to `datahub`.
endPoint	The endpoint of DataHub.	Yes	Contact Alibaba Cloud O&M engineers.
accessId	AccessKey ID	Yes	None.
accessKey	AccessKey Secret	Yes	None.
project	A project in DataHub.	Yes	None.
topic	A topic of the project.	Yes	None. **Notice** Topics support only the data records of the tuple format.
startTime	The start time of DataHub logs.	No	This parameter is required for batch processing. The format of the value is `yyyy-MM-dd hh:mm:ss`.
endTime	The end time of DataHub logs.	No	This parameter is required for batch processing. The format of the value is `yyyy-MM-dd hh:mm:ss`.

Parameter	Description	Required	Remarks
maxRetryTimes	The maximum number of retries for reading data from DataHub.	No	The default value of this parameter varies based on Blink versions. • For versions earlier than Blink 2.2.7, the default value is 3. • For Blink 2.2.7 and later, the default value is 20.
retryIntervalMs	The interval of retries.	No	Unit: milliseconds. The default value of this parameter varies based on Blink versions. • For versions earlier than Blink 2.2.7, the default value is 1000. • For Blink 2.2.7 and later, the default value is 50.
batchReadSize	The number of data records that are read at a time.	No	Default value: 10. Maximum value: 1000.
lengthCheck	The policy for checking the number of fields parsed from a row of data.	No	• Default value: NONE ◦ If the number of fields parsed from a row of data is greater than the number of defined fields, data is extracted from left to right based on the order of defined fields. ◦ If the number of fields parsed from a row of data is less than the number of defined fields, this row of data is skipped. • SKIP: If the number of fields parsed from a row of data is different from the number of defined fields, this row of data is skipped. • EXCEPTION: If the number of fields parsed from a row of data does not match the number of defined fields, an exception is reported. • PAD: Data is padded from left to right based on the order of defined fields. Note ◦ If the number of fields parsed from a row of data is greater than the number of defined fields, data is padded from left to right based on the order of defined fields. ◦ If the number of fields parsed from a row of data is less than the number of defined fields, the values of the missing fields are padded with null from left to right.

Parameter	Description	Required	Remarks
columnErrorDebug	Specifies whether debugging is enabled.	No	• false (default value): Debugging is disabled. • true: Debugging is enabled, and logs that contain parsing exceptions are printed.
isBlob	Specifies whether data of DataHub is of the BLOB type.	No	Default value: false. **Notice** • Only Blink 3.4.X and later versions support this parameter. • When you use the BLOB type, you must declare fields as VARBINARY, which is similar to METAQ.

Field type mapping

The following table lists the mapping between DataHub data types and data types of Realtime Compute for Apache Flink. We recommend that you declare the field type mapping in a DDL statement.

DataHub data type	Data type of Realtime Compute for Apache Flink
BIGINT	BIGINT
TIMESTAMP	
STRING	VARCHAR
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL

Attribute field

You can use Flink SQL to retrieve the attribute field TIMESTAMP of a DataHub source table. You can obtain the system time at which each data record is written into DataHub after the attribute field is read.

Field	Description
TIMESTAMP	The system time at which each data record is written into DataHub.

Shard ID	System Time	name (STRING)	place (STRING)
0	2019-02-15 11:11:46	/abC[Q7	beijing
0	2019-02-15 11:11:46	@"'G%T[	guangzhou
0	2019-02-15 11:11:46	XvoedT`	guangzhou
0	2019-02-15 11:11:46	%:lHj+O	guangzhou

Sample code

The following sample code demonstrates how to create a DataHub source table in a Realtime Compute for Apache Flink job.

```
create table datahub_input(
  name VARCHAR
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='test1',
  topic='topic1',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2018-06-01 00:00:00'
);
create table test_out(
 name  VARCHAR
) with (
  type='print'
);
INSERT INTO test_out
SELECT
  LOWER(name)
from datahub_input;
```

FAQ

- Q: How do I convert TIMESTAMP values between DataHub and Realtime Compute for Apache Flink?

  A: In DataHub, TIMESTAMP is a 16-bit UNIX timestamp that is accurate to microseconds. However, in Realtime Compute for Apache Flink, TIMESTAMP is 13-bit UNIX timestamp that is accurate to milliseconds. We recommend that you use BIGINT for mapping. If you need to use TIMESTAMP, we recommend that you use computed columns to perform the conversion. For more information, see Computed column. The following sample code shows the conversion.

```
CREATE TABLE datahub_test (
  id                 VARCHAR,
  `type`             VARCHAR,
  `value`            DOUBLE,
  `time`             BIGINT,
   ts as TO_TIMESTAMP(`time`/1000),
   WATERMARK wk for ts as withoffset(ts,2000)
) WITH (
  type = 'datahub'
);
```

- Q: After a DataHub topic was split or scaled in, a Realtime Compute for Apache Flink job failed to run. How do I restore the job?

  A: Blink versions earlier than 2.2.0 do not support the DataHub shard scaling feature. If a topic that Realtime Compute for Apache Flink is reading is split or scaled in, the job fails and cannot be automatically restored. To restore the job, you must stop and restart it.

- Q: Can I delete a DataHub topic in use?

  A: No, you cannot delete or recreate DataHub topics that are being referenced in Realtime Compute for Apache Flink.

2.5.2.3. Create a Message Queue for Apache Kafka source table

This topic describes how to create a Message Queue for Apache Kafka source table in Realtime Compute for Apache Flink. It also describes the mappings between the values of the type parameter and Kafka versions, and provides examples on how to parse messages of Message Queue for Apache Kafka.

> **Notice**
>
> - This topic applies only to Blink 2.0 and later.
> - This topic applies only to Realtime Compute for Apache Flink that is deployed in exclusive mode.
> - You can use Realtime Compute for Apache Flink to read data from a source table of a self-managed Kafka cluster. Before data is read, you must take note of the mappings between the values of the type parameter and Kafka versions, and the network configurations of the self-managed Kafka cluster and your Realtime Compute for Apache Flink cluster.
> - You cannot perform local debugging on binary data. If the binary data passes the syntax check, you can perform online debugging on the binary data. For more information, see Online debugging.

Introduction to Message Queue for Apache Kafka source tables

Message Queue for Apache Kafka is a distributed, high-throughput, and scalable message queue service provided by Alibaba Cloud. Message Queue for Apache Kafka is widely used in big data scenarios, such as log collection, monitoring data aggregation, streaming data processing, and online and offline analysis. Realtime Compute for Apache Flink can use Message Queue for Apache Kafka tables as source tables or result tables to process streaming data.

The output data of Message Queue for Apache Kafka is of the serialized VARBINARY type. For each data record obtained from a Message Queue for Apache Kafka source table, you must write a user-defined table-valued function (UDTF) to parse the data into a data structure before serialization. Realtime Compute for Apache Flink first extracts data from a Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to a sink. Flink SQL also allows you to use the CAST function to parse data of the VARBINARY type into data of the VARCHAR type. For more information about UDTFs, see UDTF.

DDL syntax

The DDL definition of a Message Queue for Apache Kafka source table must be the same as the DDL definition in the following SQL statement. You can modify the settings of the parameters in the WITH clause.

```
create table kafka_stream(   --The sequence and data types of the following fields must be
the same as the five fields in the Message Queue for Apache Kafka source table.
  messageKey VARBINARY,
  `message`    VARBINARY,
  topic      VARCHAR,
  `partition`  INT,
  `offset`     BIGINT
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...
);
```

Parameters in the WITH clause

- General configurations

Parameter	Description	Required	Remarks
type	The Kafka version.	Yes	Valid values: Kafka08, Kafka09, Kafka010, and Kafka011. For more information about the mappings between the values of the type parameter and Kafka versions, see Mappings between the values of the type parameter and Kafka versions.
topic	The name of the topic to read.	Yes	N/A.
topicPattern	The regular expression used to read multiple topics.	No	Topics are separated by vertical bars (|), such as `topic1|topic2|topic3`. For more information, see Class Pattern.

Parameter	Description	Required	Remarks
startupMode	The start offset for reading data.	No **Note** We recommend that you set this parameter based on your business requirements.	Valid values: • GROUP_OFFSETS: reads data by group. This is the default value. • EARLIEST: reads data from the earliest offset in the Kafka cluster. • LATEST: reads data from the latest offset in the Kafka cluster. • TIMESTAMP: reads data from a specific point in time. You can specify the startTimeMs or startTime parameter in the WITH clause. The value of the startTimeMs parameter is a timestamp, whereas the value of the startTime parameter is in the `yyyy-MM-dd HH:mm:ss` format. We recommend that you use the startTimeMs parameter. If neither of the parameters is specified, data is consumed from the start offset specified by Realtime Compute for Apache Flink. **Note** • If you do not specify this parameter in plaintext, GROUP_OFFSETS is used by default. • If you set this parameter to TIMESTAMP, you must specify the time zone in job parameters in plaintext, such as `blink.job.timeZone=Asia/Shanghai`. • In GROUP_OFFSETS mode, if no offset is configured, data is read from the end of the Kafka partition when a group consumes Kafka data for the first time. • Only Kafka010 and Kafka011 support the TIMESTAMP data type.

Parameter	Description	Required	Remarks
partitionDiscoveryIntervalMS	The interval at which new partitions are checked.	No	◦ Kafka08: By default, new partitions are checked at a specific interval. ◦ Kafka09 or later: The partitionDiscoveryIntervalMS parameter is not supported. You can specify extraConfig='flink.partition-discovery.interval-millis=60000' in the WITH clause to achieve the same effect as the partitionDiscoveryIntervalMS parameter. Default value: 60000. Unit: milliseconds.
extraConfig	Additional KafkaConsumer configuration items.	No	You can use this parameter to add configuration items that are required in special scenarios but are not included in the optional configuration items. Example: `'fetch.message.max.bytes=104857600'`. Separate multiple configuration items with semicolons (;).

- Configurations for Kafka08
  - Required configurations

Parameter	Description	Required
group.id	The ID of the consumer group.	Yes
zookeeper.connect	The ZooKeeper URL.	Yes

- Optional configurations
  - consumer.id
  - socket.timeout.ms
  - fetch.message.max.bytes
  - num.consumer.fetchers
  - auto.commit.enable
  - auto.commit.interval.ms
  - queued.max.message.chunks
  - rebalance.max.retries
  - fetch.min.bytes
  - fetch.wait.max.ms
  - rebalance.backoff.ms
  - refresh.leader.backoff.ms
  - auto.offset.reset
  - consumer.timeout.ms
  - exclude.internal.topics
  - partition.assignment.strategy
  - client.id
  - zookeeper.session.timeout.ms
  - zookeeper.connection.timeout.ms
  - zookeeper.sync.time.ms
  - offsets.storage
  - offsets.channel.backoff.ms
  - offsets.channel.socket.timeout.ms
  - offsets.commit.max.retries
  - dual.commit.enabled
  - partition.assignment.strategy
  - socket.receive.buffer.bytes
  - fetch.min.bytes

- Configurations for Kafka09, Kafka010, and Kafka011
  - Required configurations

Parameter	Description
group.id	The ID of the consumer group.
bootstrap.servers	The endpoint of the Message Queue for Apache Kafka cluster.

- For more information about the optional configurations for Kafka09, Kafka010, and Kafka011, see the following Kafka documentation:
  - Kafka09
  - Kafka010
  - Kafka011

  If you want to modify the configurations, you can add parameters to the WITH clause in the DDL statement. For example, if you want to configure Simple Authentication and Security Layer (SASL), add the `security.protocol`, `sasl.mechanism`, and `sasl.jaas.config` parameters.

```
create table kafka_stream(
  messageKey varbinary,
  `message` varbinary,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...,
  `security.protocol`='SASL_PLAINTEXT',
  `sasl.mechanism`='PLAIN',
  `sasl.jaas.config`='org.apache.kafka.common.security.plain.PlainLoginModule required
username="<yourUserName>" password="<yourPassword>";'
);
```

Mappings between the values of the type parameter and Kafka versions

type	Kafka version
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2 and later

Examples on how to parse messages of Message Queue for Apache Kafka

- Scenario 1: Realtime Compute for Apache Flink processes the data read from Message Queue for Apache Kafka and exports the result data to ApsaraDB RDS.

  Data stored in Message Queue for Apache Kafka is in the JSON format and must be processed by using Realtime Compute for Apache Flink. The following code shows the message format:

```
{
  "name":"Alice",
  "age":13,
  "grade":"A"
}
```

- Data processing method 1: Realtime Compute for Apache Flink reads and processes data from the Message Queue for Apache Kafka source table and then exports the result data to ApsaraDB RDS.

  In Blink 2.2.7 and later, you can use the CAST function to convert the VARBINARY data type into the VARCHAR data type. Then, use the JSON_VALUE function to parse the data of the Message Queue for Apache Kafka source table. The following code shows an example:

```
CREATE TABLE kafka_src (
  messageKey  VARBINARY,
  `message`   VARBINARY,
  topic       VARCHAR,
  `partition` INT,
  `offset`    BIGINT
) WITH (
  type = 'kafka010', -- For more information, see Mappings between the values of the ty
pe parameter and Kafka versions.
);
CREATE TABLE rds_sink (
  `name`       VARCHAR,
  age          VARCHAR,
  grade        VARCHAR
) WITH(
  type='rds'
);
CREATE VIEW input_view AS
  SELECT CAST(`message` as VARCHAR ) as `message`
FROM kafka_src;
INSERT INTO rds_sink
SELECT
  JSON_VALUE(`message`,'$.name'),
  JSON_VALUE(`message`,'$.age'),
  JSON_VALUE(`message`,'$.grade')
FROM input_view;
```

- Data processing method 2: Realtime Compute for Apache Flink extracts data from the Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to ApsaraDB RDS.

  To parse irregular data or complex JSON data, you must write UDTF code. Examples:

  - SQL

```
-- Define a UDTF to parse messages of Message Queue for Apache Kafka.
CREATE FUNCTION kafkaparser AS 'com.alibaba.kafkaUDTF';
-- Define a Message Queue for Apache Kafka source table. Note that the fields declare
d in the DDL statement of the Message Queue for Apache Kafka source table must be the
same as the fields in the following example. You can modify the settings of the param
eters in the WITH clause.
CREATE TABLE kafka_src (
  messageKey  VARBINARY,
```

```
  `message`   VARBINARY,
  topic       VARCHAR,
  `partition` INT,
  `offset`    BIGINT
) WITH (
  type = 'kafka010', -- For more information, see Mappings between the values of the
type parameter and Kafka versions.
  topic = 'test_kafka_topic',
  `group.id` = 'test_kafka_consumer_group',
  bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);
CREATE TABLE rds_sink (
  name       VARCHAR,
  age        INT,
  grade      VARCHAR,
  updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourDatabasePassword>'
);
-- Use a UDTF to parse data of the VARBINARY type into formatted data.
CREATE VIEW input_view (
  name,
  age,
  grade,
  updateTime
) AS
SELECT
  T.name,
  T.age,
  T.grade,
  T.updateTime
FROM
  kafka_src as S,
  LATERAL TABLE (kafkaparser (`message`)) as T (
  name,
  age,
  grade,
  updateTime
);
-- Compute the formatted data and export the result data to ApsaraDB RDS.
INSERT INTO rds_sink
SELECT
  name,
  age,
  grade,
  updateTime
FROM input_view;
```

- UDTF

For more information about how to create a UDTF, see UDTF. The following example shows Maven dependencies of Blink 2.2.4.

```
    <dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java_2.11</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table_2.11</artifactId>
            <version>blink-2.2.4-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.9</version>
        </dependency>
    </dependencies>
```

```
package com.alibaba;
import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
import java.sql.Timestamp;
public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
            /* input message :
                {
                  "name":"Alice",
                  "age":13,
                  "grade":"A",
                  "updateTime":1544173862
                }
            */
            String msg = new String(message, "UTF-8");
            try {
                JSONObject data = JSON.parseObject(msg);
                if (data != null) {
                    String name = data.getString("name") == null ? "null" : data.getS
tring("name");
```

```
                    Integer age = data.getInteger("age") == null ? 0 : data.getIntege
r("age");
                    String grade = data.getString("grade") == null ? "null" : data.ge
tString("grade");
                    Timestamp updateTime = data.getTimestamp("updateTime");
                    Row row = new Row(4);
                    row.setField(0, name);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3,updateTime );
                    System.out.println("Kafka message str ==>" + row.toString());
                    collect(row);
                }
            } catch (ClassCastException e) {
                System.out.println("Input data format error. Input data " + msg + "is
not json string");
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
    @Override
    // If the return value is declared as a row, you must reload the UDTF method and
specify the types of fields to be returned.
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.STRING, DataTypes.INT, DataTypes.STR
ING, DataTypes.TIMESTAMP);
    }
}
```

- Scenario 2: Realtime Compute for Apache Flink reads data from a Message Queue for Apache Kafka source table and processes the data by using window functions.

  You must define watermarks in the DDL statement of a source table for windows, such as tumbling and sliding windows, based on the design of Realtime Compute for Apache Flink. For more information, see Watermark. The method you use to define a watermark in a Message Queue for Apache Kafka source table is different from the method you use for other types of source tables. If you want to perform an event time-based computation by using a window function, you must use a user-defined extension (UDX) to parse the event time in the message field of a source table. Then, you can define a watermark based on the parsed event time. You must use a computed column to convert data types for the event time parsed from a Message Queue for Apache Kafka source table. For example, the data `2018-11-11 00:00:00|1|Anna|female` is written to the Message Queue for Apache Kafka source table. During the computing process, Realtime Compute for Apache Flink extracts data from the Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to ApsaraDB RDS.

- Data processing method 1: Realtime Compute for Apache Flink reads and processes data from the Message Queue for Apache Kafka source table and then exports the result data to ApsaraDB RDS.

  In Blink 2.2.7 and later, you can use the CAST function to convert the VARBINARY data type into the VARCHAR data type. Then, use the JSON_VALUE function to parse the data of the Message Queue for Apache Kafka source table. The following code shows an example:

```
CREATE TABLE kafka_src (
  messageKey VARBINARY,
  `message` VARBINARY,
  topic VARCHAR,
  `partition` INT,
  `offset` BIGINT,
  ts as to_timestamp(json_value(cast(`message` as VARCHAR ),'$.nodes.time')),
  WATERMARK wk FOR ts as withOffset(ts, 2000)
) WITH (type = 'kafka' -- For more information, see Mappings between the values of the
type parameter and Kafka versions.
);
CREATE TABLE rds_sink (
  starttime TIMESTAMP ,
  endtime   TIMESTAMP ,
  `message` BIGINT
) WITH (type = 'rds');
INSERT
  INTO rds_sink
SELECT
  TUMBLE_START(ts, INTERVAL '1' MINUTE),
  TUMBLE_END(ts, INTERVAL '1' MINUTE),
  count(`message`)
FROM
  kafka_src
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE);
```

- Data processing method 2: Realtime Compute for Apache Flink extracts data from the Message Queue for Apache Kafka source table, writes a UDTF to parse the data, and then exports the result data to ApsaraDB RDS.
  - SQL

```
-- Define a UDTF to parse messages of Message Queue for Apache Kafka.
CREATE FUNCTION kafkapaser AS 'com.alibaba.kafkaUDTF';
CREATE FUNCTION kafkaUDF AS 'com.alibaba.kafkaUDF';
-- Define a Message Queue for Apache Kafka source table. Note that the fields defined
in the DDL statement must be the same as the fields in the following statement. You c
an modify the settings of the parameters in the WITH clause.
create table kafka_src (
  messageKey VARBINARY,
  `message` VARBINARY,
  topic VARCHAR,
  `partition` INT,
  `offset` BIGINT,
  ctime AS TO_TIMESTAMP(kafkaUDF(`message`)), -- Define a computed column. A computed
column can be considered as a placeholder column that is not stored in a source table
. The values in this column are computed. If you want to define a watermark, the data
type of the computed column must be TIMESTAMP.
  watermark for `ctime` as withoffset(`ctime`, 0) -- Define a watermark in a computed
```

```
  watermark for `ctime` as withoffset( `ctime` ,0)    -- Define a watermark in a computed
column.
) WITH (
  type = 'kafka010', -- For more information, see Mappings between the values of the
type parameter and Kafka versions.
  topic = 'test_kafka_topic',
  `group.id` = 'test_kafka_consumer_group',
  bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);
create table rds_sink (
  `name` VARCHAR,
  age INT,
  grade VARCHAR,
  updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourPassword>'
);
-- Use a UDTF to parse data of the VARBINARY type to formatted data.
CREATE VIEW input_view AS
SELECT
  S.ctime,
  T.`order`,
  T.`name`,
  T.sex
  from
  kafka_src as S,
  LATERAL TABLE (kafkapaser (`message`)) as T (
  ctime,
  `order`,
  `name`,
  sex
);
-- Compute the data in input_view.
CREATE VIEW view2 (
  cnt,
  sex
) AS
  SELECT
  COUNT(*) as cnt,
  T.sex
  from
  input_view
Group BY sex, TUMBLE(ctime, INTERVAL '1' MINUTE);
-- Compute the formatted data and export the result data to ApsaraDB RDS.
insert into rds_sink
  SELECT
  cnt,sex
from view2;
```

- UDF&UDTF

For more information about how to create UDFs and UDTFs, see UDF and UDTF. The following example shows Maven dependencies of Blink 2.2.4.

```
<dependencies>
      <dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-core</artifactId>
          <version>blink-2.2.4-SNAPSHOT</version>
          <scope>provided</scope>
      </dependency>
      <dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-streaming-java_2.11</artifactId>
          <version>blink-2.2.4-SNAPSHOT</version>
          <scope>provided</scope>
      </dependency>
      <dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-table_2.11</artifactId>
          <version>blink-2.2.4-SNAPSHOT</version>
          <scope>provided</scope>
      </dependency>
      <dependency>
          <groupId>com.alibaba</groupId>
          <artifactId>fastjson</artifactId>
          <version>1.2.9</version>
      </dependency>
  </dependencies>
```

- UDTF

```
package com.alibaba;
import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
/**
  The following example shows how to parse the JSON strings in a Message Queue for
Apache Kafka source table and format the parsed data.
**/
public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
          // Read data of the VARBINARY data type and convert the data into the STR
ING data type.
            String msg = new String(message, "UTF-8");
                // Extract data from JSON strings based on the following fields:
                    String ctime = Timestamp.valueOf(data.split('\\|')[0]);
                    String order = data.split('\\|')[1];
                    String name = data.split('\\|')[2];
                    String sex = data.split('\\|')[3];
                    // Return rows of data based on the parsed fields.
                    Row row = new Row(4);
                    row.setField(0, ctime);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3, updateTime);
                    System.out.println("Kafka message str ==>" + row.toString());
                    // Return a row of data.
                    collect(row);
            } catch (ClassCastException e) {
                System.out.println("Input data format error. Input data " + msg + "
is not json string");
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
    @Override
    // If the return value is declared as a row, you must reload the UDTF method an
d specify the types of fields to be returned.
    // Define the data types for objects in output rows.
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.TIMESTAMP,DataTypes.STRING, DataTy
pes.Integer, DataTypes.STRING,DataTypes.STRING);
    }
}
```

- UDF

```
package com.alibaba;
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class KafkaUDF extends ScalarFunction {
    // The open method is optional.
    // To implement the open method, you must add "import org.apache.flink.table.fu
nctions.FunctionContext;" to the code.
    public String eval(byte[] message) {
         // Read data of the VARBINARY data type and convert it into the STRING dat
a type.
        String msg = new String(message, "UTF-8");
        return msg.split('\\|')[0];
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    // The close method is optional.
    @Override
    public void close() {
        }
}
```

Create a source table of a self-managed Kafka database

- Example

```
create table kafka_stream(
  messageKey VARBINARY,
  `message` VARBINARY,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka011',
  topic = 'kafka_01',
  `group.id` = 'CID_blink',
  bootstrap.servers = '192.168.0.251:****'
);
```

- Parameters in the WITH clause

  For more information, see Parameters in the WITH clause.

  > Note

  - bootstrap.servers specifies the address and port number of the self-managed Kafka cluster.
  - Metrics, such as transactions per second (TPS) and requests per second (RPS) of Message Queue for Apache Kafka or a self-managed Kafka database, can be displayed only in the console of Realtime Compute for Apache Flink of Blink 2.2.6 and later.

FAQ

- Q: What do I do if the following error occurs when a job is being started?

```
ERR_ID:
     SQL-00010007
CAUSE:
     Could not create table 'kafka_source' as source table
ACTION:
     Please refer to details section for hint.
     If it doesn't help, please contact customer support
DETAIL:
     java.lang.IllegalArgumentException: Startup time[1566481803000] must be before curre
nt time[1566453003356].
```

  A: This error is caused by invalid time zone settings. To solve this issue, add the following parameter to job parameters:

```
blink.job.timeZone=Asia/Shanghai
```

- Q: What do I do if a self-managed Kafka cluster cannot consume data?

  A:

  - Causes

    Each broker of the Kafka cluster sends its metadata to ZooKeeper. Then, the Kafka consumer accesses a broker to extract data by using the endpoint in listener_security_protocol_map in the metadata of the broker. The endpoint is either the IP address or the combination of the local server domain name and port number. If the machine where your Realtime Compute for Apache Flink job resides cannot access the endpoint, the consumer of the connector cannot extract data. As a result, the data consumption process stops.

  - Troubleshooting

    a. View the endpoint in listener_security_protocol_map of the broker of ZooKeeper.

    b. Use the **network detection** feature to check whether you can access the IP address or domain name in the endpoint.

    c. Log on to your machine to confirm the cause.

- Solution
  - If the endpoint is an IP address, check whether an IP address whitelist is configured for the Kafka server. If no whitelist is configured, configure a whitelist and try again.
  - Exclusive clusters of Realtime Compute for Apache Flink cannot resolve domain names. If the endpoint is a domain name and a whitelist has been configured, use one of the following methods:
    - If you cannot restart the Kafka service, perform the following operations:

      Purchase PrivateZone and configure domain name resolution for all Kafka brokers. After network detection based on the domain name succeeds, restart your Realtime Compute for Apache Flink job.
    - If you can restart the Kafka service, perform the following operations:

      Configure the IP address and port number (in boostrap.servers in typical cases) for advertised.listeners for the related broker. Make sure that the network connection is normal. Then, restart your Realtime Compute for Apache Flink job.

- Q: Why does a job terminate immediately after it is started during the consumption of Kafka data?

  A: For versions earlier than Blink 3.3.0, if the startupMode parameter is set to TIMESTAMP and all the partitions of Kafka contain no data, the Kafka connector determines that no partition data can be consumed and terminates the job. You can view log information similar to `Consumer subtask {} initially has no partitions from which to read.` in the `TaskManager.log` log file that corresponds to the job. We recommend that you upgrade the Blink version to 3.3.0 or later.

- Q: What are the features of the commit offset mechanism in Realtime Compute for Apache Flink?

  A: By default, Realtime Compute for Apache Flink uses commitOffsetOnCheckpointing. The commit offset policy configured by users does not take effect. If you enable checkpointing, Realtime Compute for Apache Flink commits the offset that is consumed at the current time to Kafka each time a checkpoint is generated. This way, data is consumed from the offset that is committed from the last checkpoint during job restoration. This ensures exactly-once processing of streaming data. If the checkpoint interval exceeds the specified upper limit, Kafka may fail to query the consumed offset.

2.5.2.4. Create an Oracle database source table

This topic describes how to create an Oracle database source table. It also describes the parameters in the WITH clause, data type mapping, sample code, and FAQ involved when you create an Oracle database source table.

> **Notice**
> - This topic applies only to Blink 3.4.X and later.
> - You can create Oracle database source tables only when you use Oracle 11g.
> - You cannot change the number of concurrent jobs for an Oracle database source table. By default, only one job is allowed for a source table.

Syntax

In Realtime Compute for Apache Flink, you can use an Oracle database to store input data. The following code shows an example:

```
create table oracle_source (
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type = 'oracle',
    url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',
    userName = 'userName',
    password = 'password',
    dbName = 'hr',
    tableName = 'job_history',
    timeField = 'START_DATE',
    startTime = '2007-1-1 00:00:00'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to *oracle*.
url	The connection string of the database.	Yes	Contact Alibaba Cloud O&M engineers.
userName	The username that is used to log on to the database.	Yes	None.
password	The password that is used to log on to the database.	Yes	None.
tableName	The name of the table in the database. Database table names can be in one of the following formats: • *Table name 1,Table name 2* • *Database name.Table name 1,Table name 2* **Note** Multiple table names are separated by commas (,).	Yes	• *table1,table2* • *db1.table1,table2*
timeField	The time when the database was updated.	Yes	None.
dbName	The database name.	No	If you have specified the tableName parameter, you do not need to specify the dbName parameter.

Parameter	Description	Required	Remarks
startTime	The start time of reading data from the source table.	No	*2019-5-15 00:00:00*
timeZone	The time zone of the database.	No	*Asia/Shanghai", "UTC*
queryTimeRangeMs	The time that is taken for data retrieval. Unit: milliseconds. **Note** The value of the queryTimeRangeMs parameter must be greater than the value of the queryIntervalMs parameter.	No	Default value: *5000.*
queryIntervalMs	The interval at which data is queried from the database. Unit: milliseconds.	No	Default value: *100.*
connectionMaxActive	The maximum number of active connections.	No	Default value: *10.*
maxRetry	The maximum number of retries upon a connection failure.	No	Default value: *3.*
escapeFields	Specifies whether to escape field names in the database.	No	Valid values: • *false*: not case-sensitive. This is the default value. • *true*: case-sensitive.

Parameter	Description	Required	Remarks

lengthCheck	The policy for checking the number of fields that are parsed from a row of data.	No	Valid values:
			• *NONE*: This is the default value.
			  ◦ If the number of fields parsed from a row of data is greater than the defined number of fields, data is extracted from left to right based on the order of defined fields.
			  ◦ If the number of fields parsed from a row of data is less than the defined number of fields, the current row is skipped.
			• *SKIP*: If the number of fields parsed from a row of data is different from the defined number of fields, the current row is skipped.
			• *EXCEPTION*: If the number of fields parsed from a row of data is different from the defined number of fields, an error is returned.
			• *PAD*:
			  ◦ If the number of fields parsed from a row of data is greater than the defined number of fields, data is padded from left to right based on the order of defined fields.
			  ◦ If the number of fields parsed from a row of data is less than the defined number of fields, the values of the missing fields are padded with *null* from left to right.

columnErrorDebug	Specifies whether the debugging feature is enabled. **Note** If the debugging feature is enabled, the system displays the logs that record the parse failures.	No	Default value: *false*.

Mapping between field data types

Data type of the Oracle database	Data type of Realtime Compute for Apache Flink
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

Sample code

The following example shows how to create an Oracle database source table in a Realtime Compute for Apache Flink job.

```
create table oracle_source (
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type = 'oracle',
    url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',
    userName = 'userName',
    password = 'password',
    dbName = 'hr',
    tableName = 'job_history',
    timeField = 'START_DATE',
    startTime = '2007-1-1 00:00:00'
);
create table test_out(
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type='print'
);
INSERT INTO test_out
SELECT
    EMPLOYEE_ID,
    START_DATE,
    END_DATE,
    JOB_ID,
    DEPARTMENT_ID
from oracle_source;
```

FAQ

Q: What do I do if no data is found?

A: The data cannot be found because Blink is faulty. To check whether Blink is faulty, view the `Round start:[{}], end:[{}]` and `Round read records` logs on the TaskManager tab. If the logs do not contain data, Blink is faulty.

> **Note**

- `Round start:[{}], end:[{}]`: displays the start time of the queried data.
- `Round read records`: displays the queried data records.

2.5.2.5. Create a Log Service source table

This topic describes how to create a Log Service source table in Realtime Compute for Apache Flink. This topic also describes the attribute fields, parameters in the WITH clause, and data type mappings used when you create a Log Service source table.

> **Notice** This topic applies only to Blink 1.4.5 and later.

What is Log Service?

Log Service is an end-to-end data logging service that is developed by Alibaba Cloud. The data format of Log Service is similar to JSON. The following code shows an example:

```
{
    "a": 1000,
    "b": 1234,
    "c": "li"
}
```

Log Service stores streaming data. Therefore, Realtime Compute for Apache Flink can use Log Service tables as result tables for the processing of streaming data.

DDL syntax

The following sample code describes how to create a Log Service source table in a data definition language (DDL) statement. In the code, `sls` indicates Log Service.

```
create table sls_stream(
  a INT,
  b INT,
  c VARCHAR
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId ='<yourAccessId>',
  accessKey ='<yourAccessKey>',
  startTime = '2017-07-05 00:00:00',
  project ='<yourProjectName>',
  logStore ='<yourLogStoreName>',
  consumerGroup ='<yourConsumerGroupName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to sls.
endPoint	The endpoint of Log Service.	Yes	Contact Alibaba Cloud O&M engineers.
accessId	The AccessKey ID that is used to access Log Service.	Yes	N/A.
accessKey	The AccessKey secret that is used to access Log Service.	Yes	N/A.
project	The name of the Log Service project from which data is read.	Yes	N/A.
logStore	The name of the Logstore in the Log Service project.	Yes	N/A.
startTime	The time at which logs start to be consumed.	No	N/A.
consumerGroup	The name of a consumer group.	No	You can specify this parameter based on your business requirements. The format of the name is not fixed.
heartBeatIntervalMills	The heartbeat interval of the consumer client.	No	Default value: 10000. Unit: milliseconds.
maxRetryTimes	The maximum number of retries for reading data.	No	Default value: 5.

Parameter	Description	Required	Remarks
batchGetSize	The number of log items that are read from a log group at a time.	No	Default value: 100.
columnErrorDebug	Specifies whether to enable debugging.	No	Default value: false. This indicates that debugging is disabled. If you enable debugging, logs that contain parsing exceptions are printed.
startupMode	Specifies whether to enable the consumption mode.	No	Valid values: • TIMESTAMP: Realtime Compute for Apache Flink starts to consume data from a specified time point in each shard. This is the default value. • Earliest: Realtime Compute for Apache Flink starts to consume data from the earliest offset in each shard. • Latest: Realtime Compute for Apache Flink starts to consume data from the latest offset in each shard. • Group_Offsets: Realtime Compute for Apache Flink preferentially consumes data from the checkpoint that is stored on the server in each shard. You must specify the consumerGroup parameter. The consumption mode varies based on the following scenarios: ◦ If recovery from the Flink state is successful, Realtime Compute for Apache Flink starts to consume data from the checkpoint of the Flink state.

Parameter	Description	Required	Remarks
			○ If recovery from the Flink state fails, the consumption mode varies based on the following scenarios: ■ If a checkpoint is available for the specified consumer group, Realtime Compute for Apache Flink attempts to consume data from this checkpoint. ■ If no checkpoint is available for the specified consumer group, the consumption mode varies based on the following scenarios: ■ If you specify startTime, Realtime Compute for Apache Flink starts to consume data at the specified start time. ■ If you do not specify startTime, Realtime Compute for Apache Flink starts to consume data from the earliest offset in each shard.

Note

- In Realtime Compute for Apache Flink V1.6.0 and earlier, the read performance may be affected if the number of shards in a consumer group is specified. This issue is being rectified.
- Log Service does not support the MAP data type.
- Log Service sets the fields that do not exist to null.
- We recommend that you define the fields in the same order as the fields in the source table. Unordered fields are also supported.
- If input data is in the JSON format, define a separator and use the built-in function JSON_VALUE to analyze the data. Otherwise, the following parsing error is returned:

```
2017-12-25 15:24:43,467 WARN [Topology-0 (1/1)] com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector - Field missing error, table column number: 3, data column number: 3, data filed number: 1, data: [{"lg_order_code":"LP00000005","activity_code":"TEST_CODE1","occur_time":"2017-12-10 00:00:01"}]
```

- The value of the batchGetSize parameter cannot exceed 1000. Otherwise, an error is returned.
- The batchGetSize parameter specifies the number of log items that are read at a time in a log group. If the value of the batchGetSize parameter and the size of a single log item that is specified in LogItem are large, frequent garbage collections (GCs) may occur. In this case, you must reduce the value of the batchGetSize parameter.

Notice

- Only Blink 3.6.5 and later support this parameter.
- The preceding configurations take effect only when no checkpoint exists in the state data.

Data type mappings

The following table describes the mapping between the data types of Log Service and Realtime Compute for Apache Flink. We recommend that you declare the mappings in a DDL statement.

Data type of Log Service	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

Attribute fields

By default, Flink SQL supports retrieving three types of Log Service attribute fields. Custom fields are also supported as the input.

Field	Description
`__source__`	The message source.
`__topic__`	The message topic.
`__timestamp__`	The time when a log was generated.

Sample code

```
create table sls_input(
  a int,
  b int,
  c varchar
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId ='<yourAccessI>',
  accessKey ='<yourAccessKey>',
  startTime = '2017-07-05 00:00:00',
  project ='ali-cloud-streamtest',
  logStore ='stream-test',
  consumerGroup ='consumerGroupTest1'
);
create table print_output(
 a int,
 b int,
 c varchar
) with (
  type='print'
);
INSERT INTO print_output
SELECT
  a, b, c
from sls_input;
```

FAQ

- Q: Why does the overall latency of a job increase, or why is no output generated for the job that has window aggregation?

  A: This issue occurs if no new data is written to a partition. To solve this issue, change the parallelism to be the same as the number of read and write partitions.

- Q: How do I set the parallelism?

  A: We recommend that you set the parallelism to the number of partitions. Otherwise, if the speeds at which data is read from two partitions vary significantly, data may be filtered out or data latency may occur when you set the start offset for a job to a time prior to the present time.

- Q: How do I troubleshoot the issue that the latency of a Flink job increases?

A: The Log Service source table may be sharded. Shard indexes may not be continuous after sharding, which increases the latency of a Flink job. If you find that the latency of a Flink job increases, check whether the Log Service source table is sharded.

2.5.2.6. Create a full MaxCompute source table

This topic describes how to create a full MaxCompute source table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause and data type mapping involved when you create a full MaxCompute source table.

> **Notice**
> - This topic applies only to Blink 2.2.7 and later.
> - Full MaxCompute source tables are typically used as bounded stream tables. This makes MaxCompute different from other data sources such as DataHub and Kafka. In Blink 3.4.4, you can specify a full MaxCompute source table as an unbounded stream table. This way, the source table can continuously listen to the generation of new partitions. If a new partition is generated, Realtime Compute for Apache Flink reads data from the new partition. This feature is deprecated in Blink 3.5.0. To use a MaxCompute source table as an unbounded stream table, see Create an incremental MaxCompute source table.

DDL syntax

In Realtime Compute for Apache Flink, you can use MaxCompute to store input data. The following code shows an example:

```
create table odps_source(
    id INT,
    user_name VARCHAR,
    content VARCHAR
) with (
    type = 'odps',
    endPoint = 'http://service.cn.maxcompute.aliyun-inc.com/api',
    project = '<projectName>',
    tableName = '<tableName>',
    accessId = '<yourAccessKeyId>',
    accessKey = '<yourAccessKeySecret>',
    `partition` = 'ds=2018****' --If your MaxCompute source table is a non-partitioned tabl
e, you do not need to declare this parameter.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
endPoint	The endpoint of MaxCompute.	Yes	Contact Alibaba Cloud O&M engineers.

Parameter	Description	Required	Remarks
tunnelEndpoint	The endpoint of the MaxCompute Tunnel service.	No	Contact Alibaba Cloud O&M engineers.
project	The name of a MaxCompute project.	Yes	None.
tableName	The name of the MaxCompute source table.	Yes	None.
accessId	AccessKey ID	Yes	None.
accessKey	AccessKey Secret	Yes	None.
partition	The name of a partition.	No	• A full MaxCompute source table that contains only one partition For example, if only one partition key column `ds` exists, `` `partition` = 'ds=20180905' `` indicates that data in the `ds=20180905` partition is read. • A full MaxCompute source table that contains multiple partitions For example, if two partition key columns `ds` and `hh` exist, `` `partition`='ds=20180905,hh=*' `` indicates that data in the `ds=20180905` partition is read. **Note** You must declare the values of all partitions when you filter partitions. In the preceding example, if you declare only `'partition' = 'ds=20180905'`, no partition is read.

Parameter	Description	Required	Remarks
subscribeNewPartition	Specifies whether to listen to new partitions that meet specific conditions.	No	Default value: false. This value indicates that the system does not listen to new partitions. **Note** • If the subscribeNewPartition parameter is set to `true`, you cannot specify the value of the partition parameter. Otherwise, new partitions cannot be read. • This parameter is provided only in Blink 3.4.4. The parameter is decrepated in Blink 3.5.0. If you need to use this parameter, create an incremental MaxCompute source table. For more information, see Create an incremental MaxCompute source table.
subscribeIntervalInSec	The interval at which new partitions are listened to.	No	Default value: 30. Unit: seconds. **Note** If the value of this parameter is too small, pressure may be caused on the MaxCompute metadata service. This may result in failures to listen to the service.

Field type mapping

MaxCompute data type	Data type of Realtime Compute for Apache Flink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP

MaxCompute data type	Data type of Realtime Compute for Apache Flink
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

Sample code

The following sample code demonstrates how to create a full MaxCompute source table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE odps_source (
    cid varchar,
    rt DOUBLE,
) with (
    type = 'odps',
    endPoint = '<yourEndpointName>',
    project = '<yourProjectName>',
    tableName = '<yourTableName>',
    accessId = '<yourAccessId>',
    accessKey = '<yourAccessPassword>',
    partition = 'ds=20190712'
);
CREATE TABLE test (
    cid varchar,
    invoke_count BIGINT
) with (
  type='print'
);
INSERT INTO test
SELECT
    cid,
    count(*) as invoke_count
FROM odps_source GROUP BY cid;
```

2.5.2.7. Create a Message Queue for Apache RocketMQ source table

This topic describes how to create a Message Queue for Apache RocketMQ source table in Realtime Compute for Apache Flink. This topic also describes the comma-separated values (CSV) file format, parameters in the WITH clause, and data type mappings used when you create a Message Queue for Apache RocketMQ source table.

> **Note** If you need to use Message Queue for Apache RocketMQ that has separate namespaces, use Blink 3.X.

Introduction to Message Queue for Apache RocketMQ

Message Queue for Apache RocketMQ is a professional messaging middleware that is developed by Alibaba Cloud. It is a core service of the enterprise-level Internet architecture. You can specify Message Queue for Apache RocketMQ tables as source tables for Realtime Compute for Apache Flink to process streaming data.

Example

```
create table mq_stream(
  x varchar,
  y varchar,
  z varchar
) with (
  type='mq',
  topic='<yourTopicName>',
  endpoint='<yourEndpoint>',
  pullIntervalMs='1000',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  startMessageOffset='1000',
  consumerGroup='<yourConsumerGroup>',
  fieldDelimiter='|'
);
```

> **Note** Message Queue for Apache RocketMQ stores unstructured data. You do not need to define schemas for Message Queue for Apache RocketMQ source tables. Instead, schemas are specified at the business layer. Realtime Compute for Apache Flink supports messages in the CSV and binary formats.

CSV format

The following example shows a Message Queue for Apache RocketMQ message in the CSV format.

```
1,name,male
2,name,female
```

> **Note** The number of data records that can be contained in a Message Queue for Apache RocketMQ message is not limited. Multiple data records are separated by `\n`.

To declare a Message Queue for Apache RocketMQ source table in a Realtime Compute for Apache Flink job, you can use the following DDL statement:

```
create table mq_stream(
  id varchar,
  name varchar,
  gender varchar
) with (
  type='mq',
  topic='<yourTopicName>',
  endpoint='<ourEndpoint>',
  pullIntervalMs='1000',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  startMessageOffset='1000',
  consumerGroup='<yourConsumerGroup>',
  fieldDelimiter='|'
);
```

Binary format

For messages in the binary format, you can use the following sample code to create a Message Queue for Apache RocketMQ source table:

```
create table source_table (
  mess varbinary
) with (
  type = 'mq',
  endpoint = '<yourEndpoint>',
  pullIntervalMs='500',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic = '<yourTopicName>',
  consumerGroup='<yourConsumerGroup>'
);
create table out_table (
  commodity varchar
) with (
  type='print'
);
INSERT INTO out_table
SELECT
  cast(mess as varchar)
FROM source_table;
```

> **Note**

- The `cast (mess as varbinary)` method is supported only in Realtime Compute for Apache Flink that uses Blink 2.0 or later. If the Blink version is earlier than 2.0, upgrade the Blink version first.
- Data of the VARBINARY type can be passed in only once.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to mq.
topic	The name of a topic.	Yes	N/A.
endPoint	The endpoint of Message Queue for Apache RocketMQ.	Yes	Contact Alibaba Cloud O&M engineers to obtain the required information.
accessId	AccessKey ID	Yes	N/A.
accessKey	AccessKey Secret	Yes	N/A.
consumerGroup	The name of a consumer group.	Yes	N/A.
pullIntervalMs	The interval at which messages are pulled.	No	Unit: milliseconds.
startTime	The time when Realtime Compute for Apache Flink starts to consume messages.	No	N/A.
startMessageOffset	The start offset to consume messages.	No	This parameter is optional. If you configure this parameter, messages are consumed from the point of time that is specified by this parameter.
tag	The subscription tag.	No	N/A.
lineDelimiter	A row delimiter that is used to parse message blocks.	No	Default value: `\n`.
fieldDelimiter	A field delimiter.	No	Default value: `\u0001`. `\u0001` is the field delimiter in read-only mode, and `^A` is the field delimiter in edit mode. `\u0001` is invisible in read-only mode.
encoding	The encoding format.	No	Default value: `utf-8`.

Parameter	Description	Required	Remarks
lengthCheck	The policy that is used to check the number of fields parsed from a row of data.	No	Default value: NONE. • If the number of fields that are parsed from a row of data is greater than the specified number of fields, data is extracted from left to right based on the order of specified fields. • If the number of fields that are parsed from a row of data is less than the specified number of fields, this row of data is skipped. Other valid values are SKIP, EXCEPTION, and PAD. • SKIP: If the number of fields that are parsed from a row of data is different from the specified number of fields, this row of data is skipped. • EXCEPTION: If the number of fields that are parsed from a row of data is different from the specified number of fields, an exception is reported. • PAD: Data is padded from left to right based on the order of specified fields. ◦ If the number of fields that are parsed from a row of data is greater than the specified number of fields, data is extracted from left to right based on the order of specified fields. ◦ If the number of fields that are parsed from a row of data is less than the specified number of fields, the values of the missing fields are padded with null.
columnErrorDebug	Specifies whether to enable debugging.	No	Default value: FALSE. If you configure this parameter to TRUE, a log entry is displayed when a parsing exception occurs.
instanceID	The ID of a Message Queue for Apache RocketMQ instance.	No	If the Message Queue for Apache RocketMQ instance does not have separate namespaces, you do not need to configure this parameter. If the Message Queue for Apache RocketMQ instance has separate namespaces, you must configure this parameter.

Data type mapping

Data type of Message Queue for Apache RocketMQ	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

2.5.2.8. Create a Tablestore source table

This topic describes how to create a Tablestore source table in Realtime Compute for Apache Flink.

> **Notice** This topic applies only to Realtime Compute for Apache Flink V3.2.2 and later.

What is Tablestore?

Tablestore is a distributed NoSQL database service that is built on the Apsara distributed operating system of Alibaba Cloud. Tablestore adopts sharding and load balancing technologies to scale out services and handle concurrent transactions. You can use Tablestore to store and query large amounts of structured data in real time.

Tunnel Service of Tablestore

Tunnel Service is a centralized service that uses the Tablestore API to allow you to consume full and incremental data. You can use the Tunnel Service API and SDKs to create tunnels from which you can consume distributed data in real time. The distributed data is divided into the following types: incremental data, full data, and full and incremental data. You can use the tunnels of the Tunnel Service to consume existing data or added data in tables in streaming processing mode. In Realtime Compute for Apache Flink, the tunnels of the Tunnel Service can serve as the source of streaming data. Each data record uses a JSON-like format. You can run the following code if you need to use Tunnel Service tunnels:

```
{
            "OtsRecordType": "PUT",  // The data operation type, such as PUT, UPDATE, and D
ELETE.
            "OtsRecordTimestamp": 1506416585740836, // The time when the data is written. T
he time unit is milliseconds. The value 0 indicates that full data is written.
            "PrimaryKey": [
                {
                    "ColumnName": "pk_1", // The first primary key column.
                    "Value": 1506416585881590900
                },
                {
                    "ColumnName": "pk_2", // The second primary key column.
                    "Value": "string_pk_value"
                }
            ],
            "Columns": [
                {
                    "OtsColumnType": "Put", // The operation type for the column, such as P
UT, DELETE_ONE_VERSION, and DELETE_ALL_VERSION.
                    "ColumnName": "attr_0",
                    "Value": "hello_table_store",
                },
                {
                    "OtsColumnType": "DELETE_ONE_VERSION", // No value is specified for the
delete operation.
                    "ColumnName": "attr_1"
                }
            ]
}
```

DDL syntax

In Realtime Compute for Apache Flink, you can use Tablestore to store input data. The following code shows an example:

```
create table tablestore_stream(
  pk_1 BIGINT,
  pk_2 VARCHAR,
  attr_0 VARCHAR,
  attr_1 DOUBLE,
  OtsRecordType  VARCHAR  HEADER //You must add HEADER to the attribute field.
) with (
  type ='ots',
  endPoint ='http://blink-demo.cn-hangzhou.vpc.tablestore.aliyuncs.com',
  instanceName = 'blink-demo',
  tableName ='demo_table',
  tunnelName = 'blink-demo-stream',
  accessId ='<yourAccessID>',
  accessKey ='<yourAccessSecret>',
  ignoreDelete = 'false' // Specifies whether to ignore the delete operation.
);
```

Attribute fields

Field	Description
`OtsRecordType`	The data operation type.
`OtsRecordTimestamp`	The data operation time. If you set this parameter to 0, full data is written.
`<Column name>_OtsColumnType`	The operation type for a column.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the connector.	Set the value to `ots`.
endPoint	The endpoint of the Tablestore instance.	Contact Alibaba Cloud O&M engineers to obtain the required information.
instanceName	The name of the Tablestore instance.	N/A.
tableName	The name of the Tablestore table.	Realtime Compute for Apache Flink does not reread data that has been read from a Tablestore source table. If you want Realtime Compute for Apache Flink to reread full data from the source table, you must create a data tunnel.

Parameter	Description	Remarks
tunnelName	The tunnel name of the Tablestore source table.	N/A.
accessId	The AccessKey ID that is used to access Tablestore.	N/A.
accessKey	The AccessKey secret that is used to access Tablestore.	N/A.
ignoreDelete	Specifies whether to ignore the delete operation.	Optional. Default value: `false`.

2.5.2.9. Create a Hologres source table

This topic describes how to create a Hologres source table. It also describes the data definition language (DDL) syntax, parameters in the WITH clause, data type mapping, and sample code used when you create a Hologres source table.

Notice

- We recommend that you use Hologres 0.7 or later.
- You can use only the Hologres tables that use the row store structure as source tables for Realtime Compute for Apache Flink.
- Hologres source tables support projection pushdown. This allows you to read data from only the required columns in the Hologres source tables.
- Concurrent Blink jobs can read data from one or more Hologres shards. We recommend that the number of concurrent Blink jobs be no more than the number of Hologres shards.
- You can use Hologres source tables to process streaming data and batch data.
- Blink jobs execute snapshot statements to read existing data from Hologres source tables at a high rate. After the read operation is complete, the jobs end. If the jobs fail to read data, they read the data again.

Introduction to Hologres

Hologres is compatible with the PostgreSQL protocol and closely connected to the big data ecosystem. Hologres allows you to analyze and process petabytes of data in high concurrency and low latency scenarios. Hologres provides an easy method for you to use the existing business intelligence (BI) tools to perform multidimensional analysis and explore your business.

DDL syntax

```
create table mysource(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='...',
  tablename='...',
  username='...',
  password='...',
  endpoint='...',
  field_delimiter='...' -- This parameter is optional.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to *hologres*.
dbname	The name of the database.	Yes	None.
tablename	The name of the table.	Yes	None.
username	The username that is used to log on to the database. You must enter the AccessKey ID of your Alibaba Cloud account.	Yes	None.
password	The password that is used to log on to the database. You must enter the AccessKey secret of your Alibaba Cloud account.	Yes	None.
endpoint	The endpoint of Hologres.	Yes	Contact Alibaba Cloud O&M engineers.
field_delimiter	The delimiter used between rows when data is being exported. **Notice** Delimiters cannot be inserted into the data.	Yes	Default value: *"\u0002"*.

Field type mapping

Hologres	BLINK
INT	INT

Hologres	BLINK
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

Sample code

The following sample code demonstrates how to create a Hologres source table in a Realtime Compute for Apache Flink job.

```
create table mysource(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='...',
  tablename='...',
  username='...',
  password='...',
  endpoint='...',
  field_delimiter='...' -- This parameter is optional.
);
create table print_output(
 a varchar,
 b BIGINT,
 c BIGINT
) with (
  type='print'
);
INSERT INTO print_output
SELECT
  a, b, c
from mysource;
```

2.5.2.10. Create an incremental MaxCompute source table

This topic describes how to create an incremental MaxCompute source table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause, data type mapping, and FAQ involved when you create an incremental MaxCompute source table.

Notice

- This topic applies only to Blink 3.5.0-hotfix.
- Incremental MaxCompute source tables cannot be used as dimension tables.
- Incremental MaxCompute source tables support only partitioned tables.

Syntax

```
create table odps_source(
    id int,
    user_name VARCHAR,
    content VARCHAR
) with (
    type = 'continuous-odps',
    endPoint = 'your_end_point_name',
    project = 'your_project_name',
    tableName = 'your_table_name',
    accessId = 'your_access_id',
    accessKey = 'your_access_key',
    startPartition = 'ds=20180905'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the connector.	Yes	Set the value to `continuous-odps`.
endPoint	The endpoint of MaxCompute.	Yes	Contact Alibaba Cloud O&M engineers.
tunnelEndpoint	The endpoint of the MaxCompute Tunnel service.	• Yes: This parameter is required if MaxCompute is deployed in a VPC. • No: This parameter is not required if MaxCompute is not deployed in a VPC.	Contact Alibaba Cloud O&M engineers.
project	The name of the project to which the table belongs.	Yes	None.
tableName	The name of the table.	Yes	None.
accessId	AccessKey ID	Yes	None.
accessKey	AccessKey Secret	Yes	None.

Parameter	Description	Required	Remarks
startPartition	The partition in which data starts to be read. When the system loads partitioned tables, it compares startPartition and all partitions in each partitioned table in alphabetical order and then loads the data that meets a specified condition from the partitions. An incremental MaxCompute source table can also continuously listen to incremental MaxCompute partitioned tables. After the source table reads data from all the existing partitions, it continues to listen to the generation of new partitions. If a new partition is generated, the source table reads data from the new partition. Note • An incremental MaxCompute source table must have level-1 partitions. Level-2 partitions are optional. • If you specify a level-2 partition, make sure that it is placed after a level-1 partition. • If the partition specified by the startPartition parameter does not exist, the system reads data from the next partition.	Yes	For example, if startPartition is set to ds=20191201, data of all partitions that meets the condition of ds >= 20191201 in the incremental MaxCompute partitioned table is loaded. Assume that an incremental MaxCompute partitioned table contains five partitions. Each partition has two partition key columns with a level-1 partition of ds and level-2 partition of type. • ds=20191201,type=a • ds=20191201,type=b • ds=20191202,type=a • ds=20191202,type=b • ds=20191202,type=c If the values of startPartition for the partitions are different, the following partitions meet the preceding condition: • ds=20191202 ◦ ds=20191202,type=a ◦ ds=20191202,type=b ◦ ds=20191202,type=c • ds=20191201,type=c ◦ ds=20191202,type=a ◦ ds=20191202,type=b ◦ ds=20191202,type=c

Field type mapping

MaxCompute data type	Data type of Realtime Compute for Apache Flink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT

MaxCompute data type	Data type of Realtime Compute for Apache Flink
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

Notice

- Incremental MaxCompute source tables do not support the CHAR, VARCHAR, ARRAY, MAP, or STRUCT data types.
- You can use the STRING data type instead of the VARCHAR data type.

Sample code

One partition is generated in an incremental MaxCompute source table every day. The partition key column is ds. The incremental MaxCompute source table loads data from partitions whose numbers are greater than or equal to 20191201 and continuously listens to the generation of new partitions.

```
-- The incremental MaxCompute source table reads data from partitions in the range of [ds=2
0191201, ∞).
CREATE TABLE odps_source (
    cid VARCHAR,
    rt DOUBLE,
) with (
    type = 'continuous-odps',
    endPoint = 'your_end_point_name',
    project = 'your_project_name',
    tableName = 'your_table_name',
    accessId = 'xxxx',
    accessKey = 'xxxx',
    startPartition = 'ds=20191201'
);
CREATE TABLE test (
    cid VARCHAR,
    rt DOUBLE,
) with (
    type='print'
);
INSERT INTO test
SELECT
    cid, rt FROM odps_source;
```

2.5.3. Create a data result table

2.5.3.1. Overview

Realtime Compute uses the `CREATE TABLE` statement to define the schema of a result table for output data and how data is written into a target data store.

Data can be written by using either of the following methods: append and update.

- Append: If the result table is stored in a relational database with no primary key defined, Log Service, or Message Queue, output data is appended to the result table.
- Update: If the result table is stored in a relational database or a Hadoop database with a primary key defined, output data is written into the result table as follows:
  - If the value for the primary key is not found in the result table, the data record is inserted into the database.
  - If the value for the primary key is found, the existing data record in the database is overwritten.

Result table syntax

```
CREATE TABLE tableName
    (columnName dataType [, columnName dataType ]*)
    [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

Example

```
CREATE TABLE rds_output(
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='rds',
url='yourDatabaseURL',
tableName='yourTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);
```

2.5.3.2. Create a DataHub result table

This topic describes how to create a DataHub result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause and data type mapping involved when you create a DataHub result table.

> **Notice** This topic applies only to Blink 1.4.5 and later.

Introduction to DataHub

DataHub is a real-time data distribution platform designed to process streaming data. You can publish and subscribe to applications for streaming data in DataHub and distribute the data to other platforms. DataHub allows you to analyze streaming data and build applications based on the streaming data. Realtime Compute for Apache Flink uses DataHub to store source and result tables for streaming data processing.

DDL syntax

In Realtime Compute for Apache Flink, you can use DataHub to store output data. In the following sample code, the data definition language (DDL) statement creates a DataHub result table:

```
create table datahub_output(
  name VARCHAR,
  age BIGINT,
  birthday BIGINT
)with(
  type='datahub',
  endPoint='<yourEndpoint>,
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessKey>',
  batchSize='<yourBatchSize>',
  batchWriteTimeoutMs='1000'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to `datahub`.
endPoint	The endpoint of DataHub.	Yes	Contact Alibaba Cloud O&M engineers.
project	The name of a DataHub project.	Yes	None.
topic	The name of a DataHub topic.	Yes	None.
accessId	AccessKey ID	Yes	None.
accessKey	AccessKey Secret	Yes	None.
maxRetryTimes	The maximum number of retries for reading data from DataHub.	No	The default value of this parameter varies based on the Blink versions. • For versions earlier than Blink 2.2.7, the default value is 3. • For Blink 2.2.7 and later, the default value is 20.

Parameter	Description	Required	Remarks
batchSize	The number of data records that can be written at a time.	No	The default value of this parameter varies based on the Blink versions. • For versions earlier than Blink 3.3, the default value is 300. • For Blink 3.3 and later, the default value is 100.
batchWriteTimeoutMs	The write timeout period.	No	Optional. Default value: 5000. Unit: milliseconds. This value indicates that if the number of input data records does not reach the data output condition within 5,000 milliseconds, all cached data is written into the result table.
maxBlockMessages	The maximum number of data blocks that are written at a time.	No	Default value: 100.
reserveMilliSecond	Specifies whether to reserve the millisecond component in a value of the TIMESTAMP data type.	No	Default value: false. **Notice** This feature is available only in Realtime Compute for Apache Flink V2.2.6 and later.
partitionBy	The partitioning rule for the result table. Before Realtime Compute for Apache Flink writes data into the result table, Realtime Compute for Apache Flink performs hash partitioning based on the value of this parameter. Then, the data flows to the related partition.	No	This parameter is empty by default. This indicates that data is randomly allocated. **Note** The partitionBy parameter determines the Blink subtask to which data flows.

Parameter	Description	Required	Remarks
hashFields	The names of columns. After column names are specified, the values of the same column are written into the same shard.	No	Default value: Null. This indicates that data is written randomly. You can specify multiple column values and separate them with commas (,), for example, `hashFields='a,b'` . **Notice** • This feature is available only in Blink 3.3.0 and later. • The hashFields parameter determines the shard into which a data stream is written.

Field type mapping

The following table lists the mappings between the data types of DataHub and Realtime Compute for Apache Flink. We recommend that you declare the mapping in DDL statements.

Data type of DataHub	Data type of Realtime Compute for Apache Flink
BIGINT	BIGINT
TIMESTAMP	BIGINT **Note** In DataHub, TIMESTAMP is a 16-bit UNIX timestamp that is accurate to microseconds. However, in Realtime Compute for Apache Flink, TIMESTAMP is 13-bit UNIX timestamp that is accurate to milliseconds. We recommend that you use BIGINT for mapping. If you need to use TIMESTAMP, we recommend that you use computed columns for conversion. For more information, see Computed column.
STRING	VARCHAR
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN

Data type of DataHub	Data type of Realtime Compute for Apache Flink
DECIMAL	DECIMAL

Sample code

The following sample code demonstrates how to create a DataHub result table in a Realtime Compute for Apache Flink job:

```
create table datahub_input(
  name VARCHAR
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='test1',
  topic='topic1',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2018-06-01 00:00:00'
);
create table datahub_output(
  name varchar
)with(
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='test2',
  topic='topic2',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  batchSize='1000',
  batchWriteTimeoutMs='500'
);
INSERT INTO datahub_output
SELECT
  LOWER(name)
from datahub_input;
```

2.5.3.3. Create a Message Queue for Apache RocketMQ result table

This topic describes how to create a Message Queue for Apache RocketMQ result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause used when you create a Message Queue for Apache RocketMQ result table.

Notice

- This topic applies only to Blink 1.4.5 and later.
- If you need to use Message Queue for Apache RocketMQ that has separate namespaces, use Blink 3.X.

Introduction to Message Queue for Apache RocketMQ

Message Queue for Apache RocketMQ is a professional message middleware that is developed by Alibaba Cloud for commercial use. It is a core product for the enterprise-level Internet architecture. Based on the high-availability distributed cluster technology, Message Queue for Apache RocketMQ provides comprehensive cloud messaging services, including message publishing and subscription, message tracing, resource statistics, message scheduling or delaying, monitoring, and alerts.

CSV format

You can specify Message Queue for Apache RocketMQ tables as result tables for Realtime Compute for Apache Flink to process streaming data. In the following sample code, the DDL statement creates a Message Queue for Apache RocketMQ result table to store streaming data in the CSV format:

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR
) WITH (
type='mq',
endpoint='<yourEndpoint>',
accessID='<yourAccessId>',
accessKey='<yourAccessSecret>',
topic='<yourTopicName>',
producerGroup='<yourGroupName>',
tag='<yourTagName>',
encoding='utf-8',
fieldDelimiter=',',
retryTimes='5',
sleepTimeMs='500'
);
```

Binary format

You can specify Message Queue for Apache RocketMQ tables as result tables for Realtime Compute for Apache Flink to process streaming data. In the following sample code, the DDL statement creates a Message Queue for Apache RocketMQ result table to store streaming data in the binary format:

```
CREATE TABLE source_table (
  commodity VARCHAR
)WITH(
  type='random'
);
CREATE TABLE result_table (
  mess VARBINARY
) WITH (
  type = 'mq',
  endpoint='<yourEndpoint>',
  accessID='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic='<yourTopicName>',
  producerGroup='<yourGroupName>'
);
INSERT INTO result_table
SELECT
CAST(SUBSTRING(commodity,0,5) AS VARBINARY) AS mess
FROM source_table;
```

Note The `cast(varchar as varbinary)` method is supported only in Blink 2.0 or later. If the Blink version is earlier than 2.0, update the Blink version first.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to mq.
topic	The name of the Message Queue for Apache RocketMQ topic to which data is written.	N/A.
endpoint	The endpoint of Message Queue for Apache RocketMQ.	Contact Alibaba Cloud O&M engineers to obtain the required information.
accessID	AccessKey ID	N/A.
accessKey	AccessKey Secret	N/A.
producerGroup	Specifies the name of the producer group to which messages are written.	N/A.

Parameter	Description	Remarks
tag	The message tag.	Optional. This parameter is empty by default.
fieldDelimiter	The field delimiter.	Optional. Default value: `\u0001`. The delimiter varies based on the following modes: • In read-only mode, the `\u0001` delimiter is used. `\u0001` is invisible in read-only mode. • In edit mode, the `^A` delimiter is used.
encoding	The encoding format.	Optional. Default value: `utf-8`.
retryTimes	The number of retries for writing data to the table.	Optional. Default value: 10.
sleepTimeMs	The retry interval.	Optional. Default value: 1000. Unit: milliseconds.
instanceID	The ID of a Message Queue for Apache RocketMQ instance.	• If the Message Queue for Apache RocketMQ instance does not have a separate namespace, the instanceID parameter cannot be used. • If the Message Queue for Apache RocketMQ instance has a separate namespace, the instanceID parameter is required.

2.5.3.4. Create an ApsaraDB RDS result table

This topic describes how to create an ApsaraDB for RDS or Distributed Relational Database Service (DRDS) result table in Realtime Compute. This topic also describes the mappings between data types of Realtime Compute and ApsaraDB for RDS or DRDS data types.

What is ApsaraDB for RDS?

ApsaraDB RDS is a stable, reliable, and scalable online database service. Based on Apsara Distributed File System and high-performance storage, ApsaraDB for RDS supports a variety of engines such as MySQL, SQL Server, PostgreSQL, and Postgres Plus Advanced Server (PPAS). PPAS is compatible with Oracle databases. ApsaraDB for RDS provides comprehensive solutions for database operations and management, such as data backup, data recovery, monitoring, and data migration.

> **Note** ApsaraDB for RDS and DRDS use the same parameters in the WITH clause. When you use ApsaraDB for RDS or DRDS to store result tables, make sure that there are real tables.

DDL syntax

The following sample code shows how to create an ApsaraDB for RDS or DRDS result table. Sample statements:

```
create table rds_output(
 id int,
 len int,
 content varchar,
 primary key(id,len)
) with (
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

- In Realtime Compute, each row of output data is converted to a row of SQL statement and then written and executed in the destination database. If you want to write multiple rows of data to the result table at the same time, you must add ?rewriteBatchedStatements=true to the URL to improve system performance.
- You can define an auto-increment primary key for an ApsaraDB RDS for MySQL database. If you want to use the auto-increment primary key, do not declare the auto-increment field in the DDL statement. For example, if you use ID as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written into the ApsaraDB RDS for MySQL database, a value is automatically filled for the auto-increment field.
- If a DRDS result table is partitioned, the partition key must be declared in primary key() of the DDL statement. Otherwise, you cannot write data into the partitioned table.

Parameters in the WITH clause

Parameter	Description	Remarks
url	The database URL.	N/A.
tableName	The name of the table in the database.	N/A.
userName	The username that is used to access the database.	N/A.
password	The password that is used to access the database.	N/A.
maxRetryTimes	The maximum number of retries for writing data to the table.	Optional. Default value: 3.
batchSize	The number of data records that are written at a time.	Optional. Default value: 50.
bufferSize	The maximum number of data records that can be stored in the buffer before deduplication is triggered. This parameter setting takes effect only after the primary key is specified.	Optional. Default value: 1. This value indicates that deduplication is triggered each time a data record is stored. In versions of Realtime Compute later than 1.4.1, the default value is 1000.

Parameter	Description	Remarks
flushIntervalMs	The timeout period for writing data.	Optional. Default value: 5000. Unit: milliseconds. This value indicates that if no data is written within 5,000 milliseconds, all cached data is written into the result table.
excludeUpdateColumns	The columns that are excluded when you update the result table based on key values.	Optional. By default, this parameter is not specified, which indicates that primary key fields are excluded.
ignoreDelete	Specifies whether to ignore the delete operation.	Default value: false.
partitionBy	The columns based on which Realtime Compute performs hash partitioning. Before writing data to the sink operator, Realtime Compute performs hash partitioning based on the value of this parameter. The data then flows to the corresponding hash operator.	Optional. This parameter is not specified by default.

Data type mappings

Data type of ApsaraDB RDS	Data type of Realtime Compute for Apache Flink
TEXT	VARCHAR
BYTE	VARCHAR
INTGER	INT
LONG	BIGINT
DOUBLE	DOUBLE
DATE	VARCHAR
DATETIME	VARCHAR
TIMESTAMP	VARCHAR
TIME	VARCHAR
YEAR	VARCHAR
FLOAT	FLOAT
DECIMAL	DECIMAL

Data type of ApsaraDB RDS	Data type of Realtime Compute for Apache Flink
CHAR	VARCHAR

JDBC parameters

Parameter	Description	Default value	Required ApsaraDB RDS for SQL Server version
useUnicode	Specifies whether to use the Unicode character set. If you want to set the characterEncoding parameter to gb2312 or gbk, you must set this parameter to true.	false	1.1g
characterEncoding	The character encoding format. This parameter must be set if the useUnicode parameter is set to true. You can set this parameter to gb2312 or gbk.	false	1.1g
autoReconnect	Specifies whether to automatically re-establish a connection when the connection to the database is unexpectedly interrupted.	false	1.1
autoReconnectForPools	Specifies whether to use the reconnection policy for a database connection pool.	false	3.1.3
failOverReadOnly	Specifies whether the database is read-only after the database is automatically reconnected.	true	3.0.12
maxReconnects	Specifies the maximum number of reconnection attempts allowed if the autoReconnect parameter is set to true.	3	1.1

Parameter	Description	Default value	Required ApsaraDB RDS for SQL Server version
initialTimeout	Specifies the interval between two reconnection attempts if the autoReconnect parameter is set to true. Unit: seconds.	2	1.1
connectTimeout	Specifies the timeout period when you use a socket connection to access the database server. Unit: milliseconds. Default value: 0. This value indicates that the connection never times out. This parameter applies to JDK 1.4 and later.	0	3.0.1
socketTimeout	The timeout period for read and write operations on a socket connection. Unit: milliseconds. Default value: 0. This value indicates that the read or write operation never times out.	0	3.0.1

FAQ

- Q: When output data is written to an ApsaraDB for RDS result table, is a new data record inserted in the table or is the result table updated based on the primary key value?

  A: If a primary key is defined in the DDL statement, you can execute the `INSERT INTO tablename(field1,field2, field3, ...) VALUES(value1, value2, value3, ...) ON DUPLICATE KEY UPDATE field1=value1,field2=value2, field3=value3, ...;` statement to update data. If the primary key does not exist, the value is directly inserted into the table. If the primary key already exists, the existing value is replaced. If no primary key is defined in the DDL statement, you can execute the `INSERT INTO` statement to insert data.

- Q: What do I need to pay attention to when I perform GROUP BY operations based on the unique index of an ApsaraDB RDS for SQL Server table?

  A: If an ApsaraDB for RDS result table has an auto-increment field, this field cannot be declared as the primary key in the DDL statement of Realtime Compute. If you want to perform GROUP BY operations based on the unique index of an ApsaraDB for RDS table, declare the unique index in primary key() in the DDL statement.

2.5.3.5. Create a TSDB result table

This topic describes how to create a Time Series Database (TSDB) result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause that is used when you create a TSDB result table.

> **Notice** This topic applies only to Blink 2.0 and later.

Introduction to TSDB

Alibaba Cloud TSDB is a database service that supports efficient reads and writes, compressed storage, and real-time computing for time series data. TSDB is widely used in Internet of Things (IoT) and Internet fields to implement real-time monitoring, forecasting, and alerting on devices and services.

DDL syntax

In Realtime Compute for Apache Flink, you can use TSDB to store output data. The following code shows an example:

```
CREATE TABLE stream_test_hitsdb (
    metric      VARCHAR,
    `timestamp` INTEGER,
    `value`     DOUBLE,
    tagk1       VARCHAR,
    tagk2       VARCHAR,
    tagk3       VARCHAR
) WITH (
    type='hitsdb',
    host='<yourHostName>',
    virtualDomainSwitch = 'false',
    httpConnectionPool = '20',
    batchPutSize = '1000'
);
```

Default format for tables:

- Column 0: metric (VARCHAR).
- Column 1: timestamp (INTEGER). Unit: seconds.
- Column 2: value (DOUBLE).
- Columns 3 to N: tag keys. The field names in the time series database are used as the tag keys.

> **Note**
> - You can specify multiple tag columns.
> - You must declare the following fields: metric, timestamp, and value. The names, sequence, and data types of the fields must be the same as those in TSDB.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to `hitsdb`.

Parameter	Description	Remarks
host	The IP address or the domain name that is mapped to the virtual IP address (VIP) of the TSDB instance.	see Connect to the instance. Enter the hostname of the TSDB instance.
port	The port number that is used to access the TSDB instance.	Default value: 8242.
virtualDomainSwitch	Specifies whether to use VIPServer.	Default value: false. If you need to use VIPServer, set this parameter to true.
httpConnectionPool	The maximum number of connections in an HTTP connection pool.	Default value: 10.
httpCompress	Specifies whether to use GZIP compression.	Default value: false. This value indicates that GZIP compression is not used.
httpConnectTimeout	The timeout period for an HTTP connection.	Default value: 0.
ioThreadCount	The number of I/O threads.	Default value: 1.
batchPutBufferSize	The buffer size.	Default value: 10000.
batchPutRetryCount	The maximum number of retries for writing data to the result table.	Default value: 3.
batchPutSize	The maximum number of data records that can be submitted at a time.	Default value: 500.
batchPutTimeLimit	The maximum duration for which a data record can be stored in the buffer.	Default value: 200. Unit: milliseconds.

Parameter	Description	Remarks
batchPutConsumerThreadCount	The number of serialized threads.	Default value: 1.

Models for writing data from Realtime Compute for Apache Flink to TSDB

In Blink 3.2 and later, Realtime Compute for Apache Flink can write data to TSDB by using one of the following models:

- Write single-value data points where no tags are included. To use this model, use the specified schema that consists of three fields. The names of these fields cannot be changed. You must use the following schema format for this model:

  ```
  metric,timestamp,value
  ```

- Write single-value data points where tags are included. To use this model, use the specified schema. In the schema, the names of the following fields cannot be changed: metric, timestamp, and value. You can specify the tag names based on your business requirements. You must use the following schema format for this model:

  ```
  metric,timestamp,value,tagKey1,....,tagKeyN
  ```

- Write single-value data points where the number of tags is unknown. To use this model, use the specified schema that consists of four fields. The names of the four fields cannot be changed. You must use the following schema format for this model:

  ```
  metric,timestamp,value,tags
  ```

  In the schema, specify the value of the tags parameter as a JSON string. The JSON string allows you to specify an unknown number of tags. If you do not use the JSON string, you must specify a fixed number of tags for the Blink table schema. In the schema, specify the JSON string in the following format:

  ```
  {"tagKey1":"tagValue1","tagKey2":"tagValue2",……,"tagKeyN":"tagValueN"}
  ```

- Write multi-value data points where tags are not included. You must use the following schema format for this model:

  ```
  metric,timestamp,field_name1,field_name2,……,field_nameN
  ```

  The names of the metric and timestamp fields cannot be changed. For multi-value fields, the field_ prefix is used for each field to distinguish fields from tags and to support single-value data writes. For example, when the `field_name1` field is written to TSDB, the prefix `field_` is automatically removed. In the preceding schema, name1 and name2 are the names of the multi-value fields. For the preceding schema, the following format is used to write data to TSDB:

  ```
  metric,timestamp,name1,name2,……,nameN
  ```

- Write multi-value data points where tags are included. To use this model, use the specified schema. In the schema, the names of the following fields cannot be changed: metric and timestamp. You can specify the tag names based on your business requirements. You must use the following schema

format for this model:

```
metric,timestamp,tagKey1,....,tagKeyN,field_name1,field_name2,……,field_nameN
```

- Write single-value data points where the number of tags is unknown. You must use the following schema format for this model:

  ```
  metric,timestamp,tags,field_name1,field_name2,……,field_nameN
  ```

  The content of tags is a JSON string shown in the following example. Therefore, the limit that the Blink table schema requires a fixed number of tags does not apply.

  ```
  {"tagKey1":"tagValue1","tagKey2":"tagValue2",……,"tagKeyN":"tagValueN"}
  ```

FAQ

Q: Why does the following error occur during a failover "The values of the LONG data type cannot be converted into the values of the INT data type"?

A: Blink versions earlier than 2.2.5 support only the INT data type. Blink 2.2.5 and later support the BIGINT data type.

2.5.3.6. Create an KVStore for Redis result table

This topic describes how to create an KVStore for Redis result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, the mappings between the field data types of KVStore for Redis and Realtime Compute for Apache Flink, and the attribute fields used when you create an KVStore for Redis result table.

> **Notice**
> - This topic applies only to Blink V3.2.0 and later.
> - Realtime Compute for Apache Flink allows you to use self-managed Redis databases to store output data in result tables.

Introduction to KVStore for Redis

KVStore for Redis is a database service that is compatible with the protocols of the open source Redis system. It supports a hybrid of memory and hard disks for storage. KVStore for Redis provides a hot standby architecture to ensure high availability. Based on the scalable cluster architecture, KVStore for Redis can meet the business requirements for high throughputs, low-latency operations, and flexible configuration changes. Realtime Compute for Apache Flink allows you to store the output streaming data in KVStore for Redis.

Syntax

You can use five data types when you write data to KVStore for Redis result tables. To create an KVStore for Redis result table, execute the following data definition language (DDL) statements:

- STRING type

  A DDL statement has two columns. The first column lists keys and the second column lists values. To insert data into an KVStore for Redis result table, run the `set key value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'string',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data when
the retraction message is returned. The default value is false.
);
```

- LIST type

  A DDL statement has two columns. The first column lists keys and the second column lists values. To insert data into an KVStore for Redis result table, run the `lpush key value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'list',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data when
the retraction message is returned. The default value is false.
);
```

- SET type

  A DDL statement has two columns. The first column lists keys and the second column lists values. To insert data into an KVStore for Redis result table, run the `sadd key value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'set',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data when
the retraction message is returned. The default value is false.
);
```

- HASHMAP type

A DDL statement has three columns. The first column lists keys, the second column lists hash keys, and the third column lists hash values. To insert data into an KVStore for Redis result table, run the `hmset key hash_key hash_value` command.

```
create table resik_output (
  a varchar,
  b varchar,
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'hashmap',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data when
the retraction message is returned. The default value is false.
);
```

- SORTEDSET type

  A DDL statement has three columns. The first column lists keys, the second column lists scores, and the third column lists values. To insert data into an KVStore for Redis result table, run the `add key score value` command.

```
create table resik_output (
  a varchar,
  b double,  -- The data must be of the DOUBLE type.
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'sortedset',
  host = '${redisHost}', -- An example value is '127.0.0.1'.
  port = '${redisPort}', -- An example value is '6379'.
  dbNum = '${dbNum}', -- The default value is 0.
  ignoreDelete = 'true' -- Specifies whether to delete the previously inserted data when
the retraction message is returned. The default value is false.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Valid value
type	The type of the result table.		Set the value to `redis`.

Parameter	Description	Required	Valid value
mode	The data type of the KVStore for Redis result table.	Yes	Valid values: • string • list • set • hashmap • sortedset
host	The endpoint of the KVStore for Redis database.		Example: `127.0.0.1`.
port	The port of the KVStore for Redis database.	No	Default value: 6379.
dbNum	The sequence number of the KVStore for Redis database.		Default value: 0.
ignoreDelete	Specifies whether to ignore the retraction message.		Valid values: true and false. Default value: false. If this parameter is set to false, the inserted data and the keys of the data are deleted when a retraction message is received.
password	The password that is used to access the KVStore for Redis database.		This parameter is empty by default. This indicates that permission verification is not required.
clusterMode	Specifies whether the KVStore for Redis database is in cluster mode.		Valid values: • true: cluster mode. • false: standalone mode. This is the default value. **Note** Only Blink 3.6.X and later support this parameter.

Field type mapping

The following table lists the data type mappings between KVStore for Redis and Realtime Compute for Apache Flink. We recommend that you declare the mappings in DDL statements.

Data type of KVStore for Redis	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

Data type of KVStore for Redis	Data type of Realtime Compute for Apache Flink
SCORE	DOUBLE

Note The data of the SCORE type is added to the values of the SORTEDSET data type in KVStore for Redis databases. You must manually set a score of the DOUBLE type for each sorted set value and sort the values based on their scores in ascending order.

Sample code

The following sample code shows how to create an KVStore for Redis result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE random_stream (
  v VARCHAR,
  p VARCHAR) with (
    type = 'random'
);
create table resik_output (
  a VARCHAR,
  b VARCHAR,
  primary key(a)
) with (
  type = 'redis',
  mode = 'string',
  host = '<yourRedisHost>',
  password = '<yourRedisPassword>'
);
INSERT INTO resik_output
SELECT v, p
FROM random_stream;
```

2.5.3.7. Create an AnalyticDB for MySQL V2.0 result table

This topic describes how to create an AnalyticDB for MySQL V2.0 result table. This topic also describes the mappings between the field data types of AnalyticDB for MySQL and Realtime Compute for Apache Flink.

Notice

- This topic applies only to Blink 1.4.5 and later.
- You are allowed to define an auto-increment primary key for an AnalyticDB for MySQL V2.0 database. If you want to use the auto-increment primary key, do not declare the auto-increment field in a data definition language (DDL) statement. For example, if you use ID as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written to the AnalyticDB for MySQL V3.0 database, the value for the auto-increment field is automatically filled.

What is AnalyticDB for MySQL?

AnalyticDB for MySQL is a real-time online analytical processing (OLAP) service that is developed by Alibaba Cloud. AnalyticDB for MySQL is a high concurrency service that has excellent performance in processing large amounts of data. AnalyticDB for MySQL allows you to query and analyze hundreds of billions of data records within milliseconds.

DDL syntax

> **Note** For more information about how to create an AnalyticDB for MySQL V3.0 result table, see Create an AnalyticDB for MySQL V3.0 result table.

In Realtime Compute for Apache Flink, you can use AnalyticDB for MySQL V2.0 to store output data. The following code shows an example:

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='ads',
url='yourDatabaseURL',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>',
batchSize='20'
);
```

> **Note** The primary key that is declared in an AnalyticDB for MySQL result table must be consistent with that in an AnalyticDB for MySQL database. The primary key is case-sensitive. Inconsistency may cause the "array index out of bounds" error.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to ads.
url	The Java Database Connectivity (JDBC) URL of a database.	Contact Alibaba Cloud O&M engineers to obtain the required information.
tableName	The name of the table in the database.	N/A.

Parameter	Description	Remarks
username	The username that is used to access the database.	N/A.
password	The password that is used to access the database.	N/A.
maxRetryTimes	The maximum number of retries for writing data to the table.	Optional. Default value: 10.
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	Optional. Default value: 5000. This value indicates that data deduplication is triggered if the number of input data records in the buffer reaches 5,000.
batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 1000. **Note** If error code 20015 is returned, it indicates that the value specified for the batchSize parameter is excessively large. For AnalyticDB for MySQL databases, the size of data records that are written at a time cannot exceed 1 MB. If the batchSize parameter is set to `1000`, the average size of each data record cannot exceed 1 KB.
batchWriteTimeoutMs	The timeout period for writing data.	Optional. Default value: 5000. Unit: milliseconds. This value indicates that if the number of input data records does not reach the value specified by the batchSize parameter within 5,000 milliseconds, all cached data is written to the result table.
connectionMaxActive	The maximum number of connections in a connection pool.	Optional. Default value: 30.
ignoreDelete	Specifies whether to ignore the delete operation.	Default value: false.

Data type mappings

We recommend that you declare the mapping between the field data types of AnalyticDB for MySQL and Realtime Compute for Apache Flink in DDL statements.

Data type of AnalyticDB for MySQL	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
TINYINT	INT
SMALLINT	
INT	
BIGINT	BIGINT
DOUBEL	DOUBLE
VARCHAR	VARCHAR
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

2.5.3.8. Create an AnalyticDB for MySQL V3.0 result table

This topic describes how to create an AnalyticDB for MySQL V3.0 result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause used when you create an AnalyticDB for MySQL V3.0 result table.

Notice

- You cannot register AnalyticDB for MySQL V3.0 storage resources in the Realtime Compute for Apache Flink console to store result tables.
- This topic applies only to `Blink 3.3.0` and later.
- For more information about how to create an AnalyticDB for MySQL V2.0 result table, see Create an AnalyticDB for MySQL V2.0 result table.
- You are allowed to define an auto-increment primary key for an AnalyticDB for MySQL V3.0 database. If you want to use the auto-increment primary key, do not declare the auto-increment field in a data definition language (DDL) statement. For example, if you use ID as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written to the AnalyticDB for MySQL V3.0 database, the value for the auto-increment field is automatically filled.

DDL syntax

In Realtime Compute for Apache Flink, you can use AnalyticDB for MySQL V3.0 to store output data. The following sample code shows how to create an AnalyticDB for MySQL V3.0 result table.

```
CREATE TABLE rds_output (
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id,len)
) WITH (
type='ADB30',
url='jdbc:mysql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>'
);
```

Principles

Realtime Compute for Apache Flink writes data to an AnalyticDB for MySQL V3.0 result table in two steps:

1. Converts each row of output data to a line of SQL statement.
2. Writes and executes the SQL statement in the destination database.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the connector.	Yes	Set the value to `ADB30`.
url	The Java Database Connectivity (JDBC) URL of the database.	Yes	The URL of the AnalyticDB for MySQL database. **Note** databaseName is the name of the AnalyticDB for MySQL database.
tableName	The name of the table.	Yes	N/A
username	The username that is used to access the database.	Yes	N/A
password	The password that is used to access the AnalyticDB for MySQL database.	Yes	N/A

Parameter	Description	Required	Remarks
maxRetryTimes	The maximum number of retries for writing data.	No	Default value: 3.
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	No	Default value: 1000. This value indicates that duplicates are removed when the number of input data records reaches 1,000. **Note** This parameter is valid only after you specify the primary key.
batchSize	The number of data records that is written at a time.	No	Default value: 1000. **Note** This parameter is valid only after you specify the primary key.
flushIntervalMs	The time interval at which the cache is cleared.	No	Default value: 3000. Unit: milliseconds. This value indicates that all the cached data is written to the result table if the number of input data records does not reach the batchSize value within 3,000 milliseconds.
ignoreDelete	Specifies whether to ignore delete operations.	No	Default value: false. This value indicates that the delete operations are supported.
replaceMode	Specifies whether to use the REPLACE INTO statement to insert data into the table.	No	Valid values: • true: The REPLACE INTO statement is used to insert data into the table. This is the default value. • false: The INSERT INTO ON DUPLICATE KEY statement is used to insert data into the table. **Note** This parameter is valid only when the following conditions are met: • The Blink version is 3.X or later. • The AnalyticDB for MySQL version is 3.1.3.5 or later.

Parameter	Description	Required	Remarks
excludeUpdateColumns	Specifies the columns that are not updated when data with the same primary key is updated.	No	If you specify multiple columns that are not updated, separate them with commas (,). Example: `excludeUpdateColumns=column1,column2`.
reserveMilliSecond	Specifies whether to reserve the millisecond component in a value of the TIMESTAMP data type.	No	Default value: false. This indicates that the millisecond component is not reserved.

2.5.3.9. Create a Log Service result table

This topic describes how to create a Log Service result table in Realtime Compute for Apache Flink.

> **Notice**
> - This topic applies only to Blink 1.4.5 and later.
> - Log Service result tables support only fields of the VARCHAR type.

Introduction to Log Service

Log Service is an end-to-end service for log data. It helps you quickly collect, consume, ship, query, and analyze data to improve O&M and operations efficiency and build up the capabilities to process a large number of logs. Log Service stores streaming data. Therefore, Realtime Compute for Apache Flink can use Log Service tables as result tables for the processing of streaming data.

DDL syntax

In Realtime Compute for Apache Flink, you can use Log Service to store output data. The following code shows an example:

```
create table sls_stream(
 `name` VARCHAR,
 age BIGINT,
 birthday BIGINT
)with(
 type='sls',
 endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 project='<yourProjectName>',
 logstore='<yourLogstoreName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
endPoint	The endpoint of Log Service.	Yes	Contact Alibaba Cloud O&M engineers.
project	The name of a project.	Yes	None.
logstore	The name of the table in the database.	Yes	None.
accessId	AccessKey ID	Yes	None.
accessKey	AccessKey Secret	Yes	None.
topic	An attribute field.	No	This parameter is empty by default. You can use the selected field as the `topic` attribute field.
timestampColumn	An attribute field.	No	This parameter is empty by default. You can use the selected field as the `timestamp` attribute field. The data type of this parameter must be INT. If no field is selected, the current time is used as the attribute field. **Note** This parameter is available in Realtime Compute for Apache Flink V2.2.2 and later.

Parameter	Description	Required	Remarks
source	An attribute field. The source of a log entry. For example, the value of this parameter can be the IP address of the server where the log entry is generated.	No	This parameter is empty by default. You can use the selected field as the `source` attribute field.
partitionColumn	The partition key column.	No	This parameter is required if the `mode` parameter is set to `partition`.
flushIntervalMs	The interval at which data writing is triggered.	No	Default value: 2000. Unit: milliseconds.
reserveMilliSecond	Specifies whether to reserve the millisecond component in a value of the TIMESTAMP data type.	No	Default value: false. This value indicates that the millisecond component is not reserved. **Note** This parameter is available in Realtime Compute for Apache Flink V2.2.6 and later.

Field type mapping

The following table describes the mapping between the data types of Log Service and Realtime Compute for Apache Flink. We recommend that you declare the mapping in DDL statements.

Data type of Log Service	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

Sample code

The following sample code demonstrates how to create a Log Service result table in a Realtime Compute for Apache Flink job:

```
CREATE TABLE random_input (
  a VARCHAR,
  b VARCHAR) with (
    type = 'random'
);
create table sls_output(
 a varchar,
 b varchar
)with(
 type='sls',
 endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 project='ali-cloud-streamtest',
 logStore='stream-test2'
);
INSERT INTO sls_output
SELECT a, b
FROM random_input;
```

FAQ

Q: How do I specify the topic field in a Log Service result table?

A: You can specify the topic field as a field in the result table. For example, specify `topic='age'` in the sample code. After the configuration is completed, the value of the `age` field is written into Log Service but Log Service does not write the `age` field into the downstream storage systems.

2.5.3.10. Create a Tablestore result table

This topic describes how to create a Tablestore result table in Realtime Compute for Apache Flink. This topic also describes the mappings between the field data types of Tablestore and Realtime Compute for Apache Flink.

> **Notice** This topic applies only to Blink 1.4.5 and later.

What is Tablestore?

Tablestore is a distributed NoSQL database service that is built on the Apsara distributed operating system of Alibaba Cloud. Tablestore adopts sharding and load balancing technologies to scale out services and handle concurrent transactions. You can use Tablestore to store and query large amounts of structured data in real time.

DDL syntax

In Realtime Compute for Apache Flink, you can use Tablestore to store output data. The following code shows an example:

```
CREATE TABLE stream_test_hotline_agent (
 name VARCHAR,
 age BIGINT,
 birthday BIGINT,
 PRIMARY KEY (name,age)
) WITH (
 type='ots',
 instanceName='<yourInstanceName>',
 tableName='<yourTableName>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessSecret>',
 endPoint='<yourEndpoint>',
 valueColumns='birthday'
);
```

Note

- The value of the valueColumns parameter cannot be a declared primary key.
- The declared Tablestore result table must contain at least one attribute column and the primary key column.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the result table.	Set the value to ots.
instanceName	The name of a Tablestore instance.	N/A.
tableName	The name of the table in the database.	N/A.
endPoint	The endpoint of the instance.	Contact Alibaba Cloud O&M engineers to obtain the required information.
accessId	The AccessKey ID that is used to access Tablestore.	N/A.
accessKey	The AccessKey secret that is used to access Tablestore.	N/A.
valueColumns	The name of a column that you want to insert.	Separate multiple column names with commas (,), such as `'ID,NAME'`.

Parameter	Description	Remarks
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	Optional. Default value: 5000. This value indicates that data deduplication is triggered if the number of input data records in the buffer reaches 5,000. **Note** Realtime Compute for Apache Flink removes data record duplicates based on the primary key of the Tablestore result table. You can set bufferSize to the number of data record duplicates that are removed. Then, Realtime Compute for Apache Flink writes the data records after duplicates are removed. You can set batchSize to the number of data records that are written at a time.
batchWriteTimeoutMs	The write timeout period.	Optional. Default value: 5000. Unit: milliseconds. This value indicates that if the number of input data records does not reach the value specified by the batchSize parameter within 5,000 milliseconds, all cached data is written to the result table.
batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 100.
retryIntervalMs	The retry interval.	Optional. Default value: 1000. Unit: milliseconds.
maxRetryTimes	The maximum number of retries that are allowed to write data to the table.	Optional. Default value: 100.
ignoreDelete	Specifies whether to ignore delete operations.	Default value: False.

Data type mappings

Data type of Tablestore	Data type of Realtime Compute for Apache Flink
INTEGER	BIGINT
STRING	VARCHAR
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

Note You must define a `primary key` in a Tablestore result table. Output data is appended to the Tablestore result table to update the result.

2.5.3.11. Create a MaxCompute result table

This topic describes how to create a MaxCompute result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause, data type mapping, and FAQ involved when you create a MaxCompute result table.

Notice

- Only Blink 1.5.1 and later versions support MaxCompute result tables.
- A clustered table of MaxCompute cannot be used as a result table.

Implementation

The MaxCompute sink works in two phases:

1. Writes data. The MaxCompute sink calls an API operation in the MaxCompute SDK to write data into the buffer. Then, the sink uploads data to the temporary files of MaxCompute at a specified interval or when the data size in the buffer exceeds 64 MB.
2. Commits sessions. When a task runs a checkpoint, the MaxCompute sink calls the COMMIT method of Tunnel to commit sessions, moves temporary files to the data directory of the MaxCompute table, and modifies the metadata.

   Note The COMMIT method does not provide atomicity. Therefore, the MaxCompute sink supports at-least-once delivery instead of exactly-once delivery.

Syntax

In Realtime Compute for Apache Flink, you can use MaxCompute to store output data. The following code shows an example:

Note The names, sequence, and types of the fields defined in the data definition language (DDL) statement must be the same as those in the MaxCompute physical table. Otherwise, the queried data in the MaxCompute physical table may be `/n`.

```
create table odps_output(
    id INT,
    user_name VARCHAR,
    content VARCHAR
) with (
    type = 'odps',
    endPoint = '<YourEndPoint>',
    project = '<YourProjectName>',
    tableName = '<YourtableName>',
    accessId = '<yourAccessKeyId>',
    accessKey = '<yourAccessKeySecret>',
    `partition` = 'ds=2018****'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to `odps`.
endPoint	The endpoint of MaxCompute.	Yes	Contact Alibaba Cloud O&M engineers.
tunnelEndpoint	The endpoint of the MaxCompute Tunnel service.	Yes	Contact Alibaba Cloud O&M engineers.
project	The name of a MaxCompute project.	Yes	None.
tableName	The name of a table.	Yes	None.
accessId	AccessKey ID	Yes	None.
accessKey	AccessKey Secret	Yes	None.

Parameter	Description	Required	Remarks
partition	The name of a partition.	No	This parameter is required if a partitioned table exists. • Fixed partition For example, `` `partition` = 'ds=20180905' `` indicates that data is written into partition `ds= 20180905`. • Dynamic partition (available in Blink 3.2.1 and later) If the partition values are not displayed in plaintext mode, data is written into different partitions based on the values of the partition key columns specified in the data. For example, `` `partition`='ds' `` indicates that data is written into partitions based on the value of the `ds` field. If you want to create multi-level dynamic partitions, make sure that the sequence of the partition fields in the WITH clause and DDL statement of the MaxCompute result table are consistent with the field sequence of the MaxCompute physical table. Multiple partition fields are separated by commas (,). **Note** ◦ In the CREATE TABLE statement, you must explicitly specify the partition key column that you use to create dynamic partitions. ◦ If the partition field for dynamic partitions is left empty and the value of the ds field is `null` or `''`, the output varies based on the Blink version: ▪ For Blink 3.2.1 and earlier, a NullPointerException (NPE) error is returned. ▪ For Blink 3.2.2 and later, a partition with ds=null is created.

Parameter	Description	Required	Remarks
flushIntervalMs	The flush interval for the buffer of a writer in MaxCompute Tunnel. Unit: milliseconds. The MaxCompute sink first inserts data into its buffer. Then, the MaxCompute sink writes the data in the buffer to the destination MaxCompute table at an interval that is specified by the flushIntervalMs parameter. The sink also writes the data to the destination table when the size of the buffer data exceeds the specified threshold.	No	Default value: 30000. Unit: milliseconds. **Note** This parameter is available in Blink 3.6.0 and later.
partitionBy	The columns based on which hash shuffling is implemented. Before data is written into a sink node, the system implements hash shuffling based on the value of this parameter. This way, data flows to the destination sink node.	No	The system implements hash shuffling based on multiple columns. The column names are separated by commas (,). This parameter is empty by default. **Note** This parameter is available in Blink 3.6.0 and later.

Parameter	Description	Required	Remarks
isOverwrite	Specifies whether to clear the result table before data is written to a sink node.	No	• For versions earlier than Blink 3.2, the default value is true. • For Blink 3.2 and later, the default value is false. When you declare a MaxCompute result table for a streaming job, you must set the isOverwrite parameter to false. Otherwise, an error is returned during compilation. **Note** For Blink 3.2 and later, you can change the value of the isOverwrite parameter to true. For Blink versions earlier than 3.2, if you want to change the value of the isOverwrite parameter, you must upgrade the Blink version first.
dynamicPartitionLimit	The number of partitions in the underlying framework.	No	The default value is 100. A map in the memory maintains the mappings between the existing partitions to which data is written and the Tunnel service or the writer. If the map size exceeds the value of the dynamicPartitionLimit parameter, the system reports the following error: `Too many dynamic partitions: 100, which exceeds the size limit: 100` .

Field type mapping

MaxCompute data type	Data type of Realtime Compute for Apache Flink
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
STRING	VARCHAR

MaxCompute data type	Data type of Realtime Compute for Apache Flink
DECIMAL	DECIMAL
BINARY	VARBINARY

Sample code

The following sample code demonstrates how to create a MaxCompute result table in a Realtime Compute for Apache Flink job.

- Write data into a fixed partition

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'random'
);
create table odps_sink (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'odps',
   endPoint = '<yourEndpoint>',
   project = '<yourProjectName>',
   tableName = '<yourTableName>',
   accessId = '<yourAccessId>',
   accessKey = '<yourAccessPassword>',
   `partition` = 'ds=20180418'
);
INSERT INTO odps_sink
SELECT
   id, len, content
FROM source;
```

- Write data into a dynamic partition

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR,
   c TIMESTAMP
) with (
   type = 'random'
);
create table odps_sink (
   id INT,
   len INT,
   content VARCHAR,
   ds VARCHAR --The partition key column that you use to create dynamic partitions must b
e explicitly specified in the CREATE TABLE statement.
) with (
   type = 'odps',
   endPoint = '<yourEndpoint>',
   project = '<yourProjectName>',
   tableName = '<yourTableName>',
   accessId = '<yourAccessId>',
   accessKey = '<yourAccessPassword>',
   `partition`='ds' --The partition value is not provided. This means that data is writte
n into different partitions based on the value of the ds field.
);
INSERT INTO odps_sink
SELECT
   id,
   len,
   content,
   DATE_FORMAT(c, 'yyMMdd') as ds
FROM source;
```

FAQ

- What do the endPoint and tunnelEndpoint parameters specify in the Alibaba Cloud public cloud? What happens if the parameter configurations are invalid?

  For more information about the endPoint and tunnelEndpoint parameters, see Configure endpoints. If the configuration of the two parameters is invalid in a VPC, a task exception may occur.

  - If the configuration of the endPoint parameter is invalid, the task stops at the progress of 91%.
  - If the configuration of the tunnelEndpoint parameter is invalid, the task fails.

- Q: Does Realtime Compute for Apache Flink clear a MaxCompute result table before it writes data into the result table in stream mode when isOverwrite is set to true?

  A: Blink versions earlier than 3.2 support this feature. Blink 3.2 and later versions do not support this feature.

  If the isOverwrite parameter is set to true, Realtime Compute for Apache Flink clears a MaxCompute result table before it writes data into the result table. Realtime Compute for Apache Flink clears data from the existing result table or the result partition each time a job is started or resumed, or before Realtime Compute for Apache Flink writes data into the result table.

- For Blink versions earlier than 3.2, the default value of isOverwrite is true and cannot be changed. Data may be lost after a streaming job is suspended or resumed.
- For Blink 3.2 and later, the default value of isOverwrite is false. If you declare a MaxCompute result table for a streaming job, you must set the isOverwrite parameter to false. Otherwise, an error is returned during compilation. MaxCompute result tables in stream mode support the at-least-once data security mechanism. If a job fails, duplicate data may be generated.

2.5.3.12. Create an Elasticsearch result table

This topic describes how to create an Elasticsearch result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause used when you create an Elasticsearch result table.

> **Notice** This topic applies only to Blink 3.2.2 and later.

DDL syntax

In Realtime Compute for Apache Flink, you can use Elasticsearch to store output data. The following code shows an example:

```
 CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1)
)WITH(
  type ='elasticsearch',
  endPoint = 'http://es-cn-mp****.public.elasticsearch.aliyuncs.com:****',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

> **Note**

- Elasticsearch supports data updates based on the PRIMARY KEY field. You can specify only one field for the PRIMARY KEY field.
- After you specify the PRIMARY KEY field, values in the field are used as document IDs.
- If you do not specify the PRIMARY KEY field, document IDs are randomly generated. For more information, see Index API.
- In full update mode, later documents overwrite earlier documents.
- In incremental update mode, the system updates the fields based on the entered field values.
- All updates use the UPSERT syntax by default, which means to insert or update data.

Parameters in the WITH clause (general configurations)

Parameter	Description	Default value	Required
type	The type of the connector.	elasticsearch	Yes
endPoint	The server address of an Elasticsearch cluster, for example, *http://127.0.0.1:9211*.	None	Yes
accessId	The AccessKey ID that is used to access an Elasticsearch cluster. **Note** If you use the Kibana plug-in to connect to Elasticsearch, enter the Kibana logon ID.	None	Yes
accessKey	The AccessKey secret that is used to access an Elasticsearch cluster. **Note** If you use the Kibana plug-in to connect to Elasticsearch, enter the Kibana logon password.	None	Yes
index	The index name, which is similar to the database name.	None	Yes
typeName	The type name, which is similar to the database table name.	None	Yes
bufferSize	The maximum number of data records that can be stored in the buffer before deduplication is triggered.	1000	No
maxRetryTimes	The maximum number of retries for writing data into a table.	30	No
timeout	The read timeout period. Unit: milliseconds.	600000	No
discovery	Specifies whether node discovery is enabled. If this feature is enabled, the client refreshes the server list every five minutes.	false	No
compression	Specifies whether to compress request bodies in the GZIP format.	true	No
multiThread	Specifies whether to enable multithreading for JestClient.	true	No

Parameter	Description	Default value	Required
ignoreWriteError	Specifies whether to ignore write exceptions.	false	No
settings	The settings used to create indexes.	None	No
updateMode	The update mode used after the primary key is specified. • full: Full data is overwritten. • inc: Incremental data is updated.	full	No

Parameters in the WITH clause (dynamic indexing)

Parameter	Description	Default value	Required
dynamicIndex	Specifies whether to enable dynamic indexing. • true: Dynamic indexing is enabled. • false: Dynamic indexing is disabled.	false	No.
indexField	The field name of the index.	None	This parameter is required when dynamicIndex is set to true. This parameter supports only the TIMESTAMP (in seconds), DATE, and LONG data types.
indexInterval	The interval at which an index is changed.	d	This parameter is required when dynamicIndex is set to true. Valid values: • d: day • m: month • w: week

Parameter	Description	Default value	Required
mapping	The mapping between the types and formats of fields in a document configured when dynamic indexing is enabled.	Empty string	No. Blink 3.7.0 and later versions support this parameter. The following example shows how to set the mapping between the type and format of the sendTime field: `{ "properties": { "sendTime": { "type": "date", "format": "yyyy-MM-dd HH:mm:ss" } } }`

Note

- Only Blink 2.2.7 and later versions support the dynamic indexing feature.
- After dynamic indexing is enabled, the `index` name in the basic configuration is used as the unified alias for indexes created subsequently. An alias can correspond to multiple indexes.
- Actual index names that correspond to different values of `indexInterval`:
  - d -> Alias "yyyyMMdd"
  - m -> Alias "yyyyMM"
  - w -> Alias "yyyyMMW"
- You can use Index API to change a single actual index, but the alias supports only the `GET` method. If you want to change the alias, see Index Aliases.

2.5.3.13. Create a Message Queue for Apache Kafka result table

This topic describes how to create a Message Queue for Apache Kafka result table in Realtime Compute for Apache Flink. It also describes the mapping between the values of the type parameter and Kafka versions.

> **Notice**
>
> - This topic applies only to Realtime Compute for Apache Flink V2.0 and later.
> - This topic applies only to Realtime Compute for Apache Flink in exclusive mode.
> - Data of a Message Queue for Apache Kafka result table can be written into a self-managed Kafka cluster. Before data is written, you must pay attention to the mapping between the values of the type parameter and Kafka versions, and the network configurations of the self-managed Kafka cluster and the Realtime Compute for Apache Flink cluster.

Introduction to the Message Queue for Apache Kafka result table

Message Queue for Apache Kafka is a distributed, high-throughput, and scalable message queue service provided by Alibaba Cloud. This service is widely used in big data fields, such as log collection, monitoring data aggregation, streaming data processing, and online and offline data analysis. Realtime Compute for Apache Flink allows you to create source tables and result tables of Message Queue for Apache Kafka for the processing of streaming data.

DDL syntax

The following example demonstrates how to create a Message Queue for Apache Kafka result table in a data definition language (DDL) statement.

```
create table sink_kafka (
  messageKey VARBINARY,
  `message` VARBINARY,
  PRIMARY KEY (messageKey)
) with (
  type = 'kafka010',
  topic = '<yourTopicName>',
  bootstrap.servers = '<yourServerAddress>'
);
```

> **Note**

- When you create a Message Queue for Apache Kafka result table, you must specify `PRIMARY KEY (messageKey)` in plaintext mode.
- Only Blink 2.2.6 and later versions support the display of metrics such as transactions per second (TPS) and requests per second (RPS) of Alibaba Cloud Message Queue for Apache Kafka or a self-managed Kafka database.

Parameters in the WITH clause

- General configurations

Parameter	Description	Required	Remarks
type	The Kafka version.	Yes	Valid values: Kafka08, Kafka09, Kafka010, and Kafka011. For more information, see Mapping between the values of the type parameter and Kafka versions.

Parameter	Description	Required	Remarks
topic	The name of the topic.	Yes	None.

- Required configurations
  - Required configurations for Kafka08

Parameter	Description
zookeeper.connect	The ZooKeeper URL.

  - Required configurations for Kafka09, Kafka010, and Kafka011

Parameter	Description
bootstrap.servers	The Kafka cluster address.

- Optional configurations
  - `consumer.id`
  - `socket.timeout.ms`
  - `fetch.message.max.bytes`
  - `num.consumer.fetchers`
  - `auto.commit.enable`
  - `auto.commit.interval.ms`
  - `queued.max.message.chunks`
  - `rebalance.max.retries`
  - `fetch.min.bytes`
  - `fetch.wait.max.ms`
  - `rebalance.backoff.ms`
  - `refresh.leader.backoff.ms`
  - `auto.offset.reset`
  - `consumer.timeout.ms`
  - `exclude.internal.topics`
  - `partition.assignment.strategy`
  - `client.id`
  - `zookeeper.session.timeout.ms`
  - `zookeeper.connection.timeout.ms`
  - `zookeeper.sync.time.ms`
  - `offsets.storage`
  - `offsets.channel.backoff.ms`
  - `offsets.channel.socket.timeout.ms`

- `offsets.commit.max.retries`
- `dual.commit.enabled`
- `partition.assignment.strategy`
- `socket.receive.buffer.bytes`
- `fetch.min.bytes`

> **Note** For more information about optional configuration items, see the following official Kafka documentation:
>
> - Kafka09
> - Kafka010
> - Kafka011

Mapping between the values of the type parameter and Kafka versions

type	Kafka version
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2 and later

Example

```
create table datahub_input (
  id VARCHAR,
  nm VARCHAR
) with (
  type = 'datahub'
);
create table sink_kafka (
  messageKey VARBINARY,
  `message` VARBINARY,
  PRIMARY KEY (messageKey)
) with (
  type = 'kafka010',
  topic = '<yourTopicName>',
  bootstrap.servers = '<yourServerAddress>'
);
INSERT INTO
  sink_kafka
SELECT
  cast(id as VARBINARY) as messageKey,
  cast(nm as VARBINARY) as `message`
FROM
  datahub_input;
```

2.5.3.14. Create an ApsaraDB RDS for SQL Server result table

This topic describes how to create an ApsaraDB RDS for SQL Server result table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, data type mappings, and Java Database Connectivity (JDBC) parameters that are used when you create such a result table.

> **Notice**
> - This topic applies only to Blink V3.2.0 and later.
> - Realtime Compute for Apache Flink cannot use ApsaraDB RDS for SQL Server as a data store.

DDL syntax

In Realtime Compute for Apache Flink, you can use ApsaraDB RDS for SQL Server to store output data. The following code shows an example:

```
create table ss_output(
 id INT,
 len INT,
 content VARCHAR,
 primary key(id,len)
) with (
 type='jdbc',
 url='jdbc:sqlserver://ip:port;database=****',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

Note

- In Realtime Compute for Apache Flink, each row of output data is converted to a line of SQL statement and then written and executed in the destination database. If you want to write multiple rows of data to the result table at the same time, you must add ?rewriteBatchedStatements=true to the URL to improve system performance.
- You can define an auto-increment primary key for an ApsaraDB RDS for SQL Server database. If you want to use the auto-increment primary key, do not declare the auto-increment field in the DDL statement. For example, if you use the ID field as an auto-increment field, do not declare the ID field in the DDL statement. When a row of output data is written into an ApsaraDB RDS for SQL Server database, a value is automatically filled for the auto-increment field.
- If a DRDS result table has partitions, the shard key must be declared in primary key() of the DDL statement. Otherwise, you cannot write data into the partitioned table.
- The fields that are declared in a DDL statement must include at least one non-primary key field. Otherwise, an error is returned.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
url	The JDBC URL of the database.	Yes	N/A.
tableName	The name of the table.	Yes	N/A.
username	The username that is used to access the database.	Yes	N/A.
password	The password that is used to access the database.	Yes	N/A.

Parameter	Description	Required	Remarks
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 10.
bufferSize	The maximum number of data records that can be stored in the buffer before data deduplication is triggered.	No	Default value: 10000. This value indicates that deduplication is triggered after the number of input data records reaches 10,000.
flushIntervalMs	The interval at which the cache is cleared.	No	Unit: milliseconds. Default value: 2000. This value indicates that if the number of input data records does not reach the value specified by the bufferSize parameter within 2,000 milliseconds, all cached data is written into the result table.
excludeUpdateColumns	Specifies whether to ignore the update of the specified field.	No	This parameter is optional. This parameter is empty by default, which indicates that the primary key field is ignored by default. When data with the same primary key is updated, the specified columns are not updated.
ignoreDelete	Specifies whether to ignore delete operations.	No	Default value: False.

Data type mappings

Data type of ApsaraDB RDS for SQL Server	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL

Data type of ApsaraDB RDS for SQL Server	Data type of Realtime Compute for Apache Flink
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

JDBC parameters

Parameter	Description	Default value	Since version (JDBC driver)
useUnicode	Specifies whether to use the Unicode character set. This parameter must be set to True if you set the characterEncoding parameter to GB2312 or GBK.	False	1.1g
characterEncoding	The character encoding format, such as GB2312 or GBK. If useUnicode is set to true, you must specify a character encoding format.	False	1.1g
autoReconnect	Specifies whether to automatically re-establish a connection when the connection to the database is unexpectedly interrupted.	False	1.1
autoReconnectForPools	Specifies whether to use the reconnection policy for a database connection pool.	False	3.1.3
failOverReadOnly	Specifies whether the database is read-only after it is automatically reconnected.	True	3.0.12

Parameter	Description	Default value	Since version (JDBC driver)
maxReconnects	The maximum number of reconnection attempts allowed. This parameter must be set if the autoReconnect parameter is set to True.	3	1.1
initialTimeout	The interval between two reconnection attempts. Unit: seconds. This parameter must be set if the autoReconnect parameter is set to True.	2	1.1
connectTimeout	The timeout period when you use a socket connection to access the database server. Unit: milliseconds.	Default value: 0. This value indicates that the connection never times out. This parameter is provided in JDK V1.4 and later.	3.0.1
socketTimeout	The timeout period for read and write operations on a socket connection. Unit: milliseconds.	Default value: 0. This value indicates that the read or write operation never times out.	3.0.1

Sample code

The following example describes how to create an ApsaraDB RDS for SQL Server result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'random'
);
CREATE TABLE rds_output(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id,len)
) WITH (
 type='jdbc',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTable>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT id, len, content FROM source;
```

FAQ

- Q: When the output data of Realtime Compute for Apache Flink is written to an ApsaraDB RDS for SQL Server table, is the result table updated based on the primary key or is a new data record generated in the table?

  A: The processing method depends on whether the primary key is defined in the DDL statement.

  - If a primary key is defined in the DDL statement, the result table is updated by using `insert into on duplicate key update`. For a data record, if the primary key does not exist, the record is inserted into the table as a new row. If the value of the primary key field exists, the original row in the table is updated.
  - If no primary key is defined in the DDL statement, new data records are appended to the table by using `insert into`.

- Q: What do I need to pay attention to when I perform GROUP BY operations based on the unique index of an ApsaraDB RDS for SQL Server table?

  A: Pay attention to the following points:

  - Declare the unique index in the primary key of your job.
  - An ApsaraDB RDS for SQL Server table has only one auto-increment primary key. Therefore, this auto-increment primary key cannot be declared as the primary key in a Realtime Compute for Apache Flink job.

2.5.3.15. Create an ApsaraDB for MongoDB result table

This topic describes how to create an ApsaraDB for MongoDB result table in Realtime Compute for Apache Flink. This parameter also describes the parameters in the WITH clause that is used when you create an ApsaraDB for MongoDB result table.

> **Notice**
>
> - This topic applies to only Blink V3.2.2 and later.
> - You cannot update the primary key in an ApsaraDB for MongoDB result table. As a result, data that has the same primary key is inserted repeatedly into the table.

Syntax

In Realtime Compute for Apache Flink, you can use ApsaraDB for MongoDB to store output data. To create an ApsaraDB for MongoDB result table, you can use the following sample code:

```
CREATE TABLE mongodb_sink (
  `a`             VARCHAR
) WITH (
   type = 'mongodb',
   database = '<yourDatabaseName>',
   collection= '<yourCollectionName>',
   uri='mongodb://{<databaseAccount>}:{<atabasePassword>}@{host}:****? replicaSet=mgset-122
4****',
   keepAlive='true',
   maxConnectionIdleTime='20000',
   batchSize='2000'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the connector.	Yes	Set the value to mongodb.
database	The name of the ApsaraDB for MongoDB database.	Yes	None.
collection	The set of result table data.	Yes	None.
uri	The connection string of the ApsaraDB for MongoDB database.	Yes	None.
keepAlive	Specifies whether to maintain the persistent connection.	No	Default value: true.

Parameter	Description	Required	Remarks
maxConnectionIdleTime	The time-out duration of the connection.	No	The integer value. Unit: milliseconds. This parameter cannot be set to a negative value. Default value: 60000. If this parameter is set to 0, the connection does not time out.
batchSize	The number of data records that can be written at a time.	No	The integer value. Default value: 1024. The system sets the maximum number of data records that can be stored in the buffer. When the number of data records reaches the specified value of batchSize, the system triggers the data output. **Note** When the checkpoint time arrives, the data output is triggered even if the number of data records in the buffer does not reach the specified value of batchSize.

2.5.3.16. Create an Oracle database result table

This topic describes how to create an Oracle database result table. It also describes the parameters in the WITH clause, data type mapping, and precautions.

Notice

- This topic applies only to Blink 3.6.0 and later.
- You can create Oracle database result tables only when you use Oracle 19c.

Precautions

- In Realtime Compute for Apache Flink, an SQL statement is executed to write each row of result data to the result table in the destination database.
- Realtime Compute for Apache Flink performs the following operations based on whether the required table exists in the Oracle database:
  - If the required table does not exist, Realtime Compute for Apache Flink creates a table in the Oracle database to store the result data.
  - If the required table exists, Realtime Compute for Apache Flink writes or updates data to the result table.
- The primary keys of logical and physical tables can be different. The primary keys of logical tables must contain the primary keys of physical tables.
- Realtime Compute for Apache Flink uses the APPEND function or the UPSERT function to insert data into an Oracle database result table.
  - If no primary key is specified in the table, the APPEND function is used.

- If a primary key is specified in the table, the UPSERT function is used. If the primary key does not exist, the primary key is inserted. If the primary key exists, the primary key is updated.

Syntax

In Realtime Compute for Apache Flink, you can use an Oracle database to store output data. The following code shows an example:

```
CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employee_age INT,
  PRIMARY KEY(employee_id)
) WITH (
    type = 'oracle',
    url = '<yourUrl>',
    userName = '<yourUserName>',
    password = '<yourPassword>',
    tableName = '<yourTableName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to *oracle*.
url	The connection string of the database.	Yes	`Contact Alibaba Cloud O&M engineers.`
userName	The username that is used to log on to the database.	Yes	None.
password	The password that is used to log on to the database.	Yes	None.
tableName	The name of the table in the database.	Yes	None.
maxRetryTimes	The maximum number of retries for writing data to a table.	No	Default value: 10.

Parameter	Description	Required	Remarks
batchSize	The number of data records that are written at a time. **Note** • The batchSize parameter takes effect only when the primary key is defined. • Write operations are triggered if the value of the batchSize or bufferSize parameter reaches the specified threshold.	No	Default value: 50.
bufferSize	The number of data records in the buffer after duplicates are removed. **Note** • The bufferSize parameter takes effect only when the primary key is defined. • Write operations are triggered if the value of the batchSize or bufferSize parameter reaches the specified threshold.	No	Default value: 500.
flushIntervalMs	The write timeout period. Unit: milliseconds. **Note** If no data is written to the database within the period specified by this parameter, the system writes the cached data to the result table again.	No	Default value: 500.
excludeUpdateColumns	Specifies whether to skip the specified columns when the data of a row in the table is updated. **Note** When the data of a row in the table is updated, the primary key is not updated by default.	No	This parameter is empty by default.

Parameter	Description	Required	Remarks
ignoreDelete	Specifies whether to skip delete operations.	No	Default value: false.

Mapping between field data types

Data type of the Oracle database	Data type of Realtime Compute for Apache Flink
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

Sample code

The following sample code shows how to create an Oracle database result table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE oracle_source (
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT
) WITH (
  type = 'random'
);
CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT,
  primary key(employee_id)
)with(
  type = 'oracle',
  url = 'jdbc:oracle:thin:@192.168.171.62:1521:sit0',
  userName = 'blink_test',
  password = 'blink_test',
  tableName = 'oracle_sink'
);
INSERT INTO oracle_sink
SELECT * FROM oracle_source;
```

2.5.3.17. Create an InfluxDB result table

This topic describes how to create an InfluxDB result table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH clause and data type mapping involved when you create an InfluxDB result table.

> **Notice**
> - InfluxDB does not support the storage registration feature.
> - This topic applies only to Blink 3.5.0-hotfix and later.

DDL syntax

In Realtime Compute for Apache Flink, you can use InfluxDB to store output data. The following code shows an example:

```
create table stream_test_influxdb(
    `metric` varchar,
    `timestamp` BIGINT,
    `tag_value1` varchar,
    `field_fieldValue1` Double
)with(
    type = 'influxdb',
    endpoint = 'http://service.cn.influxdb.aliyuncs.com:****',
    database = '<yourDatabaseName>',
    batchPutsize = '1',
    username = '<yourDatabaseUserName>',
    password = '<yourDatabasePassword>'
);
```

Default format for the created table:

- Column 0: metric (VARCHAR). This column is required.
- Column 1: timestamp (BIGINT). This column is required. Unit: milliseconds.
- Column 2: tag_value1 (VARCHAR). This column is required. You must enter at least one value in this column.
- Column 3: field_fieldValue1 (DOUBLE). This column is required. You must enter at least one value in this column.

  To specify multiple field_fieldValue values, use the following format:

  ```
  field_fieldValue1 <Data type>,
  field_fieldValue2 <Data type>,
  ...
  field_fieldValueN <Data type>
  ```

  The following code shows an example:

  ```
  field_fieldValue1 Double,
  field_fieldValue2 INTEGER,
  ...
  field_fieldValueN INTEGER
  ```

> **Note** An InfluxDB result table can contain only metric, timestamp, tag_*, and field_*.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to InfluxDB.
endpoint	The endpoint of the InfluxDB database.	Yes	Contact Alibaba Cloud O&M engineers.
database	The name of the InfluxDB database.	Yes	For example, you can set this parameter to db-blink or blink.
batchPutSize	The number of data records that are submitted at a time.	No	Default value: 500.
username	The username that is used to access the InfluxDB database.	Yes	You must have the write permission on the InfluxDB database.
password	The password that is used to access the InfluxDB database.	Yes	Default value: 0.
retentionPolicy	The retention policy.	No	If this parameter is empty, the default retention policy is used for each database.

Field type mapping

InfluxDB data type	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

InfluxDB data type	Data type of Realtime Compute for Apache Flink
VARCHAR	VARCHAR

2.5.3.18. Create a Phoenix5 result table

This topic describes how to create a Phoenix5 result table in Realtime Compute for Apache Flink.

> **Notice**
>
> - This topic applies only to Realtime Compute for Apahce Flink in exclusive node.
> - This topic applies only to Blink 3.4.0 and later.
> - Only Phoenix 5.X is supported.
> - Phoenix is an HBase SQL service deployed on an ApsaraDB for HBase instance. You can use Phoenix only after you activate this service in ApsaraDB for HBase instances.

DDL syntax

In Realtime Compute for Apache Flink, you can use Phoenix5 to store output data. The following code shows an example:

```
create table US_POPULATION_SINK (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY)--- The primary key. This field is required.
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to `PHOENIX5`.

Parameter	Description	Required	Remarks
serverUrl	The URL of the Phoenix5 query server. • If Phoenix5 is created in a cluster, the value of this parameter is the URL of Server Load Balancer (SLB). • If Phoenix5 is created on a single server, the value of this parameter is the URL of the server.	Yes	You must enable the HBase SQL service in an ApsaraDB for HBase instance. The value of the serverUrl parameter is in the http://host:port format. In the format: • host: indicates the domain name of the Phoenix5 service. • port: indicates the port number of the Phoenix5 service. Set the value to 8765.
tableName	The name of the Phoenix5 table.	Yes	The name of the Phoenix5 table is in the SchemaName.TableName format. In the format: • SchemaName: indicates the schema name, which can be empty. This means that the schema name is not used and only the table name is used. In this case, the default schema of the database is used. • TableName: the name of the table.

Sample code

The following sample code shows how to create a Phoenix5 result table in a Realtime Compute for Apache Flink job.

```
create table `source` (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar
) WITH (
   type = 'random'
);
create table sink (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar,
  primary key (id)
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);
INSERT INTO sink
  SELECT  `id` ,`name` , `age` ,`birthday`
FROM `source`;
```

2.5.3.19. Create a Hologres result table

This topic describes how to create a Hologres result table. It also describes the parameters in the WITH clause, streaming semantics, and data type mapping involved when you create a Hologres result table.

Notice

- This topic applies only to Blink 3.6.0 and later.
- We recommend that you use Hologres 0.7 or later.
- Hologres writes data asynchronously. Therefore, you must add blink.checkpoint.fail_on_checkpoint_error=true to the code so that a failover is triggered only when a job exception occurs.

Introduction to Hologres

Hologres is compatible with the PostgreSQL protocol and integrates seamlessly with the big data ecosystem. Hologres supports real-time analysis and the processing of petabytes of data with high concurrency and low latency. This allows you to use existing Business Intelligence (BI) tools to easily perform multidimensional analysis and business exploration.

Syntax

In Realtime Compute for Apache Flink, you can use Hologres to store output data. The following code shows an example:

```
create table Hologres_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='<yourDbname>',
  tablename='<yourTablename>',
  username='<yourUsername>',
  password='<yourPassword>',
  endpoint='<yourEndpoint>',
  field_delimiter='|' -- This parameter is optional.
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to *hologres*.
dbname	The name of the database. **Note** If the public schema is not used, you must set tableName to schema.tableName.	Yes	N/A.
tablename	The name of the table.	Yes	N/A.
username	The username that is used to access the database.	Yes	N/A.
password	The password that is used to access the database.	Yes	N/A.
endpoint	The virtual private cloud (VPC) endpoint of Hologres.	Yes	Contact Alibaba Cloud O&M engineers to obtain the required information.
field_delimiter	The delimiter used between rows when data is being exported. **Notice** Do not insert delimiters in data. This parameter takes effect only when the bulkload semantics is used.	No	Default value: *"\u0002"*.
mutateType	The streaming write semantics. For more information, see Streaming data semantics.	No	Default value: *insertorignore*.

Parameter	Description	Required	Remarks
partitionrouter	Specifies whether to write data to a partitioned table.	No	Default value: *false*. **Note** For Blink 3.6.X, the Hologres streaming sink cannot automatically create partitioned tables. Before Realtime Compute for Apache Flink writes data to a partitioned table, you must manually create a sub-table in Hologres.
ignoredelete	Specifies whether to ignore retraction messages.	No	Default value: *false*. **Note** This parameter takes effect only after the streaming data semantics is used.
createPartTable	Specifies whether to automatically create a non-existent partitioned table to which data is written based on partition values. **Note** If the partition values contain hyphens (-), partitioned tables cannot be automatically created.	No	• false: Partitioned tables cannot be automatically created. This is the default value. • true: Partitioned tables can be automatically created. **Note** Only Blink versions later than 3.7 support this parameter.

Streaming data semantics

Stream processing, also known as streaming data or event processing, refers to the continuous processing of a series of unbounded data or events. In most cases, the system that processes streaming data or events allows you to specify a reliability pattern or processing semantics to ensure data accuracy. Network or device failures may cause a data loss.

Semantics can be classified into the following types based on the configurations of the Hologres streaming sink that you use and the attributes of the Hologres table:

- Exactly-once: The system processes data or events only once even multiple faults occur.
- At-least-once: If streaming data or events to be processed are lost, the system transfers the data again from the transmission source. Therefore, the system may process streaming data or events for multiple times. If the first retry succeeds, no further retries are required.

When you use streaming semantics in a Hologres result table, take note of the following points:

- If no primary keys have been configured in the Hologres physical table, the Hologres streaming sink uses the at-least-once semantics.
- If primary keys have been configured in the Hologres physical table, the Hologres streaming sink uses the exactly-once semantics based on the primary keys. If multiple records with the same primary key are written to the table, you must set the mutationType parameter to determine how the result table is updated. This parameter has the following valid values:
  - insertorignore (default value): Hologres keeps the first record and discards the subsequent records.
  - insertorreplace: Hologres completely replaces the existing record with the one that arrives later.
  - insertorupdate: Hologres partially replaces the existing record with the one that arrives later.

  > Note
  > - If the mutationType parameter is set to insertorupdate or insertorreplace, the system updates data based on the primary key.
  > - The number of columns in the result table defined by Blink can be different from the number of columns in the Hologres physical table. Make sure that the value null can be used to fill the missing columns. Otherwise, an error is returned.

- By default, the Hologres streaming sink can import data to only one non-partitioned table. If the sink imports data into the parent table of a partitioned table, data queries fail even if data import succeeds. To enable data to be automatically written to a partitioned table, you can set the partitionRouter parameter to true. Take note of the following points:
  - You must set tablename to the name of the parent table.
  - Blink connectors do not automatically create partitioned tables. We recommend that you create a partitioned table before you import data. Otherwise, the data fails to be imported.

Merge data into a wide table

If you need to write data from multiple streaming jobs to one Hologres wide table, you can merge the data into a wide table.

For example, one Flink data stream contains fields A, B, and C, the other contains fields A, D, and E. The Hologres wide table WIDE_TABLE contains fields A, B, C, D, and E, among which field A is the primary key. You can perform the following operations:

1. Execute Flink SQL statements to create two Hologres result tables. One table is used to declare

fields A, B, and C, and the other is used to declare fields A, D, and E. Map the two result tables to the Hologres wide table WIDE_TABLE.

2. Parameter settings of the two Hologres result tables:
   - Set mutatetype to insertorupdate. This indicates that data is updated based on the primary key.
   - Set ignoredelete to true. This prevents retraction messages from generating Delete requests.
3. Insert data from the two Flink data streams into the two result tables.

> **Note** Limits:
> - The wide table must have a primary key.
> - The data of each stream must contain all the fields in the primary key.
> - If the wide table is a column-oriented table and the requests per second (RPS) value is large, the CPU utilization increases. We recommend that you disable dictionary encoding for the fields in the table.

Data type mapping

Hologres	BLINK
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

2.5.3.20. Create an AnalyticDB for PostgreSQL result table

This topic describes how to create an AnalyticDB for PostgreSQL result table. This topic also describes the parameters in the WITH clause and data type mappings used when you create an AnalyticDB for PostgreSQL result table.

> **Notice** This topic applies to only Blink 3.6.0 and later.

Principles

Realtime Compute for Apache Flink writes data to an AnalyticDB for PostgreSQL result table in two steps:

1. Converts each row of output data to a line of SQL statement.
2. Writes and executes the SQL statement in the destination database.

DDL syntax

In Realtime Compute for Apache Flink, you can use AnalyticDB for PostgreSQL to store output data. The following sample code shows how to create an AnalyticDB for PostgreSQL result table.

```
create table rds_output(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY(id)
) with (
  type='adbpg',
  url='jdbc:postgresql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the source table.	Yes	Set the value to `adbpg`.
url	The Java Database Connectivity (JDBC) URL of the database.	Yes	Contact Alibaba Cloud O&M engineers to obtain the required information.
tableName	The name of the table.	Yes	N/A

Parameter	Description	Required	Remarks
username	The account that is used to access the AnalyticDB for PostgreSQL database.	Yes	N/A
password	The password that is used to access the AnalyticDB for PostgreSQL database.	Yes	N/A
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 3.
useCopy	Specifies whether to use the copy API to write data.	No	Valid values: • 0: indicates that the INSERT statement is executed to write data to the AnalyticDB for PostgreSQL database. This is the default value. • 1: indicates that the copy API is used to write data to the AnalyticDB for PostgreSQL database.
batchSize	The number of data records that can be written at a time.	No	Default value: 5000.
exceptionMode	The policy that is used to handle exceptions during data writing.	No	Valid values: • *ignore*: The system ignores the data that is written when exceptions occur. This is the default value. • *strict*: If an exception occurs during data writing, an error message appears on the Failover tab.
conflictMode	The policy that is used to handle primary key conflicts or unique index conflicts.	No	Valid values: • *ignore*: Primary key conflicts are ignored and the existing data is retained. This is the default value. • *strict*: If a primary key conflict occurs, an error message appears on the Failover tab. • *update*: If a primary key conflict occurs, data is updated.
targetSchema	The name of the schema.	No	Default value: public.

Parameter	Description	Required	Remarks
connectionMaxActive	The maximum number of connections allowed for a single task.	No	Configure this parameter based on the actual number of parallel tasks and the maximum number of connections allowed to the destination database.

Data type mapping

The following table lists the mappings between the field data types of AnalyticDB for PostgreSQL and Realtime Compute for Apache Flink.

Data type of AnalyticDB for PostgreSQL	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
SMALLINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
DOUBLE PRECISION	DOUBLE
TEXT	VARCHAR
TIMESTAMP	DATETIME
DATE	DATE
REAL	FLOAT
DOUBLE PRECISION	DECIMAL
TIME	TIME
TIMESTAMP	TIMESTAMP

2.5.3.21. Create an ApsaraDB for HBase result table

This topic describes how to create an ApsaraDB for HBase result table in Realtime Compute for Apache Flink.

> **Notice**
>
> - This topic applies only to Realtime Compute for Apache Flink in exclusive mode.
> - Blink versions earlier than 3.3.0 support only HBase Standard Edition.
> - Blink 3.3.0 and later versions support both HBase Standard Edition and HBase Enhanced Edition.
> - Blink 3.5.0 and later versions support switchover between primary and secondary ApsaraDB for HBase databases for data writing.
> - ApsaraDB for HBase result tables in Realtime Compute for Apache Flink do not support self-managed open source HBase.

DDL syntax

In Realtime Compute for Apache Flink, you can use ApsaraDB for HBase to store output data.

- The following sample code demonstrates how to create an ApsaraDB for HBase result table of HBase Standard Edition:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '2',
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- The following sample code demonstrates how to create an ApsaraDB for HBase result table of HBase Enhanced Edition:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    endPoint = '<host:port>', ----The Java API URL that is used to access the Enhanced Ed
ition of an ApsaraDB for HBase database.
    userName  = 'root', -- The username that is used to access the ApsaraDB for HBase dat
abase.
    password = 'root', --The password that is used to access the ApsaraDB for HBase datab
ase.
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- The following sample code demonstrates how to create an ApsaraDB for HBase dimension table of HBase Enhanced Edition in Blink 3.5.0 or later:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '<host:port>', ----The Java API URL that is used to access the Enhanced Ed
ition of an ApsaraDB for HBase database.
    userName  = 'root', --The username that is used to access the ApsaraDB for HBase data
base.
    password = 'root', --The password that is used to access the ApsaraDB for HBase datab
ase.
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- The following sample code demonstrates how to create an ApsaraDB for HBase dimension table in Blink 3.5.0 or later that supports switchover between primary and secondary ApsaraDB for HBase databases for data writing:

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '<host:port>', ---- The URL that is used to access ApsaraDB for HBase data
bases in high availability (HA) mode.
    haClusterID = 'ha-xxx', ---- The instance ID of ApsaraDB for HBase databases in HA mo
de.
    userName  = 'root', --The username that is used to access the ApsaraDB for HBase data
base.
    password = 'root', --The password that is used to access the ApsaraDB for HBase datab
ase.
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

Note

- You can define multiple fields for the primary key. Multiple fields are separated with the value of `rowkeyDelimiter`. The default value is a colon (`:`).
- When you undo the deletion operation in ApsaraDB for HBase, if a column stores multiple versions of a value, all the versions of the value are deleted.
- The connection parameters are different between ApsaraDB for HBase Standard Edition and ApsaraDB for HBase Enhanced Edition.
  - In ApsaraDB for HBase Standard Edition, the connection parameter is `zkQuorum`.
  - In ApsaraDB for HBase Enhanced Edition, the connection parameter is `endPoint`.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the result table.	Yes	Set the value to cloudhbase.
zkQuorum	The ZooKeeper address that is configured for the ApsaraDB for HBase cluster.	Yes	You can view the configuration related to hbase.zookeeper.quorum in the *hbase-site.xml* file. **Note** This parameter takes effect in only ApsaraDB for HBase Standard Edition.

Parameter	Description	Required	Remarks
zkNodeParent	The path of the cluster configured on the ZooKeeper servers.	No	You can view the relevant configurations of hbase.zookeeper.quorum in the *hbase-site.xml* file. **Note** This parameter takes effect only in ApsaraDB for HBase Standard Edition.
endPoint	The name of the region where your ApsaraDB for HBase instance is deployed.	Yes	Contact Alibaba Cloud O&M engineers.
userName	The username that is used to access an ApsaraDB for HBase database.	No	This parameter takes effect only in ApsaraDB for HBase Enhanced Edition.
password	The password that is used to access an ApsaraDB for HBase database.	No	This parameter takes effect only in ApsaraDB for HBase Enhanced Edition.
tableName	The name of the ApsaraDB for HBase table.	Yes	None.
columnFamily	The name of the column family.	Yes	Only the same column family can be inserted.
maxRetryTimes	The maximum number of retries.	No	Default value: 10. **Note** This parameter is available only in Realtime Compute for Apache Flink V3.2.3 and later.
bufferSize	The maximum number of data records that can be stored in the buffer before deduplication is triggered.	No	Default value: 5000.

Parameter	Description	Required	Remarks
batchSize	The number of data records that can be written at a time.	No	Default value: 100. **Note** We recommend that you set this parameter to a value that ranges from 200 to 300. A large value of batchSize may cause an out-of-memory (OOM) error for the task.
flushIntervalMs	The interval at which the buffer is cleared to reduce the latency of writing data into ApsaraDB for HBase.	No	Default value: 2000. Unit: milliseconds.
writePkValue	Specifies whether to write the primary key value.	No	Default value: false.
stringWriteMod	Specifies whether to insert data as the STRING type.	No	Default value: false.
rowkeyDelimiter	The delimiter of rowKey.	No	The default delimiter is a colon (:).
isDynamicTable	Specifies whether the table is a dynamic table.	No	Default value: false.
haClustserID	The ID of an ApsaraDB for HBase instance in HA mode.	No	This parameter is required only when you access zone-disaster recovery instances.

Dynamic table

Some result data of Realtime Compute for Apache Flink is used as a dynamic column based on the value of a column and written to ApsaraDB for HBase. In the following example, the turnover per hour is used as a dynamic column data in ApsaraDB for HBase.

rowkey	cf:0	cf:1	cf:2
20170707	100	cf:1	300

If isDynamicTable is set to true, the table is an ApsaraDB for HBase table that supports dynamic columns.

A dynamic table can contain only three columns, such as ROW_KEY, COLUMN, and VALUE. The second column that is COLUMN in this example is a dynamic column. Other parameters in the dynamic table are the same as those in the WITH clause of ApsaraDB for HBase.

> **Note** When a dynamic table is used, all data types must be converted to the STRING type before data is written into ApsaraDB for HBase.

```
CREATE TABLE stream_test_hotline_agent (
  name varchar,
  age varchar,
  birthday varchar,
  primary key (name)
) WITH (
  type = 'cloudhbase',
  ...
  columnFamily = 'cf',
  isDynamicTable ='true'
);
```

> **Note**

- In the preceding declaration, `birthday` is inserted into the `cf:age` column with ROW_KEY of `name`. For example, `(wang,18,2016-12-12)` is inserted into the row for which the ROW_KEY value is `wang` and the `cf:18` column.
- In the DDL statement, you must declare ROW_KEY as the primary key and declare the following fields in the descending order: ROW_KEY, COLUMN, and VALUE. In this example, ROW_KEY is `name`, COLUMN is `age`, and VALUE is `birthday`.

Sample code

The following sample code demonstrates how to create an ApsaraDB for HBase result table in a Realtime Compute for Apache Flink job:

```
create table source (
  id   TINYINT,
  name BIGINT
) with (
  type = 'random'
);
create table sink (
  id    TINYINT,
  name  BIGINT,
  primary key (id)
) with (
  type = 'cloudhbase',
  zkQuorum = '<yourZkQuorum>',
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>'
);
INSERT INTO sink
SELECT id, name FROM source;
```

FAQ

Why is the error `cloudHbase update error, No columns to insert for #10 item` reported when a job for an ApsaraDB for HBase result table is running?

The column data for a single record that is written into the ApsaraDB for HBase result table cannot all be null. The column data excludes ROW_KEY. Before you use Realtime Compute for Apache Flink to write data into the ApsaraDB for HBase result table, filter out all null data.

2.5.4. Create a data dimension table

2.5.4.1. Overview

You must add `PERIOD FOR SYSTEM_TIME` in the CREATE TABLE statement that is used to create a dimension table. `PERIOD FOR SYSTEM_TIME` defines the period of time when a data record is valid. This means that the table is frequently updated.

```
CREATE TABLE white_list (
   id varchar,
   name varchar,
   age int,
   PRIMARY KEY (id),  -- You must define a primary key for the dimension table.
   PERIOD FOR SYSTEM_TIME -- Defines the period of time when a data record is valid.
   ) with (
   type = 'rds',
   ...
   )
```

> **Note**
>
> - When you declare a dimension table, you must specify the primary key.
> - When you join a dimension table with another table, the ON condition must contain an equivalence condition that includes the primary key of either table.

2.5.4.2. Create a Tablestore dimension table

This topic describes how to create a Tablestore dimension table in Realtime Compute for Apache Flink.

> **Notice** This topic applies only to Blink 1.4.5 and later.

Introduction to Tablestore

Tablestore is a distributed NoSQL database service that is built on the Apsara distributed operating system of Alibaba Cloud. Tablestore adopts sharding and load balancing technologies to scale out services and handle concurrent transactions. You can use Tablestore to store and query large amounts of structured data in real time.

Example

In Realtime Compute for Apache Flink, you can use a Tablestore table as a dimension table. The following code shows an example:

```
CREATE TABLE ots_dim_table (
 id int,
 len int,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME--Define the change period of the dimension table.
) WITH (
 type='ots',
 endPoint='<yourEndpoint>',
 instanceName='<yourInstanceName>',
 tableName='<yourTableName>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>'
);
```

Note

- When you declare a dimension table, you must specify a primary key.
- When you join a dimension table with another table, the ON clause must contain the equivalent (=) conditions for all the primary key fields.
- The primary key of a Tablestore table is the row key of the table.

Parameters in the WITH clause

Parameter	Description	Remarks
type	The type of the dimension table.	Set the value to `ots`.
endPoint	The endpoint of the Tablestore instance.	Contact Alibaba Cloud O&M engineers.
instanceName	The name of the Tablestore instance.	None.
tableName	The name of the Tablestore dimension table.	None.
accessId	The AccessKey ID that is used to access Tablestore.	None.
accessKey	The AccessKey secret that is used to access Tablestore.	None.

Parameters in the CACHE clause

Parameter	Description	Remarks

Parameter	Description	Remarks
cache	The cache policy.	Valid values: • *None*: indicates that data is not cached. This is the default cache policy. • *LRU*: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If you use this cache policy, you must specify the cacheSize and cacheTTLMs parameters.
cacheSize	The maximum number of rows that can be cached.	You can set this parameter when the cache parameter is set to LRU. Default value: 10000.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	You can set this parameter when the cache parameter is set to LRU.

Sample code

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
type='datahub'
);
CREATE TABLE phoneNumber(
name VARCHAR,
phoneNumber bigint,
primary key(name),
PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
)with(
type='ots'
);
CREATE TABLE result_infor(
id bigint,
phoneNumber bigint,
name VARCHAR
)with(
type='rds'
);
INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --You must include this clause when
you perform a JOIN operation on the dimension table.
ON t.name = w.name;
```

For more information about the syntax for dimension tables, see JOIN statements for dimension tables.

2.5.4.3. Create an ApsaraDB for HBase dimension table

This topic describes how to create an ApsaraDB for HBase dimension table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH and CACHE clauses used when you create such a dimension table.

> **Notice**
>
> - Blink versions earlier than Blink 3.3.0 support only HBase Standard Edition.
> - Blink 3.3.0 and later versions support HBase Standard Edition and HBase Enhanced Edition.
> - Blink 3.5.0 and later versions support switchover between primary and secondary ApsaraDB for HBase databases for data writing.
> - For more information about the JOIN syntax of an ApsaraDB for HBase dimension table, see JOIN statements for dimension tables.
> - ApsaraDB for HBase dimension tables in Realtime Compute for Apache Flink do not support user-created open source HBase.
> - Only one primary key is allowed in an ApsaraDB for HBase dimension table.

DDL syntax

- HBase Standard Edition

```
CREATE TABLE hbase (
  `key` varchar,
  `name` varchar,
  PRIMARY KEY (`key`), -- The rowkey field of the ApsaraDB for HBase dimension table.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
  TYPE = 'cloudhbase',
  zkQuorum = '<yourzkQuorum>',
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
```

- HBase Enhanced Edition

```
CREATE TABLE hbase (
  `key` varchar,
  `name` varchar,
  PRIMARY KEY (`key`), -- The rowkey field of the ApsaraDB for HBase dimension table.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
  TYPE = 'cloudhbase',
  endPoint = '<host:port>',-- The Java API URL that is used to access the Enhanced Editio
n of an ApsaraDB for HBase database.
  userName  = 'root', -- The username that is used to access an ApsaraDB for HBase databa
se.
  password = 'root', -- The password that is used to access an ApsaraDB for HBase databas
e.
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
```

- HBase Enhanced Edition for Blink 3.5.0 and later

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', -- The Java API URL that is used to access the Enhanced Editi
on of an ApsaraDB for HBase database.
  userName  = 'root', -- The username that is used to access an ApsaraDB for HBase databa
se.
  password = 'root', -- The password that is used to access an ApsaraDB for HBase databas
e.
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

- Blink 3.5.0 and later versions support switchover between primary and secondary ApsaraDB for HBase databases for data writing.

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', -- The URL that is used to access ApsaraDB for HBase database
s in high availability (HA) mode.
  haClusterID = 'ha-xxx', -- The instance ID of ApsaraDB for HBase databases in HA mode.
  userName  = 'root', -- The username that is used to access an ApsaraDB for HBase databa
se.
  password = 'root', -- The password that is used to access an ApsaraDB for HBase databas
e.
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

> **Note**
>
> - When you declare a dimension table, you must specify a primary key.
> - When you join a dimension table with another dimension table, the ON condition must contain equivalent conditions that include all primary keys. The primary key in the preceding examples is **row_key**.
> - The connection parameters in HBase Standard Edition and HBase Enhanced Edition are different.
>   - HBase Standard Edition: `zkQuorum`.
>   - HBase Enhanced Edition: `endPoint`.
>   - HBase Standard Edition and HBase Enhanced Edition for Blink 3.5.0 and later: `zkQuorum`.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `cloudhbase`.
zkQuorum	The ZooKeeper address configured for the ApsaraDB for HBase cluster. The address is a list of hosts separated by commas (,).	Yes	You can view the configuration related to hbase.zookeeper.quorum in the hbase-site.xml file. **Note** This parameter takes effect only in HBase Standard Edition.
zkNodeParent	The path of the cluster configured on the ZooKeeper servers.	No	You can view the configuration related to hbase.zookeeper.quorum in the hbase-site.xml file. **Note** This parameter takes effect only in HBase Standard Edition.
endPoint	The name of the region where your ApsaraDB for HBase instance is deployed.	Yes	You can obtain the value of this parameter from the console of your ApsaraDB for HBase instance. **Note** This parameter takes effect only in HBase Enhanced Edition.
userName	The username that is used to log on to the ApsaraDB for HBase database.	No	**Note** This parameter takes effect only in HBase Enhanced Edition.

Parameter	Description	Required	Remarks
password	The password that is used to log on to the ApsaraDB for HBase database.	No	Note This parameter takes effect only in HBase Enhanced Edition.
tableName	The name of the ApsaraDB for HBase dimension table.	Yes	None.
columnFamily	The column family name.	Yes	Only the same column family can be inserted.
maxRetryTimes	The maximum number of retries for writing data into the table.	No	Default value: 10.
partitionedJoin	Specifies whether to use joinKey for partitioning.	No	Default value: False. If you set this parameter to True, joinKey is used for partitioning. Data is delivered to each node for joining, which increases the cache hit ratio.
shuffleEmptyKey	Specifies whether to randomly send upstream empty keys to downstream nodes.	No	Default value: True. Valid values: • True: If multiple empty keys exist in the upstream, Realtime Compute for Apache Flink randomly sends all the empty keys to each JOIN node. • False: If multiple empty keys exist in the upstream, Realtime Compute for Apache Flink sends all the empty keys to a single JOIN node. Note This parameter can be used only after the partitionedJoin parameter takes effect.

Parameters in the CACHE clause

Parameter	Description	Required	Remarks

Parameter	Description	Required	Remarks
cache	The policy for caching data.	No	Valid values: • *None* (default value): indicates that no data is cached. • *LRU*: indicates that partial data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • *ALL*: indicates that all data in the dimension table is cached. Before a Realtime Compute for Apache Flink job starts to run, Realtime Compute for Apache Flink loads all data in the dimension table to the cache, and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If this cache policy is used, you must configure the cacheTTLMs and cacheReloadTimeBlackList parameters. **Note** If the cache parameter is set to ALL, the memory of the node for joining tables must be increased because Realtime Compute for Apache Flink asynchronously loads data from the dimension table. The increased memory size is two times that of the remote table.
cacheSize	The maximum number of data records that can be cached. Unit: lines.	No	You can set this parameter when the cache parameter is set to `LRU`. Default value: 10000.

Parameter	Description	Required	Remarks
cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	If the cache parameter is set to `LRU`, the cacheTTLMs parameter specifies the time before cache entries expire. Cache entries do not expire by default. If the cache parameter is set to `ALL`, the cacheTTLMs parameter specifies the interval at which the cache is loaded. The cache is not reloaded by default.

cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter is used when the cache parameter is set to `ALL`. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for massive online promotional events such as Double 11.	No	This parameter is empty by default. Custom input format:
			`2017-10-24 14:00 -> 2017-10-24 15:00,`
			`2017-11-10 23:30 -> 2017-11-11 08:00`
			Separate multiple time periods with commas (,). Separate the start time and end time of each time period with a hyphen and a greater-than sign (->).

cacheScanLimit	The maximum number of rows returned by the server to the client for each remote procedure call (RPC) when the server reads full ApsaraDB for HBase data. This parameter is used when the cache parameter is set to `ALL`. .	No	Default value: 100.

Sample code

The following sample code demonstrates how to create an ApsaraDB for HBase dimension table in a Realtime Compute for Apache Flink job.

```
create table source (
  id   TINYINT,
  name BIGINT
) with (
  type = 'random'
);
create table dim (
  id   TINYINT,
  score BIGINT
  primary key(id),
  PERIOD FOR SYSTEM_TIME
)with(
  type = 'cloudhbase',
  zkQuorum = '<yourzkQuorum>',
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
CREATE table result_infor(
  id BIGINT,
  score BIGINT
)with(
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.score
FROM source as t
JOIN dim FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.id = w.id;
```

2.5.4.4. Create a KVStore for Redis dimension table

This topic describes how to create a KVStore for Redis dimension table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH and CACHE clauses, data type mapping, and sample code used when you create such a dimension table.

> **Notice**
>
> - This topic applies only to Blink 3.2.2 and later.
> - KVStore for Redis dimension tables in Realtime Compute for Apache Flink can only reference data of the STRING type in KVStore for Redis databases.
> - KVStore for Redis dimension tables in Realtime Compute for Apache Flink support user-created Redis databases.

Syntax

In Realtime Compute for Apache Flink, you can create a KVStore for Redis dimension table. The following code shows an example:

```
CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id),  -- The Row Key field in a KVStore for Redis database.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) WITH (
  type = 'redis',
  host = '<yourHostName>',
  port = '<yourPort>',
  password = '<yourPassword>',
  dbNum = '<yourDatabaseNumber>'
);
```

Note

- Only one primary key can be declared for a KVStore for Redis dimension table.
- When you join a dimension table with another dimension table, the ON condition must contain equivalent conditions of all the primary keys.
- You can declare only two fields for a KVStore for Redis dimension table, and the fields must be of the VARCHAR type.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `redis`.
host	The endpoint of a KVStore for Redis instance.	Yes	None.
port	The port of the KVStore for Redis database.	No	Default value: 6379.
dbNum	The sequence number of the KVStore for Redis database.	No	Default value: 0.
password	The password that is used to access the KVStore for Redis database.	No	This parameter is empty by default, which indicates that permission verification is not required.

Parameter	Description	Required	Remarks
hashName	The hash key name in hash mode.	No	This parameter is empty by default, which indicates that Realtime Compute for Apache Flink reads data of the STRING type from the KVStore for Redis database. In typical cases, the data type in the Redis dimension table is STRING, which is represented as `key-value` pairs. If you set the hashName parameter, data in the KVStore for Redis dimension table is of the HASHMAP type, which is presented as `key-{field-value}` pairs. • key is the value of the hashName parameter. • field is the value of the key parameter that you specify in the CREATE TABLE statement. • value is the value assigned to key, which has the same meaning as value in `key-value` of the STRING type.

Parameters in the CACHE clause

Parameter	Description	Required	Remarks
cache	The policy for caching data.	No	Valid values: • *None* (default value): indicates that no data is cached. • *LRU*: indicates that partial data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters.
cacheSize	The maximum number of data records that can be cached. Unit: lines.	No	If you set the cache parameter to `LRU`, you can set this parameter to specify the maximum number of data records that can be cached. Default value: 10000.

Parameter	Description	Required	Remarks
cacheTTLMs	The cache timeout period.	No	The cache does not expire by default. Unit: milliseconds. If the cache policy is set to LRU, this parameter specifies the time before the cache expires.
cacheEmpty	Specifies whether to clear the cache.	No	Default value: true.

Field type mapping

The following table describes the mapping between KVStore for Redis data types and data types of Realtime Compute for Apache Flink. We recommend that you declare the mapping in a DDL statement.

Data type of KVStore for Redis	Data type of Realtime Compute for Apache Flink
STRING	VARCHAR

Sample code

The following sample code demonstrates how to create a KVStore for Redis dimension table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE event (
  id VARCHAR,
  data VARCHAR) with (
  type = 'random'
);
CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id),  -- The Row Key field in a KVStore for Redis database.
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) WITH (
  type = 'redis',
  host = '<yourRedisHost>',
  password = '<yourRedisPassword>'
);
SELECT e.*, w. *
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

2.5.4.5. Create an ApsaraDB RDS for MySQL dimension table

This topic describes how to create an ApsaraDB RDS for MySQL dimension table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause, cache parameters, and data type mapping used when you create an ApsaraDB RDS for MySQL dimension table.

ApsaraDB RDS for MySQL

ApsaraDB RDS for MySQL is developed based on a branch of the MySQL source code. Its excellent performance has been proven over years of Double 11, during which it needs to handle large volumes of concurrent traffic. ApsaraDB RDS for MySQL provides basic features, such as instance management, account management, database management, backup and restoration, control access, Transparent Data Encryption (TDE), and data migration. ApsaraDB RDS for MySQL also provides the following advanced features and functions:

- ApsaraDB MyBase dedicated clusters: An ApsaraDB MyBase dedicated cluster consists of multiple hosts, such as ECS instances of the ecs.i2.xlarge instance type and ECS Bare Metal instances. You can run instances on these hosts. For more information, see What is ApsaraDB for MyBase?
- Read-only RDS instances: If the primary RDS instance is overwhelmed by a large number of read requests, your workloads may be interrupted. In this case, you can create one or more read-only RDS instances to offload read requests from the primary RDS instance. For more information, see Overview of ApsaraDB RDS for MySQL read-only instances. This scales up the read capability of your database system and increases the throughput of your application.
- Read/write splitting: The read/write splitting function provides a read/write splitting endpoint. This endpoint connects to the primary RDS instance and all of the read-only RDS instances to establish an automatic read/write splitting link. For more information, see Read/write splitting. Your application can read and write data into your database system after it connects to this endpoint. ApsaraDB for RDS distributes write requests to the primary RDS instance and read requests to the read-only RDS instances based on the specified read weights. You can create more read-only RDS instances to scale up the read capability of your database system. In addition, you do not need to modify your application.
- Dedicated proxy: A dedicated proxy uses dedicated computing resources. It provides more advanced functions, such as read/write splitting, short-lived connection optimization, and transaction splitting. For more information, see What are database proxies?.
- Database Autonomy Service (DAS): DAS supports intelligent diagnostics and optimization at the instance level based on various metrics. These metrics include SQL execution performance, CPU utilization, input/output operations per second (IOPS) utilization, memory usage, disk usage, number of connections, locks, and hotspot tables. For more information, see DAS overview. DAS allows you to identify existing and potential issues that may compromise the health of your database system. In addition, DAS provides details and solutions for the identified issues. This facilitates database maintenance.

ApsaraDB RDS for MySQL supports only two storage engines: InnoDB and X-Engine. For more information, see Features of ApsaraDB RDS for MySQL instances.

Limits

Realtime Compute for Apache Flink does not allow you to use ApsaraDB RDS for MySQL V8.0 by using the storage registration method. To use ApsaraDB RDS for MySQL V8.0, we recommend that you configure a plaintext AccessKey pair. For more information about the storage registration method, see Overview.

DDL syntax

The following sample code shows how to create an ApsaraDB RDS for MySQL dimension table:

```
CREATE TABLE rds_dim_table(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME --Define the change period of the dimension table.
) with (
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

Note You must specify a primary key when you declare a dimension table. When you join a dimension table with another table, the ON condition must contain equivalent conditions that include all primary keys. The primary key of an ApsaraDB RDS for MySQL or Distributed Relational Database Service (DRDS) database can be defined as the primary key or unique index column of an ApsaraDB RDS for MySQL or DRDS dimension table.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `rds`.
url	The Java Database Connectivity (JDBC) URL of the database.	Yes	Contact Alibaba Cloud O&M engineers to obtain the required information.
tableName	The name of the table.	Yes	N/A.
userName	The username that is used to access the ApsaraDB RDS database.	Yes	N/A.
password	The password that is used to access the ApsaraDB RDS database.	Yes	N/A.
maxRetryTimes	The maximum number of connection attempts.	No	Default value: 10.

Cache parameters

Parameter	Description	Required	Remarks
cache	The policy that is used to cache data.	No	Valid values: • *None*: indicates that data is not cached. This is the default cache policy. • *LRU*: indicates that only the specified data in the dimension table is cached. Each time the system receives a data record, the system searches the cache. If the system does not find the record in the cache, the system searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • *ALL*: indicates that all the data in the dimension table is cached. Before the system runs a job, the system loads all data in the dimension table to the cache. This way, the cache is searched for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data in a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. The source table and dimension table cannot be associated based on the ON clause. If you use this cache policy, you must configure the cacheTTLMs and cacheReloadTimeBlackList parameters. **Note** If you set the cache parameter to ALL, you must increase the memory of the node for joining tables because the system asynchronously loads data from the dimension table. The increased memory size is twice the memory size of the remote table.
cacheSize	The maximum number of rows of data records that can be cached.	No	This parameter is available only if you set the cache parameter to `LRU`. Default value: 10000.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	If the cache parameter is set to `LRU`, the cacheTTLMs parameter specifies the time allowed before cache entries expire. Cache entries do not expire by default. If the cache parameter is set to `ALL`, the cacheTTLMs parameter specifies the interval at which the cache is loaded. The cache is not reloaded by default.

Parameter	Description	Required	Remarks
cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter takes effect when the cache parameter is set to `ALL`. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	No	This parameter is empty by default. For example, you can set this parameter to `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00`. Multiple time periods are separated by commas (,). The start time and end time of each time period are separated by a hyphen and a greater-than sign (->).
maxJoinRows	The maximum number of results that are returned each time a data record in the primary table is queried and matched with data records in the dimension table.	No	Default value: 1024. If you can estimate that a data record in the primary table corresponds to a maximum of n data records in the dimension table, you can set the maxJoinRows parameter to n to ensure efficient matching in Realtime Compute for Apache Flink. **Note** When you join a dimension table with another table, this parameter specifies the maximum number of results that can be returned after a data record in the primary table is matched with data records in the dimension table.

Sample code

The following sample code shows how to create an ApsaraDB RDS dimension table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2017-07-21 00:00:00'
);
create table phoneNumber(
name VARCHAR,
phoneNumber BIGINT,
primary key(name),
PERIOD FOR SYSTEM_TIME--Define the change period of the dimension table.
)WITH(
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
CREATE table result_infor(
id BIGINT,
phoneNumber BIGINT,
name VARCHAR
)WITH(
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --You must include this clause when
you perform a JOIN operation on the dimension table.
ON t.name = w.name;
```

For more information about the detailed syntax of the JOIN statements for a dimension table, see JOIN statements for dimension tables.

Data type mapping

Data type of ApsaraDB RDS	Data type of Realtime Compute for Apache Flink
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

2.5.4.6. Create a MaxCompute dimension table

This topic describes how to create a MaxCompute dimension table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH and CACHE clauses and data type mapping used when you create a MaxCompute dimension table.

Notice

- Blink 2.1.1 and later versions support MaxCompute dimension tables.
- For more information about the query syntax of a dimension table, see JOIN statements for dimension tables.
- To use a MaxCompute dimension table, you must grant the read permissions to the account used to access MaxCompute.

DDL syntax

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) WITH (
  type = 'odps',
  endPoint = '<YourEndPoint>',
  project = '<YourProjectName>',
  tableName = '<YourtableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****',
  cache = 'ALL'
);
```

Note

- When you declare a dimension table, you must specify a primary key. The primary key of a MaxCompute dimension table must be unique. Duplicate primary keys are deleted.
- When you join a dimension table with another table, the ON condition must contain equivalence conditions that include all primary keys.
- partition is a keyword and must be commented with backticks (`), for example, `'partition'`.
- If the dimension table is a partitioned table, Realtime Compute for Apache Flink does not write partition key columns to the DDL statement.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `odps`.
endPoint	The endpoint of MaxCompute.	Yes	Contact Alibaba Cloud O&M engineers.
tunnelEndpoint	The endpoint of the MaxCompute Tunnel service.	Yes	Contact Alibaba Cloud O&M engineers.
project	The name of a MaxCompute project.	Yes	None.
tableName	The name of the table.	Yes	None.

Parameter	Description	Required	Remarks
accessId	AccessKey ID	Yes	None.
accessKey	AccessKey Secret	Yes	None.

partition	The name of a partition.	No	• Fixed partition
			  ◦ A MaxCompute table with only one partition
			  For example, if only one partition key column `ds` exists, `` `partition` = 'ds=20180905' `` indicates that the data in the `ds=20180905` partition is read.
			  ◦ A MaxCompute table with multiple partitions
			  For example, if two partition key columns `ds` and `hh` exist, `` `partition`='ds=20180905,hh=*' `` indicates that the data in the `ds=20180905` partition is read.
			  Note You must declare the values of all partitions when you filter partitions. In the preceding example, if you declare only `` `partition` = 'ds=20180905' ``, no partition is read.
			• Non-fixed partition
			  ◦ Blink 2.2.0 and later versions support `'partition' = 'max_pt()'`. This setting indicates that the partition ranked first in alphabetical order among all partitions is loaded each time the system loads data of partitions.
			  ◦ Blink 3.2.2 and later versions support `'partition' = 'max_pt_with_done()'`. This setting indicates that the partition with the file name extension `.done` and ranked first in alphabetical order among all partitions is loaded each time the system loads data of partitions.

maxRowCount	The maximum number of rows that Realtime Compute for Apache Flink can load from a table.	No	Default value: 100000. Note If your data contains more than 100,000 rows, you must set this parameter. We recommend that you set this parameter to a greater value than the actual number of rows to be loaded.

Parameters in the CACHE clause

Parameter	Description	Remarks
cache	The cache policy.	You must set the cache parameter to `ALL` for a MaxCompute dimension table and explicitly declare this setting in the DDL statement. *ALL*: indicates that all data in the dimension table is cached. Before a Realtime Compute for Apache Flink job runs, Realtime Compute for Apache Flink loads all the data in the dimension table to the cache. Then, Realtime Compute for Apache Flink searches among the cache if requests are sent to retrieve data from the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the data amount of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If you set this parameter to ALL, you must specify the following parameters: cacheTTLMs and cacheReloadTimeBlackList. **Note** • If the cache parameter is set to ALL, you must increase the memory of the join node because the system asynchronously loads data of the dimension table. We recommend that you increase the size of the memory at least four times the data amount of the remote table. The value is related to the MaxCompute storage compression algorithm. • If a job exception occurs due to frequent garbage collection when you use a super-large MaxCompute dimension table, and this issue persists even if you increase the memory of the join node, we recommend that: ◦ For Blink 3.6.0 and later versions, set the partitionedJoin parameter to true to enable partitionedJoin optimization. ◦ Use a dimension table with key-value pairs for which you can set the cache parameter to LRU, for example, an ApsaraDB for HBase dimension table.
cacheSize	The maximum number of data records that can be cached.	You can specify the cacheSize parameter based on your business requirements. By default, Realtime Compute for Apache Flink can cache 100,000 rows that are stored in a MaxCompute dimension table.

Parameter	Description	Remarks
cacheTTLMs	The cache timeout period.	Unit: milliseconds. If you set the cache parameter to `ALL`, the timeout period specifies the interval at which Realtime Compute for Apache Flink refreshes the cache. The cache is not refreshed by default.
cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter takes effect when the cache parameter is set to ALL. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	This parameter is empty by default. The following example shows the format of the values: `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00`. The following list describes the delimiters that you use for the parameter values: • Separate multiple time periods with commas (`,`). • Separate the start time and end time of each time period with a hyphen and a greater-than sign (`->`).
partitionedJoin	Specifies whether to cache full data of a dimension table in the memory of each concurrent job.	This parameter is optional. Default value: false. This value indicates that full data of the dimension table is cached in the memory of each concurrent job. If you set this parameter to true, partial data of the dimension table is cached in the memory of each concurrent task.

Sample code

The following sample code demonstrates how to create a MaxCompute dimension table in a Realtime Compute for Apache Flink job.

```
CREATE TABLE datahub_input1 (
   id    BIGINT,
   name  VARCHAR,
   age   BIGINT
) with (
   type='datahub'
);
CREATE TABLE odps_dim (
   name VARCHAR,
   phoneNumber BIGINT,
   PRIMARY KEY (name),
   PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
   type = 'odps',
   endPoint = '<yourEndpointName>',
   project = '<yourProjectName>',
   tableName = '<yourTableName>',
   accessId = '<yourAccessId>',
   accessKey = '<yourAccessPassword>',
  `partition` = 'ds=20180905',--The fixed partition. For more information about dynamic par
titions and fixed partitions, see the description of parameters in the WITH clause.
  cache = 'ALL'
);
CREATE table result_infor(
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
)with(
  type='print'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN odps_dim FOR SYSTEM_TIME AS OF PROCTIME() as w --You must include this clause when you
perform a JOIN operation on a dimension table.
ON t.name = w.name;
```

Field type mapping

The following table lists the mapping between the data types of MaxCompute and Realtime Compute for Apache Flink (fully-managed Flink). We recommend that you declare the mapping in a DDL statement.

MaxCompute	BLINK
TINYINT	TINYINT
SMALLINT	SMALLINT

MaxCompute	BLINK
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

Note Realtime Compute for Apache Flink supports only the preceding data types in MaxCompute dimension tables.

2.5.4.7. Create an AnalyticDB for MySQL V3.0 dimension table

This topic describes how to create an AnalyticDB for MySQL V3.0 dimension table. This topic also describes the parameters in the WITH and CACHE clauses used when you create an AnalyticDB for MySQL V3.0 dimension table.

Notice This topic applies only to Blink-3.5.0-hotfix and later.

Syntax

```
CREATE TABLE dim_ads(
    `name` VARCHAR,
    id VARCHAR,
    PRIMARY KEY (`name`),
    PERIOD FOR SYSTEM_TIME
)with(
    type='ADB30',
    url='jdbc:mysql://<Internal endpoint>/<databaseName>',
    tableName='xxx',
    userName='xxx',
    password='xxx'
);
```

Note

- You must specify a primary key when you declare a dimension table.
- When you join a dimension table with another table, the ON clause must contain the equivalent (=) conditions for all the primary key fields.
- You can define a primary key of AnalyticDB for MySQL as the primary key or the unique index column of the dimension table.

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to ADB30.
url	The URL of the AnalyticDB for MySQL database.	Yes	Contact Alibaba Cloud O&M engineers to obtain the required information.
tableName	The name of the table.	Yes	N/A.
userName	The username that is used to access the AnalyticDB for MySQL database.	Yes	N/A.
password	The password that is used to access the AnalyticDB for MySQL database.	Yes	N/A.

Parameter	Description	Required	Remarks
maxRetryTimes	The maximum number of retries for writing data to the table.	No	Default value: 3.

Parameters in the CACHE clause

Parameter	Description	Remarks
cache	The cache policy.	AnalyticDB for MySQL V3.0 supports the following cache policies: • *None*: indicates that no data is cached. This is the default value. • *LRU*: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record from the source table. If the system does not find the record in the cache, the system searches for the data record in the physical dimension table. If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters. • *ALL*: indicates that all the data in the dimension table is cached. Before Realtime Compute for Apache Flink runs a job, Realtime Compute for Apache Flink loads all the data in the dimension table to the cache and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the data amount of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters.

Parameter	Description	Remarks
		Note • If you set the cache parameter to ALL, you must increase the memory space of the node where the dimension table is joined with another table. This is because Realtime Compute for Apache Flink asynchronously loads the data from the dimension table. The increased memory size is twice the size of the data in the remote table. • If a dimension table stores a large volume of data and the cache parameter is set to ALL, an out of memory (OOM) error may occur or a full garbage collection (GC) may be time-consuming. To address this issue, you can use the following methods: ◦ If the cache parameter can be set to ALL for a dimension table, enable the partitionedJoin feature For a Blink version earlier than Blink 3.6.0, the full data of the dimension table is loaded for each concurrent job by default. For a Blink version later than Blink 3.6.0, the partitionedJoin feature is available if you set the cache parameter to ALL. After you enable the partitionedJoin feature, only the required data is cached for each concurrent job. ◦ Use an ApsaraDB for HBase or ApsaraDB RDS dimension table that uses key-value pairs to store data.
cacheSize	The cache size.	This parameter is available only if you set the cache parameter to *LRU*. Default value: 10000. Unit: rows.
cacheTTLMs	The interval at which Realtime Compute for Apache Flink refreshes the cache. The system reloads the latest data in the dimension table based on the value of this parameter. This ensures that the data in the source table is associated with the latest data in the dimension table.	Unit: milliseconds. This parameter is empty by default. This indicates that the updates in the dimension table are not reloaded.
cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter takes effect when the cache parameter is set to ALL. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	This parameter is optional. It is empty by default. For example, you can specify this parameter as '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'. Use the following delimiters to separate time periods: • Separate multiple time periods with commas (,). • Separate the start time and end time of each time period with a hyphen and a greater-than sign (->).

Parameter	Description	Remarks
partitionedJoin	Specifies whether to enable the partitionedJoin feature. If the partitionedJoin feature is enabled, shuffling is implemented based on JOIN keys before the source table is joined with the dimension table. This process provides the following benefits: • If you set the cache parameter to *LRU*, the cache hit rate increases. • If you set the cache parameter to *ALL*, memory resources are reduced because only the required data is cached for each concurrent job.	The default value of this parameter is false. This indicates that the partitionedJoin feature is disabled. **Note** Before you enable the partitionedJoin feature, specify the following setting: partitionedJoin = 'true'.

Sample code

```
CREATE TABLE datahub_input1 (
  id      BIGINT,
  name    VARCHAR,
  age     BIGINT
) WITH (
  type='datahub'
);
create table phoneNumber (
  name VARCHAR,
  phoneNumber BIGINT,
  primary key(name),
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
  type='ADB30'
);
CREATE table result_infor (
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
) with (
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w -- You must include this clause in t
he INSERT INTO statement if you are performing a JOIN operation on the dimension table.
ON t.name = w.name;
```

2.5.4.8. Create a Hologres dimension table

This topic describes how to create a Hologres dimension table. It also describes the parameters in the WITH clause, parameters in the CACHE clause, and data type mapping used when you create such a dimension table.

> **Notice**
>
> - This topic applies only to Blink 3.6.0 and later.
> - We recommend that you use Hologres 0.7 or later.

Introduction to Hologres

Hologres is compatible with the PostgreSQL protocol and closely connected to the big data ecosystem. Hologres supports real-time analysis and processing of petabytes of data with high concurrency and low latency. This allows you to use existing Business Intelligence (BI) tools to easily perform multidimensional analysis and business exploration.

Syntax

In Realtime Compute for Apache Flink, you can use a Hologres table as a dimension table. The following code shows an example:

```
CREATE TABLE rds_dim_table(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME  -- The identifier of a dimension table.
) with (
  type='hologres',
  endpoint='...',
  dbname='...',
  tablename='...',
  username='...',
  password='...'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the database.	Yes	Set the value to *hologres*.
endpoint	The endpoint of Hologres.	Yes	None.
tablename	The table from which data is read.	Yes	None.
dbname	The database from which data is read.	Yes	None.
username	The username that is used to log on to the database.	Yes	None.
password	The password that is used to log on to the database.	Yes	None.

Parameters in the CACHE clause

Parameter	Description	Required	Example
			Hologres dimension tables support the following cache policies: • *None* (default value): indicates that no data is cached. • *LRU*: indicates that partial data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system

Parameter	Description	Required	Example
cache	The policy for caching data.	No	does not find the record in the cache, it searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • *ALL*: indicates that all data in the dimension table is cached. Before a Realtime Compute for Apache Flink job starts to run, Realtime Compute for Apache Flink loads all data in the dimension table to the cache, and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the amount of data of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If this cache policy is used, you must configure the cacheTTLMs and cacheReloadTimeBlackList parameters.

Parameter	Description	Required	Example
			Note • If the cache parameter is set to ALL, the memory of the node for joining tables must be increased because Realtime Compute for Apache Flink asynchronously loads data from the dimension table. The increased memory size is two times that of the remote table. • If the amount of data in a dimension table is large and the cache parameter is set to ALL, an out of memory (OOM) error may occur or a full garbage collection may be time-consuming. To address this issue, you can use one of the following methods: ◦ If the cache parameter can be set to ALL for a dimension table, enable the partitionedJoin function. For Blink versions earlier than 3.6.0, full data of the dimension table is loaded in each concurrent job by default. For Blink versions later than 3.6.0, the partitionedJoin function is supported when you set the cache parameter to ALL. After you enable the partitionedJoin function, only the required data is cached for each concurrent job. ◦ Use an HBase or ApsaraDB RDS dimension table that uses key-value pairs to store data.
cacheSize	The maximum number of data records that can be cached. Unit: lines.	No	You can specify this parameter only when you set the cache parameter to *LRU*. Default value: *10000*.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	• You can specify this parameter only when you set the cache parameter to *LRU*. The cache will not expire by default. • If the cache parameter is set to *ALL*, the cacheTTLMs parameter specifies the interval at which the cache is reloaded. The cache is not reloaded by default.
partitionedJoin	Specifies whether to search for data by partition.	No	Valid values: • *false* (default value) • *true*
async	Specifies whether to read data asynchronously.	No	Valid values: • *false* (default value) • *true*

Field type mapping

Hologres data type	Data type of Realtime Compute for Apache Flink
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

Sample code

The following sample code demonstrates how to create a Hologres dimension table in a Realtime Compute for Apache Flink job:

```
create table randomSource (a int, b VARCHAR, c VARCHAR) with (type = 'random');
create table test (
   a int,
   b VARCHAR,
   c VARCHAR,
   PRIMARY KEY (a, b), PERIOD FOR SYSTEM_TIME
) with (
   type = 'hologres',
   ...
);
create table print_sink (
  a int,
  b VARCHAR
) with (
  type = 'print',
  `ignoreWrite` = 'false'
);
insert into print_sink
select randomSource.a, test.b from randomSource
LEFT JOIN test FOR SYSTEM_TIME AS OF PROCTIME()
on randomSource.a = test.a and randomSource.b = test.b;
```

2.5.4.9. Create a Phoenix5 dimension table

This topic describes how to create a Phoenix5 dimension table in Realtime Compute for Apache Flink. It also describes the parameters in the WITH and CACHE clauses used when you create a Phoenix5 dimension table.

> **Notice** Only Blink versions later than Blink 3.4.0 support Phoenix5 dimension tables.

Syntax

```
create table US_POPULATION_DIM (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<YourServerUrl>',
  tableName = '<YourTableName>'
);
```

Parameters in the WITH clause

Parameter	Description	Required	Remarks
type	The type of the dimension table.	Yes	Set the value to `PHOENIX5`.

Parameter	Description	Required	Remarks
serverUrl	The URL of the Phoenix5 query server. • If Phoenix5 is created in a cluster, the value of this parameter is the URL of Server Load Balancer (SLB). • If Phoenix5 is created on a single server, the value of this parameter is the URL of the server.	Yes	You must enable the HBase SQL service in an ApsaraDB for HBase instance. The value of the serverUrl parameter is in the http://host:port format. In the format: • host: indicates the domain name of the Phoenix5 service. • port: indicates the port number of the Phoenix5 service. Set the value to 8765.
tableName	The name of the Phoenix5 table.	Yes	The name of the Phoenix5 table is in the SchemaName.TableName format. In the format: • SchemaName: indicates the schema name, which can be empty. This means that the schema name is not used and only the table name is used. In this case, the default schema of the database is used. • TableName: indicates the name of the table.

Parameters in the CACHE clause

Parameter	Description	Required	Remarks

Parameter	Description	Required	Remarks
cache	The cache policy.	No	Valid values: • *None*: indicates that no data is cached. This is the default value. • *LRU*: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record. If the system does not find the record in the cache, it searches for the data record in the physical dimension table. If this cache policy is used, you must configure the cacheSize and cacheTTLMs parameters. • *ALL*: indicates that all the data in the dimension table is cached. Before Realtime Compute for Apache Flink runs a job, Realtime Compute for Apache Flink loads all the data in the dimension table to the cache and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the join key does not exist. The system reloads all data in the cache after cache entries expire. If the data amount of a remote table is small and a large number of missing keys exist, we recommend that you set this parameter to ALL. (The source table and dimension table cannot be associated based on the ON clause.) If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters. **Note** If you set the cache parameter to ALL, you must increase the memory of the node for joining tables because Realtime Compute for Apache Flink asynchronously loads data from the dimension table. The increased memory size is twice that of the remote table.
cacheSize	The maximum number of data records that can be cached.	No	This parameter is available only if you set the cache parameter to *LRU*. Default value: 10000. Unit: rows.
cacheTTLMs	The cache timeout period. Unit: milliseconds.	No	If the cache parameter is set to *LRU*, the cacheTTLMs parameter specifies the time allowed before cache entries expire. Cache entries do not expire by default. If the cache parameter is set to *ALL*, the cacheTTLMs parameter specifies the interval at which the cache is loaded. The cache is not reloaded by default.

Parameter	Description	Required	Remarks
cacheReloadTimeBlackList	The time periods during which the cache is not refreshed. This parameter takes effect when the cache parameter is set to ALL. The cache is not refreshed during the time periods that you specify for this parameter. This parameter is useful for large-scale online promotional events such as Double 11.	No	This parameter is optional. It is empty by default. For example, you can set this parameter to '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'. Multiple time periods are separated by commas (,). The start time and end time of each time period are separated with a hyphen and a greater-than sign (->).

Sample code

```
CREATE TABLE datahub_input1 (
  id  BIGINT,
  name  VARCHAR,
  age   BIGINT
) WITH (
  type='datahub'
);
create table phoneNumber(
  name VARCHAR,
  phoneNumber BIGINT,
  primary key(name),
  PERIOD FOR SYSTEM_TIME--Define the change period of the dimension table.
)with(
  type='PHOENIX5'
);
CREATE table result_infor(
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
)with(
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w -- You must include this clause in t
he INSERT INTO statement if you are performing a JOIN operation on the dimension table.
ON t.name = w.name;
```

2.5.4.10. Create an Elasticsearch dimension table

This topic describes how to create an Elasticsearch dimension table in Realtime Compute for Apache Flink. This topic also describes the parameters in the WITH clause and CACHE clauses used when you create an Elasticsearch dimension table.

> **Notice** This topic applies only to Blink 3.2.2 and later.

DDL syntax

In Realtime Compute for Apache Flink, you can use an Elasticsearch table as a dimension table. The following code shows an example:

```
 CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type ='elasticsearch',
  endPoint = '<yourEndPoint>',
  accessId = '<yourUsername>',
  accessKey = '<yourPassword>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

> **Note** An Elasticsearch dimension table supports data updates based on the primary key of an Elasticsearch cluster. You can specify only one field for the primary key field.

Parameters in the WITH clause

Parameter	Description	Default value	Required
type	The type of the dimension table.	elasticsearch	Yes
endPoint	The endpoint of the Elasticsearch cluster, for example, *http://127.0.0.1:9211*.	No default value	Yes
accessId	The AccessKey ID that is used to access the Elasticsearch cluster.	No default value	Yes
accessKey	The AccessKey secret that is used to access the Elasticsearch cluster.	No default value	Yes
index	The index name, which is similar to the database name.	No default value	Yes

Parameter	Description	Default value	Required
typeName	The type name, which is similar to the database table name.	No default value	Yes
maxRetryTimes	The maximum number of retries for writing data to the table.	30	No
timeout	The read timeout period. Unit: milliseconds.	600000	No
discovery	Specifies whether node discovery is enabled. If this feature is enabled, the client refreshes the server list every 5 minutes.	false	No
compression	Specifies whether to compress request bodies in the GZIP format.	true	No
multiThread	Specifies whether to enable multithreading for JestClient.	true	No

Parameters in the CACHE clause

Parameter	Description	Remarks
cache	The cache policy.	• *None*: indicates that no data is cached. This value is the default value. • *LRU*: indicates that only the specified data in the dimension table is cached. The system searches the cache each time it receives a data record from the source table. If the system does not find the record in the cache, the system searches for the data record in the physical dimension table. If you use this cache policy, you must configure the cacheSize and cacheTTLMs parameters. • *ALL*: indicates that all the data in the dimension table is cached. Before Realtime Compute for Apache Flink runs a job, Realtime Compute for Apache Flink loads all the data in the dimension table to the cache and then searches the cache for all subsequent queries in the dimension table. If the system does not find the data record in the cache, the key does not exist. The system reloads all data in the cache after cache entries expire.
cacheSize	The cache size.	You can specify this parameter only if you set the cache parameter to `LRU`. Default value: 10000.

Parameter	Description	Remarks
cacheTTLMs	The interval at which the cache is refreshed.	The cache does not time out by default. The purpose of setting this parameter varies based on the cache policy. • If the cache parameter is set to LRU, the cacheTTLMs parameter specifies the timeout period of the cache. • If the cache parameter is set to ALL, the cacheTTLMs parameter specifies the interval at which the cache is refreshed. The cache is not refreshed by default.

2.6. DML statements

2.6.1. INSERT INTO statement

This topic describes how to use the INSERT INTO statement in Realtime Compute.

Syntax

```
INSERT INTO tableName
        [ (columnName[ , columnName]*) ]
            queryStatement;
```

Example

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
INSERT INTO Orders(z, v)
SELECT c,d FROM OO;
```

- A Realtime Compute job created from an SQL file can include multiple DML operations. It also supports multiple data stores for input and output data, and supports multiple dimension tables. For example, a job can contain two paragraphs of SQL statements written for different businesses. You can use SQL statements to write output data into different data stores.
- In Realtime Compute, you cannot use separate SELECT statements. SELECT statements must be used after CREATE VIEW or INSERT INTO statements.
- You can update a table by using the INSERT INTO statement. For example, you can use an INSERT INTO statement to write a data record with a key value into an RDS table. If the key value is found in the table, the existing data record in the table is overwritten. If the key value is not found, the data record is added to the table.

Table types and statements

Table type	Statement

Table type	Statement
Source table	You can use the FROM clause. The INSERT INTO statement is not supported.
Dimension table	You can use the JOIN statement. The INSERT INTO statement is not supported.
Result table	You can only use the INSERT INTO statement.
View	You can only use the FROM clause.

2.7. Query statements

2.7.1. SELECT statements

You can execute SELECT statements to retrieve data from tables.

Syntax

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

Test data

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

Simple queries

- Test statement

  ```
  SELECT * FROM <Table name>;
  ```

- Test result

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14

a (VARCHAR)	b (INT)	c (DATE)
a1	46	2016-04-05

Rename objects

- Test statement

```
SELECT a, c AS d FROM <Table name>;
```

- Test result

a (VARCHAR)	d (DATE)
a1	1990-02-20
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

Deduplication queries

- Test statement

```
SELECT DISTINCT a FROM Table name;
```

- Test result

a (VARCHAR)
a1
b1
c1

Subqueries

In most cases, SELECT statements read data from tables, for example, `SELECT column_1, column_2 … FROM table_name`. SELECT statements can also read data from the results of other SELECT statements. This is known as subqueries.

> **Note** You must specify aliases in subqueries.

- Test statement

```
INSERT INTO result_table
SELECT * FROM
               (SELECT   t.a,
                         sum(t.b) AS sum_b
                FROM     t1 t
                GROUP BY t.a
               ) t1
WHERE  t1.sum_b > 100;
```

- Test result

a (VARCHAR)	b (INT)
a1	211
b1	120
a1	257

> **Note** The preceding test result is a debugging result. In the result, you can view the computing process. If your job is published and the result table is stored in DataHub, Alibaba Cloud Message Queue for Apache Kafka, or Alibaba Cloud Message Queue, the computing process is displayed. If your job is published and the result table is stored in a relational database such as ApsaraDB RDS, the records that have the same primary key values are combined into one record.

2.7.2. WHERE

A WHERE clause filters data returned by a SELECT statement.

Syntax

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ];
```

The following table describes the operators that can be used in a WHERE clause.

Operator	Description
=	Equal to
<>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than

Operator	Description
<=	Less than or equal to

Example

- Test data

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	Shanghai

- Test statements

```
SELECT * FROM XXXX WHERE City='Beijing';
```

- Test results

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

2.7.3. HAVING

When you use an aggregate function, you must add a HAVING clause to achieve the same filter effect as a WHERE clause.

Syntax

```
SELECT [ ALL | DISTINCT ]{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ];
```

Example

- Test data

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700

Customer	OrderPrice
Bush	300
Adams	2000
Carter	100

- Test statements

```
SELECT Customer,SUM(OrderPrice) FROM XXX
GROUP BY Customer
HAVING SUM(OrderPrice)<2000;
```

- Test results

Customer	SUM (OrderPrice)
Carter	1700

2.7.4. GROUP BY

A GROUP BY clause groups a result set by one or more columns.

Syntax

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ GROUP BY { groupItem [, groupItem ]* } ];
```

Example

- Test data (Table T1)

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- Test statements

```
SELECT Customer,SUM(OrderPrice) FROM T1
GROUP BY Customer;
```

- Test results

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

2.7.5. JOIN statements

JOIN statements in Realtime Compute for Apache Flink have the same semantic meanings as those for batch processing. The two types of JOIN statements allow you to join two tables. The difference is that each JOIN statement in Realtime Compute for Apache Flink joins two dynamic tables. The join results are dynamically updated to ensure that the final results are the same as the corresponding results of batch processing.

Syntax

```
tableReference [, tableReference ]* | tableexpression
[ LEFT ] JOIN tableexpression [ joinCondition ];
```

- tableReference: specifies the table name.
- tableexpression: specifies the expression.
- joinCondition: specifies the join condition.

> **Notice**
> - Only EQUI JOIN operations are supported.
> - Only INNER JOIN and LEFT OUTER JOIN operations are supported.

Example 1: Join the Orders table and the Products table

- Test data

Orders

rowtime	productId	orderId	units
`10:17:00`	30	5	4
`10:17:05`	10	6	1
`10:18:05`	20	7	2
`10:18:07`	30	8	20

rowtime	productId	orderId	units
11:02:00	10	9	6
11:04:00	10	10	1
11:09:30	40	11	12
11:24:11	10	12	4

Products

productId	name	unitPrice
30	Cheese	17
10	Beer	0.25
20	Wine	6
30	Cheese	17
10	Beer	0.25
10	Beer	0.25
40	Bread	100
10	Beer	0.25

- Test statement

```
SELECT o.rowtime, o.productId, o.orderId, o.units,p.name, p.unitPrice
FROM Orders AS o
JOIN Products AS p
ON o.productId = p.productId;
```

- Test result

o.rowtime	o.productId	o.orderId	o.units	p.name	p.unitPrice
10:17:00	30	5	4	Cheese	17
10:17:05	10	6	1	Beer	0.25
10:18:05	20	7	2	Wine	6
10:18:07	30	8	20	Cheese	17
11:02:00	10	9	6	Beer	0.25
11:04:00	10	10	1	Beer	0.25

o.rowtime	o.productId	o.orderId	o.units	p.name	p.unitPrice
11:09:30	40	11	12	Bread	100
11:24:11	10	12	4	Beer	0.25

Example 2: Join the datahub_stream1 table and the datahub_stream2 table

- Test data

datahub_stream1

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21

datahub_stream2

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21
0	10	test31
1	10	test41

- Test statement

```
SELECT s1.c,s2.c
FROM datahub_stream1 AS s1
JOIN datahub_stream2 AS s2
ON s1.a =s2.a
WHERE s1.a = 0;
```

- Test result

s1.c (VARCHAR)	s2.c (VARCHAR)
test11	test11
test11	test31

2.7.6. JOIN statements for dimension tables

In Realtime Compute for Apache Flink, each data stream can be associated with a dimension table that is stored in an external data source. This allows you to perform associated queries in Realtime Compute for Apache Flink.

> **Note** A dimension table constantly changes. Therefore, when you associate a data stream with a dimension table, you must specify the time of the dimension table snapshot with which the data stream is associated. A data stream can be associated only with the dimension table snapshot that is taken at the current time. In the future, Realtime Compute for Apache Flink will allow you to associate data streams with dimension table snapshots that are taken at different points in time. The points in time are specified by the rowtime field in the left table.

Syntax

```
SELECT column-names
FROM table1  [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF PROCTIME() [AS <alias2>]
ON table1.column-name1 = table2.key-name1;
```

The following example shows an event stream that is joined with a whitelist dimension table.

```
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

> **Note**

- Dimension tables support `INNER JOIN` and `LEFT JOIN` operations, and do not support `RIGHT JOIN` or `FULL JOIN` operations.
- You must append `FOR SYSTEM_TIME AS OF PROCTIME()` to the end of the dimension table. This way, each data record in the dimension table that can be viewed at the current time is associated with the source data.
- The subsequent input data in the source table is associated only with the latest records that are stored in the dimension table at the current time. This means that the JOIN operation is performed only at the processing time. Therefore, if the data in the dimension table is added, updated, or deleted after the JOIN operation is performed, the associated data remains unchanged.
- The ON clause must contain equivalent (=) conditions for all primary key fields of the dimension table. The primary key fields in the conditions must be the same as those in the physical tables that are referenced in the SQL statement. The ON clause can also contain other equivalent (=) conditions.
- If you want to perform one-to-many table joins, you must specify the join keys in the data definition language (DDL) INDEX syntax for dimension tables..
- Two dimension tables cannot be joined.
- In the join conditions that are specified in the ON clause, the fields in the dimension table cannot use type conversion functions, such as CAST. If you need to convert data types, perform the conversion on the fields in the source table.

Example

- Test data

datahub_input1

id (BIGINT)	name (VARCHAR)	age (BIGINT)
1	lilei	22
2	hanmeimei	20
3	libai	28

phoneNumber

name (VARCHAR)	phoneNumber (BIGINT)
dufu	13900001111
baijuyi	13900002222
libai	13900003333
lilei	13900004444

- Test statements

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
type='datahub'
);
create table phoneNumber(
name VARCHAR,
phoneNumber bigint,
primary key(name),
PERIOD FOR SYSTEM_TIME
)with(
type='rds'
);
CREATE table result_infor(
id bigint,
phoneNumber bigint,
name VARCHAR
)with(
type='rds'
);
INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.name = w.name;
```

- Test results

id (BIGINT)	phoneNumber (BIGINT)	name (VARCHAR)
1	13900004444	lilei
3	13900003333	libai

2.7.7. UNION ALL

A UNION ALL clause is used to combine two data streams. The field types and sequences of the two data streams must be the same.

Syntax

```
select_statement
UNION ALL
select_statement;
```

> **Note** Realtime Compute for Apache Flink also supports the `UNION` function. `UNION ALL` allows duplicate values and `UNION` does not allow duplicate values. In Realtime Compute for Apache Flink, `UNION` is equivalent to the combination of `UNION ALL and DISTINCT`. We recommend that you do not use `UNION` because its operating efficiency is low.

Example

- Test data

test_source_union1

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10

test_source_union2

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10
test2	2	20

test_source_union3

a (VARCHAR)	b (BIGINT)	c (BIGINT)
test1	1	10
test2	2	20
test1	1	10

- Sample code

```
SELECT
    a,
    sum(b) as d,
    sum(c) as e
FROM
    (SELECT * from test_source_union1
    UNION ALL
    SELECT * from test_source_union2
    UNION ALL
    SELECT * from test_source_union3
    )t
 GROUP BY a;
```

- Test results

a (VARCHAR)	d (BIGINT)	e (BIGINT)
test1	1	10
test2	2	20
test1	2	20
test1	3	30
test2	4	40
test1	4	40

Note The preceding test results are debugging results. In these results, you can view the computing process. If your job is published and the result table is stored in DataHub, Alibaba Cloud Message Queue for Apache Kafka, or Alibaba Cloud Message Queue, the result data contains data about the computing process. If your job is published and the result table is stored in a relational database such as ApsaraDB RDS, the records that have the same primary key values are combined into one record.

2.7.8. TopN

A TopN clause is used to compute the largest or smallest N data records of a metric in real-time data. Flink SQL uses an OVER window function to flexibly implement TopN computing.

Syntax

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
    ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
  FROM table_name)
WHERE rownum <= N [AND conditions]
```

Note

- `ROW_NUMBER()`: specifies an `OVER` window function to compute the row number. The value starts from 1.
- `PARTITION BY col1[, col2..]`: specifies one or more partitioning columns. This parameter is optional.
- `ORDER BY col1 [asc|desc][, col2 [asc|desc]...]`: specifies the columns based on which you want to sort data and the sorting order of each column.

As shown in the preceding syntax, a TopN clause requires two levels of queries.

- In the subquery, the `ROW_NUMBER()` window function is used to sort data based on the specified columns and mark the data with rankings.

- In the outer query, only the first N data records in the ranking list are obtained. For example, if N is set to 10, the first 10 data records are obtained.

During the execution, Flink SQL sorts an input data stream based on the sort key. If the first N data records in a partition are changed, the updated data is sent downstream as an update stream.

> **Note** Therefore, if you want to export the TopN data to external storage, the target result table must contain a primary key.

WHERE

To enable Flink SQL to identify a TopN query, use `rownum <= N` in the outer query to specify the first N records. Do not place `rownum` in an expression, for example, `rownum - 5 <= N`. If you specify multiple conditions in the WHERE clause, you must use `AND` to join the conditions.

Example 1

You can use the following example to return the top 100 keywords that are queried the most in each city within a specific hour. The hour, city, and ranking columns in the output table identify a unique record. Therefore, you must declare the three columns as a composite key. The key must also be configured in the external storage.

```
CREATE TABLE rds_output (
  rownum BIGINT,
  start_time BIGINT,
  city VARCHAR,
  keyword VARCHAR,
  pv BIGINT,
  PRIMARY KEY (rownum, start_time, city)
) WITH (
  type = 'rds',
  ...
)
INSERT INTO rds_output
SELECT rownum, start_time, city, keyword, pv
FROM (
  SELECT *,
     ROW_NUMBER() OVER (PARTITION BY start_time, city ORDER BY pv desc) AS rownum
  FROM (
        SELECT SUBSTRING(time_str,1,12) AS start_time,
            keyword,
            count(1) AS pv,
            city
        FROM tmp_search
        GROUP BY SUBSTRING(time_str,1,12), keyword, city
    ) a
) t
WHERE rownum <= 100
```

Example 2

- Test data

ip (VARCHAR)	time (VARCHAR)
192.168.1.1	100,000,000
192.168.1.2	100,000,000
192.168.1.2	100,000,000
192.168.1.3	100,030,000
192.168.1.3	100,000,000
192.168.1.3	100,000,000

- Test statements

```
CREATE TABLE source_table (
  IP VARCHAR,
  `TIME` VARCHAR
)WITH(
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>'
);
CREATE TABLE result_table (
  rownum BIGINT,
  start_time VARCHAR,
  IP VARCHAR,
  cc BIGINT,
  PRIMARY KEY (start_time, IP)
) WITH (
  type = 'rds',
  url='<yourDatabaseAddress>',
  tableName='blink_rds_test',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO result_table
SELECT rownum,start_time,IP,cc
FROM (
  SELECT *,
     ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY cc DESC) AS rownum
  FROM (
        SELECT SUBSTRING(`TIME`,1,2) AS start_time,-- You can specify a value based on th
e actual time. The data specified here is an example.
        COUNT(IP) AS cc,
        IP
        FROM  source_table
        GROUP BY SUBSTRING(`TIME`,1,2), IP
    )a
) t
WHERE rownum <= 3 -- You can specify a value based on the number of data records you want
to obtain. The data specified here is an example.
```

- Test results

rownum (BIGINT)	start_time (VARCHAR)	ip (VARCHAR)	cc (BIGINT)
1	10	`192.168.1.3`	3
2	10	`192.168.1.2`	2
3	10	`192.168.1.1`	1

No ranking number optimization

- You can use no ranking number optimization to solve the data bloat issue.
  - Data bloat

    Based on the TopN syntax, the `rownum` field is written into a result table as one of its primary keys. This may lead to data bloat. For example, if the ranking of a data record rises from the ninth to the first after a data update, the rankings of the records from the first to the ninth all change. The changes must be updated in the result table. This results in data bloat. The data update in the result table may slow down because an excessive volume of data is received.
  - Method of no ranking number optimization

    To avoid data bloat, remove `rownum` from the result table and compute `rownum` at the front end. The volume of TopN data records is not large, so the top 100 data records can be obtained quickly at the front end. In this case, if the ranking of a data record rises from the ninth to the first after an update, only this record needs to be delivered to the result table. This accelerates data update in the result table.
- Syntax

```
SELECT col1, col2, col3
FROM (
 SELECT col1, col2, col3
   ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
   ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
 FROM table_name)
WHERE rownum <= N [AND conditions]
```

  The syntax is similar to the original TopN syntax. You only need to remove the rownum field from the outer query.

  > **Note** If the rownum field is removed, pay attention to the definition of the primary keys in the result table. If the definition is incorrect, the TopN query result is incorrect. The primary keys must be defined in the key list at the GROUP BY node before the TopN clause.
- Example

  This example is a case from a customer in the video industry. Heavy traffic is generated each time a video is distributed. You can identify the most popular videos based on the video traffic. The following example identifies the top 5 videos that generate the most traffic per minute.
  - Test statements

```
-- Read the original data storage table from Log Service.
CREATE TABLE sls_cdnlog_stream (
vid VARCHAR, -- The video ID.
rowtime TIMESTAMP, -- The time when the video was watched.
response_size BIGINT, -- The traffic generated for watching the video.
WATERMARK FOR rowtime as withOffset(rowtime, 0)
) WITH (
type='sls',
...
);
-- Compute the consumed bandwidth by video ID in a 1-minute window.
CREATE VIEW cdnvid_group_view AS
SELECT vid,
TUMBLE_START(rowtime, INTERVAL '1' MINUTE) AS start_time,
SUM(response_size) AS rss
FROM sls_cdnlog_stream
GROUP BY vid, TUMBLE(rowtime, INTERVAL '1' MINUTE);
-- Create a result table.
CREATE TABLE hbase_out_cdnvidtoplog (
vid VARCHAR,
rss BIGINT,
start_time VARCHAR,
   -- Do not store the rownum field in the result table.
   -- Pay attention to the definition of the primary keys. The primary keys must be def
ined in the key list at the GROUP BY node before the TopN clause.
PRIMARY KEY(start_time, vid)
) WITH (
type='RDS',
...
);
-- Identify and export the IDs of the top 5 videos that generate the most traffic per m
inute.
INSERT INTO hbase_out_cdnvidtoplog
-- Do not include the rownum field in the outer query.
SELECT vid, rss, start_time FROM
(
SELECT
vid, start_time, rss,
ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY rss DESC) as rownum
FROM
cdnvid_group_view
)
WHERE rownum <= 5;
```

- Test data

vid (VARCHAR)	rowtime (TIMESTAMP)	response_size (BIGINT)
10000	2017-12-18 15:00:10	2000
10000	2017-12-18 15:00:15	4000
10000	2017-12-18 15:00:20	3000
10001	2017-12-18 15:00:20	3000
10002	2017-12-18 15:00:20	4000
10003	2017-12-18 15:00:20	1000
10004	2017-12-18 15:00:30	1000
10005	2017-12-18 15:00:30	5000
10006	2017-12-18 15:00:40	6000
10007	2017-12-18 15:00:50	8000

- Test results

start_time (VARCHAR)	vid (VARCHAR)	rss (BIGINT)
2017-12-18 15:00:00	10000	9000
2017-12-18 15:00:00	10007	8000
2017-12-18 15:00:00	10006	6000
2017-12-18 15:00:00	10005	5000
2017-12-18 15:00:00	10002	4000

2.7.9. CEP statements

MATCH_RECOGNIZE is a complex event processing (CEP) statement that identifies events from input data streams based on the specified rules and generates output events based on the specified method.

Syntax

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[MATCH_RECOGNIZE (
[PARTITION BY {partitionItem [, partitionItem]*}]
[ORDER BY {orderItem [, orderItem]*}]
[MEASURES {measureItem AS col [, measureItem AS col]*}]
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATC
H WITH TIMEOUT ROWS]
[AFTER MATCH SKIP]
PATTERN (patternVariable[quantifier] [ patternVariable[quantifier]]*) WITHIN intervalExpres
sion
DEFINE {patternVariable AS patternDefinationExpression [, patternVariable AS patternDefinat
ionExpression]*}
)];
```

Clause	Description
PARTITION BY	Optional. You can use the clause to divide the rows of an input table into partitions based on one or more partition key columns.
ORDER BY	Optional. You can use the clause to sort the rows of a partition based on one or more columns. If you use multiple columns to sort the rows, specify `EVENT TIME` or `PROCESS TIME` as the first column.
MEASURES	You can use the clause to define the output events that are generated based on the matched input events.
ONE ROW PER MATCH	If you use the clause, only one output event is generated for each match.
ONE ROW PER MATCH WITH TIMEOUT ROWS	If you use the clause, an output event is generated for each match or time-out. The time-out period is specified by the WITHIN clause in PATTERN. **Note** Blink V3.6.0 and later do not support the ONE ROW PER MATCH WITH TIMEOUT ROWS clause due to Calcite upgrades.
ALL ROWS PER MATCH	If you use the clause, an output event is generated for each input event upon each match.
ALL ROWS PER MATCH WITH TIMEOUT ROWS	If you use the clause, an output event is generated for each input event upon each match or time-out. The time-out period is specified by the WITHIN clause in PATTERN. **Note** Blink V3.6.0 and later do not support the ALL ROWS PER MATCH WITH TIMEOUT ROWS clause due to Calcite upgrades.

Clause	Description
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]	Optional. By default, `ONE ROW PER MATCH` is used.
AFTER MATCH SKIP TO NEXT ROW	If you use the clause, the next match starts to be performed from the event that follows the first event in the sequence of matched events.
AFTER MATCH SKIP PAST LAST ROW	If you use the clause, the next match starts to be performed from the event that follows the last event in the sequence of matched events.
AFTER MATCH SKIP TO FIRST patternItem	If you use the clause, the next match starts to be performed from the first event that corresponds to patternItem in the sequence of matched events.
AFTER MATCH SKIP TO LAST patternItem	If you use the clause, the next match starts to be performed from the last event that corresponds to patternItem in the sequence of matched events.
PATTERN	You can use the clause to specify the rule that the sequence of events to be identified must meet. You must enclose the rule in parentheses `()`. The rule is specified by a set of custom patternVariable variables. **Note** • If two patternVariable variables are separated with a space, no events exist between the events that correspond to the patternVariable variables. • If two patternVariable variables are separated with an arrow sign (->), other events may exist between the events that correspond to the patternVariable variables. • Blink V3.6.0 and later do not support the PATTERN clause due to Calcite upgrades.

Parameter description

- quantifier

  The quantifier parameter specifies the number of occurrences of events that meet the patternVariable definition.

Value	Description
*	Zero or multiple times.
+	One or more times.
?	Zero or once.
{n}	n times.

Value	Description
{n,}	At least n times.
{n, m}	Ranges from n times to m times.
{,m}	At most m times.

By default, greedy matching is performed. For example, if the PATTERN clause is `A -> B+ -> C` and the input is `a bc1 bc2 c`, the output is `a bc1 bc2 c`. In the input, bc1 and bc2 means that the results must match B and C. To perform reluctant matching, append the quantifier with a question mark (?). You can use the following reluctant quantifiers:

- `*?`
- `+?`
- `{n}?`
- `{n,}?`
- `{n, m}?`
- `{,m}?`

> **Note** Blink V3.x and later do not support (e1 e2+) greedy matching. You can use e1 e2+ e3 e3 as not e2 as an alternative. In the alternative method, at least one e3 entry must be included to make sure that the output data is returned as expected.

In this case, the output that is generated for the input and the PATTERN setting in the preceding example changes to a bc1 bc2,a bc1 bc2 c.

- The WITHIN clause defines the maximum time span of the events that meet the specified rule in an event sequence.
- The format of static windows is `INTERVAL 'string' timeUnit [ TO timeUnit ]`, for example, `INTERVAL '10' SECOND, INTERVAL '45' DAY, INTERVAL '10:20' MINUTE TO SECOND, INTERVAL '10:20.10' MINUTE TO SECOND, INTERVAL '10:20' HOUR TO MINUTE, INTERVAL '1-5' YEAR TO MONTH`.
- The format of dynamic windows is `INTERVAL intervalExpression`, for example, `INTERVAL A.windowTime + 10`. In this example, A is the first patternVariable variable in the PATTERN clause. When you specify intervalExpression, you can use only the first patternVariable variable in the PATTERN clause. When you specify the intervalExpression parameter, you can use user defined functions (UDFs). The intervalExpression result indicates the window size that is measured in milliseconds. The data type of the result must be LONG.
- The DEFINE statement specifies the meanings of patternVariable variables in the PATTERN clause. If a patternVariable variable is not defined in the DEFINE statement, the patternVariable variable is valid for each event.

- Functions in MEASURES and DEFINE statements

Function	Description

Function	Description
Row Pattern Column References	This function uses the format of `patternVariable.col`. This function is used to access the specified column of an event that corresponds to a patternVariable variable.
PREV	This function can be used in only the DEFINE statement. In most cases, this function is used in conjunction with the `Row Pattern Column References` function. This function is used to access the specified column of a previous event that has a specified offset. The event occurs before the events that are specified by the PATTERN clause. For example, you can use `DOWN AS DOWN.price < PREV(DOWN.price)`. In this example, `PREV(A.price)` specifies the `price` column value for the event that occurs before the current event. **Note** ○ `DOWN.price` is equivalent to `PREV(DOWN.price, 0)`. ○ `PREV(DOWN.price)` is equivalent to `PREV(DOWN.price, 1)`.
FIRST or LAST	In most cases, the FIRST or the LAST function is used in conjunction with the `Row Pattern Column References` function. You can use the FIRST or the LAST function to access the event that has a specified offset in the sequence of events. The sequence of events is specified by the PATTERN clause. The following examples are used to explain the functions: ○ `FIRST(A.price, 3)` specifies the fourth event in the sequence of events that match the `pattern A`. ○ `LAST(A.price, 3)` specifies the last but three event in the sequence of events that match the `pattern A`.

- Output columns

Function	Output column
ONE ROW PER MATCH	The columns that are specified in `PARTITION BY` and `MEASURES` are included. The columns that are specified in `PARTITION BY` do not need to be specified in `MEASURES` again.
ONE ROW PER MATCH WITH TIMEOUT ROWS	An output event is generated for each match or time-out. The time-out period is specified by the WITHIN clause in PATTERN.

> **Note**
>
> - When you define the PATTERN clause, we recommend that you define the WITHIN clause. If the WITHIN clause is not defined, the state size may grow larger.
> - The first column that is specified in the ORDER BY clause must be EVENT TIME or PROCESS TIME.

Examples

- Syntax in the example

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
PARTITION BY symbol
ORDER BY tstamp
MEASURES STRT.tstamp AS start_tstamp,
LAST(DOWN.tstamp) AS bottom_tstamp,
LAST(UP.tstamp) AS end_tstamp
ONE ROW PER MATCH
AFTER MATCH SKIP TO NEXT ROW
PATTERN (STRT DOWN+ UP+) WITHIN INTERVAL '10' SECOND
DEFINE
DOWN AS DOWN.price < PREV(DOWN.price),
UP AS UP.price > PREV(UP.price)
);
```

- Test data

timestamp (TIMESTAMP)	card_id(VARCHAR)	location (VARCHAR)	action (VARCHAR)
2018-04-13 12:00:00	1	Beijing	Consumption
2018-04-13 12:05:00	1	Shanghai	Consumption
2018-04-13 12:10:00	1	Shenzhen	Consumption
2018-04-13 12:20:00	1	Beijing	Consumption

- Syntax in the test

  Each credit card is identified by a unique card ID that is specified by the card_id parameter. If payments with a credit card are made within 10 minutes at two different locations, an alert is triggered. This helps you monitor unauthorized operations of credit cards.

```
CREATE TABLE datahub_stream (
    `timestamp`               TIMESTAMP,
    card_id                   VARCHAR,
    location                  VARCHAR,
    `action`                  VARCHAR,
    WATERMARK wf FOR `timestamp` AS withOffset(`timestamp`, 1000)
) WITH (
    type = 'datahub'
    ...
);
CREATE TABLE rds_out (
    start_timestamp               TIMESTAMP,
    end_timestamp                 TIMESTAMP,
    card_id                       VARCHAR,
    event                         VARCHAR
) WITH (
    type= 'rds'
    ...
);
--Define the computational logic.
insert into rds_out
select
`start_timestamp`,
`end_timestamp`,
card_id, `event`
from datahub_stream
MATCH_RECOGNIZE (
    PARTITION BY card_id   --Partition the table by card ID. The data that has the same c
ard ID is allocated to the same compute node.
    ORDER BY `timestamp`   --Sort events by time in a window.
    MEASURES               --Define how to generate output events based on the input even
ts that are matched.
        e2.`action` as `event`,
        e1.`timestamp` as `start_timestamp`,   --Specify the time of the first event as t
he start_timestamp value.
        LAST(e2.`timestamp`) as `end_timestamp` --Specify the time of the last event as t
he end_timestamp value.
    ONE ROW PER MATCH           --The system generates an output event for each match.
    AFTER MATCH SKIP TO NEXT ROW --The system performs the next match from the next row a
fter each match.
    PATTERN (e1 e2+) WITHIN INTERVAL '10' MINUTE  --Define two events: e1 and e2.
    DEFINE                     --Define the meanings of patternVariable variables in the
PATTERN clause.
        e1 as e1.action = 'Consumption',    --Mark the action of the e1 event as Consumpt
ion.
        e2 as e2.action = 'Consumption' and e2.location <> e1.location --Mark the action
of the e2 event as Consumption. The locations of e1 and e2 are different.
);
```

> **Note** In some scenarios, some data meets the CEP condition but no outputs are returned. This occurs because only data that meets the watermark > e2.ts condition is processed. No input data flows into the system after e2 and the watermark is always e2.ts -1000. As a result, the e2 data cannot be processed. Therefore, no outputs are returned.

- Test result

start_timestamp (TIMESTAMP)	end_timestamp (TIMESTAMP)	card_id (VARCHAR)	event (VARCHAR)
2018-04-13 12:00:00.0	2018-04-13 12:05:00.0	1	Consumption
2018-04-13 12:05:00.0	2018-04-13 12:10:00.0	1	Consumption

2.8. Create a view

In Realtime Compute, you can define a view to query data in one or more tables in a database. You can simplify the development process by using views.

A view displays a logical table that helps to describe the computational logic. It does not physically store any data. To simplify queries, you can use the SQL statement to define a manageable view. A view can be created based on fields of one or more tables in a database. Views do not require any disk space.

Syntax

```
CREATE VIEW viewName
        [ (columnName[ , columnName]*) ]
            AS queryStatement;
```

Example 1

```
CREATE VIEW  LargeOrders(r, t, c, u) AS
 SELECT rowtime,
   productId,
 c,
   units
 FROM Orders;
 INSERT INTO rds_output
 SELECT
 r,t,c,u
 FROM LargeOrders;
```

Example 2

Input example

a (VARCHAR)	b (BIGINT)	c (TIMESTAMP)
test1	1	1506823820000
test2	1	1506823850000
test1	1	1506823810000
test2	1	1506823840000
test2	1	1506823870000
test1	1	1506823830000
test2	1	1506823860000

Sample code

```
CREATE TABLE datahub_stream (
a VARCHAR,
b BIGINT,
c TIMESTAMP,
d AS PROCTIME()
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyuncs.com',
  project='blink_test',
  topic='window_count',
  accessId='yourAccessId',
  accessKey='yourAccessSecret'
);
CREATE TABLE rds_output (
a varchar,
b TIMESTAMP,
cnt BIGINT,
PRIMARY KEY(a)
)with(
  type = 'rds',
  url='jdbc:mysql://rm-bp1gz4k20****.mysql.rds.aliyuncs.com:3306/****',
  tableName='yourTableName',
  userName='yourAccessUserName',
  password='yourAccessPassword'
);
CREATE VIEW rds_view AS
SELECT a,
   CAST(HOP_START(d, interval '5' second, interval '30' second) AS TIMESTAMP) AS cc,
   sum(b) AS cnt
FROM datahub_stream
GROUP BY HOP(d, interval '5' second, interval '30' second),a;
INSERT INTO rds_output
SELECT
a,
cc,
cnt
FROM rds_view
WHERE cnt=4
```

Output example

a (VARCHAR)	b (TIMESTAMP)	cnt (BIGINT)
test2	2017-11-06 16:54:10	4

2.9. Window functions

2.9.1. Overview

This topic describes the window functions, time attributes, and window types that Flink SQL supports.

Window functions

Flink SQL supports aggregation over infinite windows. You do not need to explicitly define windows in your SQL statements. Flink SQL also supports aggregation over a specific window. For example, to count the number of users who clicked a URL in the last minute, you can define a window to collect the data about user clicks that occur in the last minute. Then, you can compute the data in the window to obtain the result.

Flink SQL supports window aggregates and over aggregates. This topic describes window aggregates. Window aggregates support the windows that are defined based on the following two time attributes: event time and processing time. For each time attribute, Flink SQL supports three window types: tumbling window, sliding window, and session window.

Time attributes

Flink SQL supports two time attributes: event time and processing time. Realtime Compute for Apache Flink aggregates data in windows based on the following time attributes:

- Event time: the event time that you provide in the table schema. In most cases, the event time is the original time when data is created.
- Processing time: the local system time at which the system processes an event.

> **Note** For more information about the time attributes, see Time attributes.

Cascading windows

The event time attribute of the Rowtime column no longer takes effect after a window operation is complete. You can use a helper function such as `TUMBLE_ROWTIME`, `HOP_ROWTIME`, or `SESSION_ROWTIME` to obtain `max(rowtime)` of the rowtime column in a window. You can use the obtained value as the rowtime of the time window. The value is `window_end - 1` and is of the TIMESTAMP data type. The TIMESTAMP value has the rowtime attribute. For example, if the time span for a window is `[00:00, 00:15]`, `00:14:59.999` is returned.

In the following example, 1-hour tumbling windows are used to aggregate data based on the aggregation results of 1-minute tumbling windows. This helps you meet various window requirements.

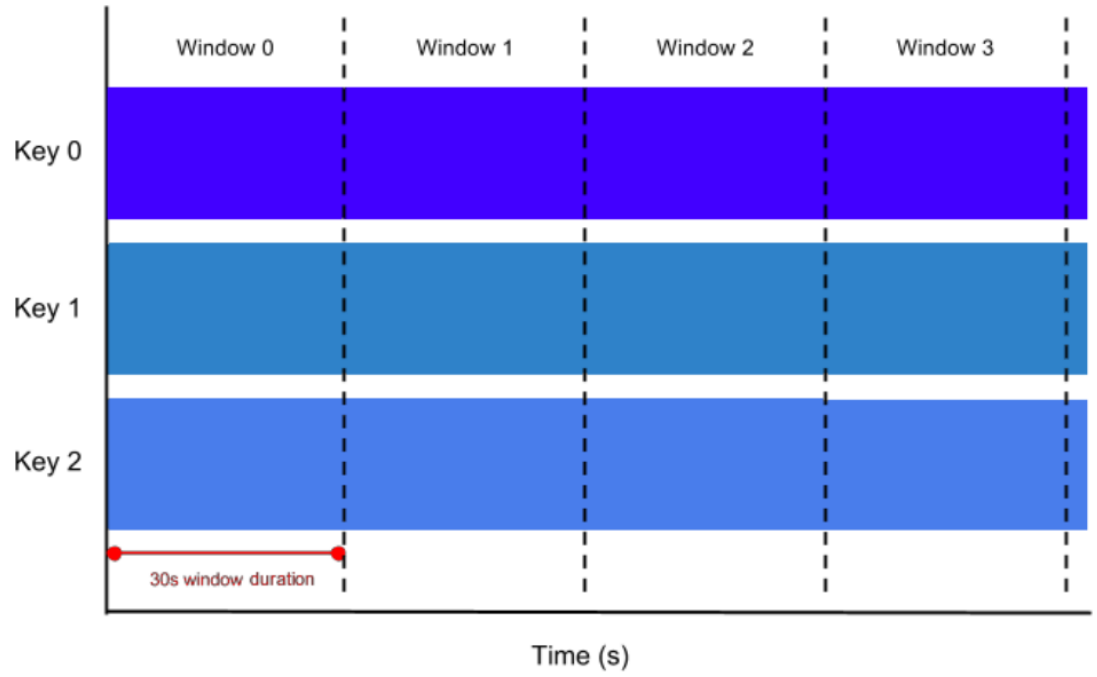
```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000)  --Define a watermark for the rowtime.
) with (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='print'
);
CREATE VIEW one_minute_window_output as
SELECT
  // Use each TUMBLE_ROWTIME value as the aggregation time for the level-two window.
  TUMBLE_ROWTIME(ts, INTERVAL '1' MINUTE) as rowtime,
  username,
  COUNT(click_url) as cnt
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
INSERT INTO tumble_output
SELECT
  TUMBLE_START(rowtime, INTERVAL '1' HOUR),
  TUMBLE_END(rowtime, INTERVAL '1' HOUR),
  username,
  SUM(cnt)
FROM one_minute_window_output
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), username;
```

2.9.2. Tumbling window

This topic describes how to use the tumbling window function in Realtime Compute for Apache Flink.

Introduction to a tumbling window

A tumbling window assigns segments of a data stream to a window with a specified size. Generally, tumbling windows are fixed in size and do not overlap each other. For example, if a 5-minute tumbling window is defined, an infinite data stream is divided based on the time period into windows such as `[0:00 - 0:05)`, `[0:05, 0:10)`, and `[0:10, 0:15)`. The following figure shows a 30-second tumbling window.



Syntax

The TUMBLE function is used to define a tumbling window in a GROUP BY clause.

```
TUMBLE(<time-attr>, <size-interval>)
<size-interval>: INTERVAL 'string' timeUnit
```

Note The `<time-attr>` parameter must be a valid time attribute in a time stream to specify whether the time is the processing time or event time. For more information, see Overview, Time attributes, and Watermark.

Window identifier functions

A window identifier function specifies the start time, end time, or the time attribute of a window. The time attribute is used to aggregate lower-level windows.

Window identifier function	Return type	Description
`TUMBLE_START(time-attr, size-interval)`	TIMESTAMP	Returns the start time (including the boundary value) of a window. For example, if the window is `[00:10, 00:15)`, `00:10` is returned.

Window identifier function	Return type	Description
`TUMBLE_END(time-attr, size-interval)`	TIMESTAMP	Returns the end time (including the boundary value) of a window. For example, if the window is `[00:00, 00:15]`, `00:15` is returned.
`TUMBLE_ROWTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	Returns the end time (excluding the boundary value) of a window. For example, if the window is `[00:00, 00:15]`, `00:14:59.999` is returned. The return value is a rowtime attribute, based on which time operations can be performed. For example, a cascading window function can be used only in windows based on the event time.
`TUMBLE_PROCTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	Returns the end time (excluding the boundary value) of a window. For example, if the window is `[00:00, 00:15]`, `00:14:59.999` is returned. The return value is a proctime attribute, based on which time operations can be performed. For example, a cascading window function can be used only in windows defined based on the processing time.

Example of counting the number of clicks per user on a specific website every one minute based on the event time

- Test data

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	*http://taobao.com/xxx*	`2017-10-10 10:00:00.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:00:10.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:00:49.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:01:05.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:01:58.0`
Timo	*http://taobao.com/xxx*	`2017-10-10 10:02:10.0`

- Test statements

```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000) -- Define a watermark for rowtime.
) with (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='RDS'
);
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE) as window_start,
TUMBLE_END(ts, INTERVAL '1' MINUTE) as window_end,
username,
COUNT(click_url)
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
```

- Test results

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	Jark	3
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1

Example of counting the number of clicks per user on a specific website every one minute based on the processing time

- Test data

username (VARCHAR)	click_url (VARCHAR)
Jark	*http://taobao.com/xxx*
Jark	*http://taobao.com/xxx*
Jark	*http://taobao.com/xxx*

username (VARCHAR)	click_url (VARCHAR)
Jark	*http://taobao.com/xxx*
Jark	*http://taobao.com/xxx*
Timo	*http://taobao.com/xxx*

- Test statements

```
CREATE TABLE window_test (
  username   VARCHAR,
  click_url  VARCHAR,
  ts as PROCTIME()
) WITH (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='print'
);
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE),
TUMBLE_END(ts, INTERVAL '1' MINUTE),
username,
COUNT(click_url)
FROM window_test
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
```

- Test results

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Jark	5
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Timo	1

> **Note** Local debugging is instantaneous and the processing time may be less than 1s. Therefore, if the processing time attribute is used to aggregate data in windows, local debugging may fail.

2.9.3. Sliding windows

This topic describes how to use sliding window functions in Realtime Compute for Apache Flink.

> **Note** In Realtime Compute for Apache Flink, sliding windows cannot be used in conjunction with LAST_VALUE, FIRST_VALUE, or TopN functions.

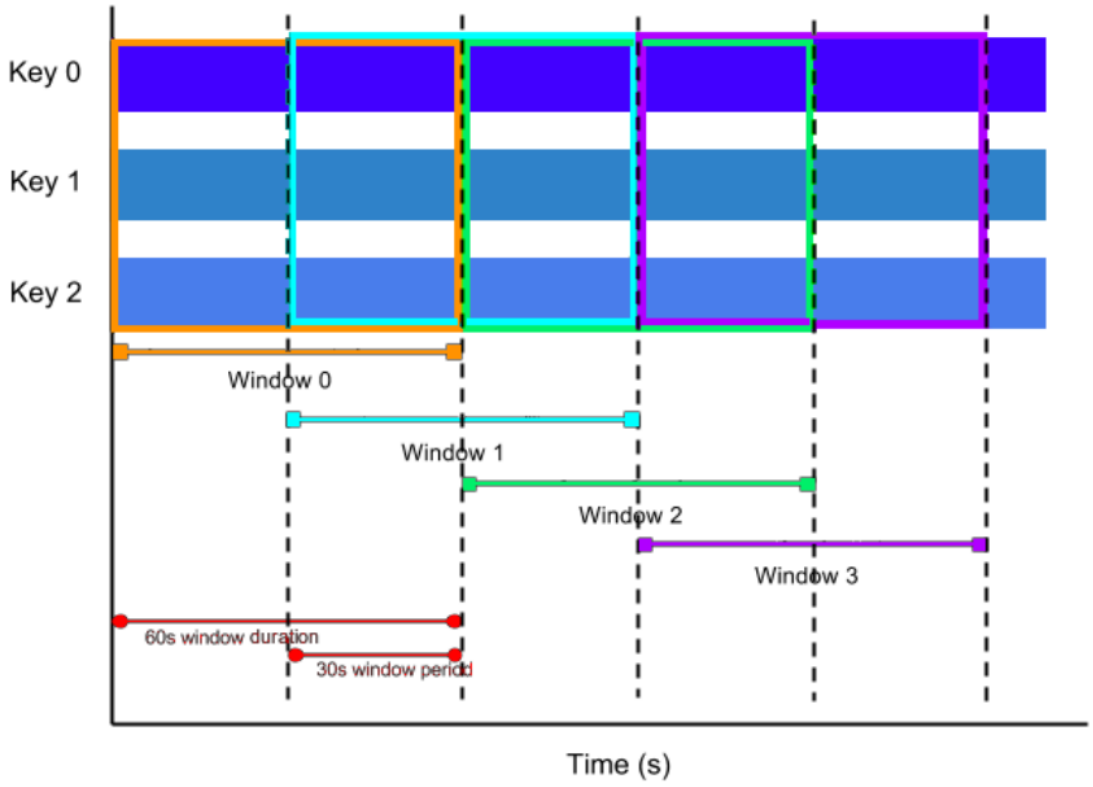
Introduction

A sliding window is also known as a hop window. Unlike tumbling windows, sliding windows can overlap with each other.

A sliding window is defined by the following parameters: slide and size. The slide parameter specifies the length of a sliding step. The size parameter specifies the size of the window.

- If the value of slide is smaller than that of size, the windows overlap with each other and each element is assigned to multiple windows.
- If the value of slide is equal to that of size, the windows are tumbling windows.
- If the value of slide is greater than that of size, the windows do not overlap with each other but are separated with gaps.

In most cases, most elements are assigned to multiple windows and the windows overlap with each other. Sliding windows are used to calculate moving averages. For example, to calculate the data average in the last 5 minutes every 10 seconds, set slide to *10* seconds and set size to *5* minutes. The following figure shows sliding windows for which the slide value is 30 seconds and the size value is 1 minute.



Syntax of sliding window functions

You can use the HOP function to define a sliding window in a GROUP BY clause.

```
HOP(<time-attr>, <slide-interval>,<size-interval>)
<slide-interval>: INTERVAL 'string' timeUnit
<size-interval>: INTERVAL 'string' timeUnit
```

> **Note**
>
> The `<time-attr>` parameter must be a valid time attribute field in a stream. This parameter specifies whether the time attribute is processing time or event time. For more information about Time attributes and Watermark, see Overview.

Window identifier functions

A window identifier function specifies the start time, end time, or time attribute of a specified window. The time attribute is used to aggregate lower-level windows.

Function	Return type	Description
`HOP_START (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	Returns the start time of the window. The boundary for the start time is included in the time span of the window. For example, if the time span of a window is `[00:10, 00:15)`, `00:10` is returned.
`HOP_END (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	Returns the end time of the window. The boundary for the end time is included in the time span of the window. For example, if the time span of a window is `[00:00, 00:15)`, `00:15` is returned.
`HOP_ROWTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time of the window. The boundary for the end time is excluded from the time span of the window. For example, if the time span of a window is `[00:00, 00:15)`, `00:14:59.999` is returned. The returned value is a rowtime attribute value based on which time operations can be performed. This function can be used in only the windows that are based on the event time, such as cascading windows. For more information, see Cascading windows.
`HOP_PROCTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time of the window. The boundary for the end time is excluded from the time span of the window. For example, if the time span of a window is `[00:00, 00:15)`, `00:14:59.999` is returned. The returned value is a proctime attribute value based on which time operations can be performed. This function can be used in only the windows that are based on the processing time, such as cascading windows. For more information, see Cascading windows.

Examples

In the following example, a 1-minute window slides once every 30 seconds. You can use the windows to count the number of clicks per user over the last minute every 30 seconds.

- Test data

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	http://taobao.com/xxx	2017-10-10 10:00:00.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:10.0
Jark	http://taobao.com/xxx	2017-10-10 10:00:49.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:05.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:58.0
Timo	http://taobao.com/xxx	2017-10-10 10:02:10.0

- Test statements

```
CREATE TABLE user_clicks (
    username VARCHAR,
    click_url VARCHAR,
    ts TIMESTAMP,
    WATERMARK wk FOR ts AS WITHOFFSET (ts, 2000)--Define the watermark for the rowtime.
) WITH ( TYPE = 'datahub',
        ...) ;
CREATE TABLE hop_output (
    window_start TIMESTAMP,
    window_end TIMESTAMP,
    username VARCHAR,
    clicks BIGINT
) WITH (TYPE = 'rds',
        ...) ;
INSERT INTO
    hop_output
SELECT
    HOP_START (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    HOP_END (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    username,
    COUNT (click_url)
FROM
    user_clicks
GROUP BY
    HOP (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    username
```

- Test result

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 09:59:30.0	2017-10-10 10:00:30.0	Jark	2
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	Jark	3
2017-10-10 10:00:30.0	2017-10-10 10:01:30.0	Jark	2
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:01:30.0	2017-10-10 10:02:30.0	Jark	1
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1
2017-10-10 10:02:30.0	2017-10-10 10:03:30.0	Timo	1

If a sliding window cannot read the time at which data enters the window, the start time of the first window is moved forward. You can use the following formula to calculate the time interval by which the start time is moved forward: Time interval = Window duration - Sliding step.

Window duration (seconds)	Sliding step (seconds)	Event Time	Start time of the first window	End time of the first window
120	30	2019-07-31 10:00:00.0	2019-07-31 09:58:30.0	2019-07-31 10:00:30.0
60	10	2019-07-31 10:00:00.0	2019-07-31 09:59:10.0	2019-07-31 10:00:10.0

2.9.4. Session window

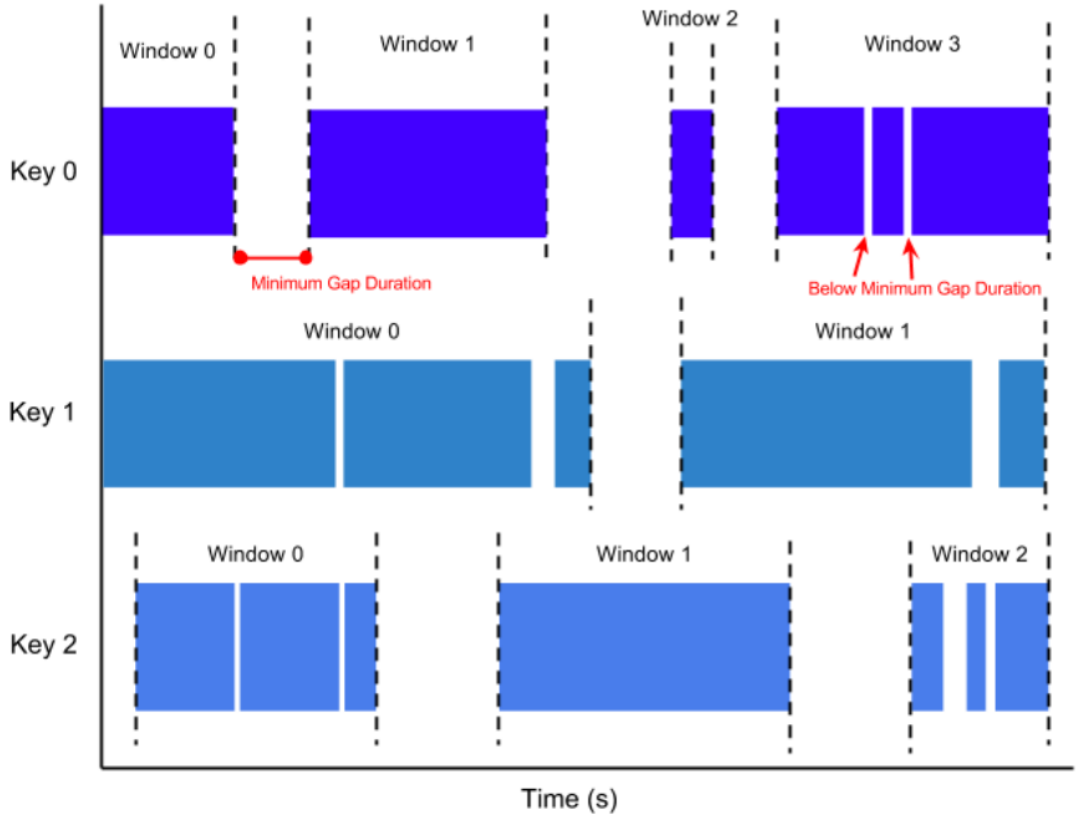
This topic describes how to use the session window function in Realtime Compute for Apache Flink.

Description

A session window groups elements by session activity. Unlike tumbling and sliding windows, session windows do not overlap and are not fixed in size. If a session window does not receive elements within a period of time, the session is disconnected and the window is closed.

A session window is configured by using a gap, which defines the length of the inactive period. For example, a data stream that represents mouse click activities may include highly clustered mouse click events, separated by inactive periods. The data that arrives after a specified gap is assigned to a new window.

The following figure shows a session window. Different keys form different windows due to differences in data distribution.



Syntax

The SESSION function is used to define a session window in a GROUP BY clause.

```
SESSION(<time-attr>, <gap-interval>)
<gap-interval>: INTERVAL 'string' timeUnit
```

Note The `<time-attr>` parameter must be a valid time attribute in a data stream to specify whether the time is the processing time or event time. For more information, see Overview, Time attributes, and Watermark.

Window identifier functions

A window identifier function specifies the start or end time of a window, or the window time attribute for the aggregation of lower-level windows.

Window identifier function	Return type	Description
`SESSION_START(<time-attr>, <gap-interval>)`	Timestamp	Returns the start time (including the boundary value) of the window. For example, if the window is `[00:10, 00:15]`, `00:10` is returned. This is the first time recorded in the session window.
`SESSION_END(<time-attr>, <gap-interval>)`	Timestamp	Returns the end time (including the boundary value) of the window. For example, if the window is `[00:00, 00:15]`, `00:15` is returned. This is the last time recorded in the session window plus `<gap-interval>`.
`SESSION_ROWTIME(<time-attr>, <gap-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time (excluding the boundary value) of the window. For example, if the window is `(00:00, 00:15)`, `00:14:59.999` is returned. The return value is an event time attribute, based on which time type operations such as window cascading can be performed. The function applies only to windows based on the event time.
`SESSION_PROCTIME(<time-attr>, <gap-interval>)`	TIMESTAMP (rowtime-attr)	Returns the end time (excluding the boundary value) of the window. For example, if the window is `(00:00, 00:15)`, `00:14:59.999` is returned. The return value is a processing time attribute, based on which time type operations such as window cascading can be performed. The function applies only to windows based on the processing time.

Example

The following example describes how to compute the number of clicks per user during each active session. The session timeout interval is 30 seconds.

- Test data

username (VARCHAR)	click_url (VARCHAR)	ts (TIMESTAMP)
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:00.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:10.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:49.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:01:05.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:01:58.0`
Timo	`http://taobao.com/xxx`	`2017-10-10 10:02:10.0`

- Test statements

```
CREATE TABLE user_clicks(
username varchar,
click_url varchar,
ts timeStamp,
WATERMARK wk FOR ts as withOffset(ts, 2000) -- Define a watermark for rowtime.
) with (
type='datahub',
...
);
CREATE TABLE session_output(
window_start TIMESTAMP,
window_end TIMESTAMP,
username VARCHAR,
clicks BIGINT
) with (
type='rds',
...
);
INSERT INTO session_output
SELECT
SESSION_START(ts, INTERVAL '30' SECOND),
SESSION_END(ts, INTERVAL '30' SECOND),
username,
COUNT(click_url)
FROM user_clicks
GROUP BY SESSION(ts, INTERVAL '30' SECOND), username;
```

- Test results

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 10:00:00.0	2017-10-10 10:00:40.0	Jark	2
2017-10-10 10:00:49.0	2017-10-10 10:01:35.0	Jark	2
2017-10-10 10:01:58.0	2017-10-10 10:02:28.0	Jark	1
2017-10-10 10:02:10.0	2017-10-10 10:02:40.0	Timo	1

2.9.5. OVER window

An OVER window is a standard window used in traditional databases. Over aggregate is different from window aggregate. In streaming data that uses OVER windows, each element corresponds to an OVER window. An OVER window can be determined based on an actual row or an actual value (timestamp value) of an element. Elements of a stream are distributed across multiple windows.

In a stream that applies the OVER window, each element corresponds to an OVER window and triggers data computing once. The row determined by each element that triggers computing is the last row of the window where the element is located. In the underlying implementation of Realtime Compute, the OVER window data is centrally managed. Only one copy of the data is stored. Logically, an OVER window is created for each element. Realtime Compute for Apache Flink calculates the data for each OVER window and then deletes the data that is no longer used after the calculation is complete.

Syntax

```
SELECT
    agg1(col1) OVER (definition1) AS colName,
    ...
    aggN(colN) OVER (definition1) AS colNameN
FROM Tab1;
```

- agg1(col1): aggregates input data based on the col1 column specified by GROUP BY.
- OVER (definition1): defines an OVER window.
- AS colName: specifies the alias of a column.

> **Note**
> - OVER (definition1) for agg1 through aggN must be the same.
> - The alias specified by AS can be queried by using an outer SQL statement.

Window types

In Flink SQL, OVER windows are defined in compliance with the standard SQL syntax. The traditional OVER windows are not classified into fine-grained window types. OVER windows are classified into the following two types based on the ways of determining computed rows:

- ROWS OVER window: Each row of elements is treated as a new computed row. A new window is generated for each row.
- RANGE OVER window: All rows of elements with the same timestamp are treated as one computed row and are assigned to the same window.

Attributes

Orthogonal attribute	Description	proctime	eventtime
ROWS OVER Window	A window is determined based on the actual row of an element.	Supported	Supported
RANGE OVER Window	A window is determined based on the timestamp value of an element.	Supported	Supported

ROWS OVER window

- Description

For a ROWS OVER window, a window is generated for each element.

- Syntax

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     ROWS
     BETWEEN (UNBOUNDED | rowCount) PRECEDING AND CURRENT ROW) AS colName, ...
FROM Tab1;
```

  - value_expression: specifies the value expression used for partitioning.
  - timeCol: specifies the time field used to sort elements.
  - rowCount: specifies the number of rows that precede the current row.

- Example

  This example describes bounded ROWS OVER windows. Assume that an on-sale product table contains item IDs, item types, launch time, and prices. Calculate the highest price among the three products similar to the current product before the current product is on sale.

  - Test data

Item ID	Item type	Launch time	Price
ITEM001	Electronic	2017-11-11 10:01:00	20
ITEM002	Electronic	2017-11-11 10:02:00	50
ITEM003	Electronic	2017-11-11 10:03:00	30
ITEM004	Electronic	2017-11-11 10:03:00	60
ITEM005	Electronic	2017-11-11 10:05:00	40
ITEM006	Electronic	2017-11-11 10:06:00	20
ITEM007	Electronic	2017-11-11 10:07:00	70
ITEM008	Clothes	2017-11-11 10:08:00	20

- Test statements

```
CREATE TABLE tmall_item(
   itemID VARCHAR,
   itemType VARCHAR,
   onSellTime TIMESTAMP,
   price DOUBLE,
   WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
  type = 'sls',
   ...
);
SELECT
    itemID,
    itemType,
    onSellTime,
    price,
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY onSellTime
        ROWS BETWEEN 2 preceding AND CURRENT ROW) AS maxPrice
  FROM tmall_item;
```

- Test results

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	60
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

RANGE OVER window

- Description

  For a RANGE OVER window, all elements with the same timestamp are assigned to the same window.

- Syntax

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     RANGE
     BETWEEN (UNBOUNDED | timeInterval) PRECEDING AND CURRENT ROW) AS colName,
...
FROM Tab1;
```

  - value_expression: specifies the value expression used for partitioning.
  - timeCol: specifies the time field used to sort elements.
  - timeInterval: specifies the time interval between the time of the current row and that of the element row to which it can be traced back.

- Example

  This example describes bounded RANGE OVER windows. Assume that an on-sale product table contains item IDs, item types, launch time, and prices. Calculate the highest price among similar products that are on sale two minutes earlier than the current product.

  - Test data

Item ID	Item type	Launch time	Price
ITEM001	Electronic	2017-11-11 10:01:00	20
ITEM002	Electronic	2017-11-11 10:02:00	50
ITEM003	Electronic	2017-11-11 10:03:00	30
ITEM004	Electronic	2017-11-11 10:03:00	60
ITEM005	Electronic	2017-11-11 10:05:00	40
ITEM006	Electronic	2017-11-11 10:06:00	20
ITEM007	Electronic	2017-11-11 10:07:00	70
ITEM008	Clothes	2017-11-11 10:08:00	20

- Test statements

```
CREATE TABLE tmall_item(
   itemID VARCHAR,
   itemType VARCHAR,
   onSellTime TIMESTAMP,
   price DOUBLE,
   WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
  type = 'sls',
   ...
);
SELECT
    itemID,
    itemType,
    onSellTime,
    price,
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY onSellTime
        RANGE BETWEEN INTERVAL '2' MINUTE preceding AND CURRENT ROW) AS maxPrice
  FROM tmall_item;
```

- Test results

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	40
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

2.10. Logical operations

2.10.1. =

Syntax

```
A = B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 = int2;
```

- Output example

aa (INT)
97

2.10.2. >

Syntax

```
A > B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is greater than B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	65	100

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 > int2;
```

- Output example

aa (INT)
97

2.10.3. >=

Syntax

```
A >= B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is greater than or equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	65	65
9	6	61

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 >= int2;
```

- Output example

aa (INT)
97
9

2.10.4. <

Syntax

```
A < B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is smaller than B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 < int2;
```

- Output example

aa (INT)
97
9

2.10.5. <=

Syntax

```
A <= B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is smaller than or equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	66	65
9	6	5

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 <= int2;
```

- Output example

aa (INT)
97

aa (INT)
9

2.10.6. <>

Syntax

```
A <> B
```

Input parameters

Parameter	Data type
A	INT
B	INT

Description

Returns TRUE if A is not equal to B, and returns FALSE in other cases.

Example

- Input example

int1 (INT)	int2 (INT)	int3 (INT)
97	66	6

- Sample code

```
SELECT int1 as aa
FROM T1
WHERE int3 <> int2;
```

- Output example

aa (INT)
97

2.10.7. AND

Syntax

```
A AND B
```

Argument: Both A and B are of the BOOLEAN type.

Description

TRUE is returned if both A and B are TRUE. Otherwise, FALSE is returned.

Sample code

Input example

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1 = 255 AND int3 = 65;
```

Output example

aa (int)
97

2.10.8. BETWEEN AND

Syntax

```
A BETWEEN B AND C
```

Arguments: A, B, and C are values of the DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME type.

Description

Use the BETWEEN operator to select a value that is between the two given argument values.

Example 1

Input example

int1 (INT)	int2 (INT)	int3 (INT)
90	80	100
11	10	7

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int1 BETWEEN int2 AND int3;
```

Output example

aa(int)
90

Example 2

Input example

var1 (varchar)	var1 (varchar)	var1 (varchar)
b	a	c

Sample code

```
SELECT var1 as aa
FROM T1
WHERE var1  BETWEEN var2 AND var3;
```

Output example

aa (varchar)
b

Example 3

Input example

TIMESTAMP1 (TIMESTAMP)	TIMESTAMP2 (TIMESTAMP)	TIMESTAMP3 (TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:20	1969-07-20 20:17:45

Sample code

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1  BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

Output example

aa (TIMESTAMP)
1969-07-20 20:17:30

2.10.9. IS NOT FALSE

Syntax

```
A IS NOT FALSE
```

Argument: A of the BOOLEAN type

Description

TRUE is returned if A is true. FALSE is returned if A is false.

Examples

Input example

int1(INT)	int2(INT)
255	97

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS NOT FALSE;
```

Output example

aa(int)
97

2.10.10. IS NOT NULL

Syntax

```
A IS NOT NULL
```

Arguments: A of the INT type

Description

TRUE is returned if A is null. Otherwise, FALSE is returned.

Examples

Input example

int1(INT)	int2(VARCHAR)
97	null
9	ww123

Note Here, null indicates empty, rather than a null string.

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NOT NULL;
```

Output example

aa(int)
9

2.10.11. IS NOT TRUE

Syntax

```
A IS NOT TRUE
```

Argument: A specifies a condition of the BOOLEAN type.

Description

FALSE is returned if the condition is TRUE. TRUE is returned if the condition is FALSE.

Examples

Input example

int1 (INT)	int2 (INT)
255	97

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1 = 25 IS NOT TRUE;
```

Output example

aa (int)
97

2.10.12. IS NOT UNKNOWN

Syntax

```
A IS NOT UNKNOWN
```

Argument: A is of the BOOLEAN type.

Description

A is a logical comparison expression, such as 6 < 8.

The value of A is TRUE or FALSE if a logical comparison between two numeric values is performed successfully. If a numeric value is compared with a non-numeric value, the logical comparison fails, and the value of A is neither TRUE nor FALSE. **IS NOT UNKNOWN** is used to check for such failures. It returns **FALSE** if the value of A is neither TRUE nor FALSE and returns **TRUE** if the value of A is TRUE or FALSE.

Examples

Input example

int1 (INT)	int2 (INT)
255	97

Sample code 1

```
SELECT int2 as aa
FROM T1
WHERE int1 = 25 IS NOT UNKNOWN;
```

Output example 1

aa (int)
97

Sample code 2

```
SELECT int2 as aa
FROM T1
WHERE int1 < null IS NOT UNKNOWN;
```

Output example 2

aa (int)
Null

2.10.13. IS NULL

Syntax

```
value IS NULL
```

Argument: value specifies a value of any type.

Description

TRUE is returned if A is null. Otherwise, FALSE is returned.

Examples

Input example

int1 (INT)	int2 (VARCHAR)
97	null
9	www

> **Note** In this case, null indicates empty, rather than a null string.

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NULL;
```

Output example

aa (int)
97

2.10.14. IS TRUE

Syntax

```
A IS TRUE
```

Argument: A of the BOOLEAN type

Description

TRUE is returned if A is TRUE. FALSE is returned if A is FALSE.

Examples

Input example

int1(INT)	int2(INT)
255	97

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS TRUE;
```

Output example

aa(int)
97

2.10.15. IS UNKNOWN

Syntax

```
A IS UNKNOWN
```

Argument: A of the BOOLEAN type

Description

A is a logical comparison expression, such as 6 <> 8.

The value of A is TRUE or FALSE if a logical comparison between two numeric values is performed successfully. If a numeric value is compared with a non-numeric value, the logical comparison fails, and the value of A is neither TRUE nor FALSE. **IS UNKNOWN** is used to check for such failures. It returns **TRUE** if the value of A is neither TRUE nor FALSE and returns **FALSE** if the value of A is TRUE or FALSE.

Examples

Input example

int1(INT)	int2(INT)
255	97

Sample code 1

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS UNKNOWN;
```

Output example 1

aa(int)
Null

Sample code 2

```
SELECT int2 as aa
FROM T1
WHERE int1 > null IS UNKNOWN;
```

Output example 2

aa(int)
97

2.10.16. LIKE

Syntax

```
A LIKE B
```

Arguments: A and B of the VARCHAR type

Description

TRUE is returned if A matches B. Otherwise, FALSE is returned.

> **Note** The percent sign (%) can be used as a wildcard.

Examples

Sample code

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

Sample code 1

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR2 LIKE 'ss%';
```

Output example 1

aa(int)
90
9

Sample code 2

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ss';
```

Output example 2

aa(int)
90

Sample code 3

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ho%';
```

Output example 3

aa(int)
99

2.10.17. NOT

Syntax

```
NOT A
```

Argument: A of the BOOLEAN type

Description

FALSE is returned if A is TRUE. TRUE is returned if A is FALSE.

Examples

Input example

int2(INT)	int3(INT)
97	65

Sample code

```
SELECT int2 as aa
FROM T1
WHERE  NOT int3=62;
```

Output example

aa(int)
97

2.10.18. NOT BETWEEN AND

Syntax

```
A NOT BETWEEN B AND C
```

Arguments: A, B, and C can be of the DOUBLE, BIGINT, INT, VARCHAR, DATE, TIMESTAMP, or TIME types.

Description

The NOT BETWEEN operator is used to select a value that is not between the two given argument values.

Example 1

Input example

int1 (INT)	int2 (INT)	int3 (INT)
90	97	80
11	10	7

Sample code

```
SELECT int1 as aa
FROM T1
WHERE int1 NOT BETWEEN int2 AND int3;
```

Output example

aa (int)
11

Example 2

Input example

var1 (varchar)	var1 (varchar)	var1 (varchar)
d	a	c

Sample code

```
SELECT var1 as aa
FROM T1
WHERE var1 NOT BETWEEN var2 AND var3;
```

Output example

aa (varchar)
d

Example 3

Input example

TIMESTAMP1 (TIMESTAMP)	TIMESTAMP2 (TIMESTAMP)	TIMESTAMP3 (TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:40	1969-07-20 20:17:45

Sample code

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 NOT BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

Output example

aa (TIMESTAMP)
1969-07-20 20:17:30

2.10.19. OR

Syntax

```
A OR B
```

Arguments: Both A and B are of the BOOLEAN type

Description

TRUE is returned if both A and B are TRUE. Otherwise, FALSE is returned.

Examples

Input example

int1 (INT)	int2 (INT)	int3 (INT)
255	97	65

Sample code

```
SELECT int2 as aa
FROM T1
WHERE int1 = 255 OR int3 = 65;
```

Output example

aa (int)
97

2.10.20. IN

Syntax

```
SELECT column_name(s)
  FROM table_name
 WHERE column_name IN (value1,value2,...)
```

Arguments: Both value1 and value2 must be constants.

Description

Query records that match the arguments.

Examples

Input example

id (INT)	LastName (Varchar)
1	Adams

id (INT)	LastName (Varchar)
2	Bush
3	Carter

Sample code

```
SELECT * FROM T1
 WHERE LastName IN ('Adams','Carter')
```

Output example

id (INT)	LastName (Varchar)
1	Adams
3	Carter

2.10.21. NOT IN

Syntax

```
SELECT column_name(s)
  FROM table_name
 WHERE column_name NOT IN (value1,value2,...)
```

Input parameters: value1 and value2, which must be constants

Description

Searches for records that do not match the values in the arguments.

Example

Input example

id (INT)	LastName (VARCHAR)
1	Adams
2	Bush
3	Carter

Sample code

```
SELECT * FROM T1
 WHERE LastName NOT IN ('Adams','Carter')
```

Output example

id (INT)	LastName (VARCHAR)
2	Bush

2.10.22. IS DISTINCT FROM

Syntax

```
A IS DISTINCT FROM B
```

Arguments: A and B of generic types

Description

Check whether two arguments are equal. TRUE is returned if their data types or values are different. FALSE is returned if their data types and values are the same. If either or both arguments are null, they are considered to be the same.

Examples

Input example

A(Int)	B(Varchar)
97	97
null	sss
null	null

Sample code

```
SELECT
 A IS DISTINCT FROM B as `result`
FROM T1
```

Output example

result(BOOLEAN)
true
true
false

2.10.23. IS NOT DISTINCT FROM

Syntax

```
A IS NOT DISTINCT FROM B
```

Arguments: A and B of generic types

Description

Check whether two arguments are equal. FALSE is returned if their data types or values are different. TRUE is returned if their data types and values are the same. If either or both arguments are null, they are considered to be the same.

Examples

Input example

A(Int)	B(Varchar)
97	97
null	sss
null	null

Sample code

```
SELECT
 A IS NOT DISTINCT FROM B as `result`
FROM T1
```

Output example

result(BOOLEAN)
false
false
true

2.11. Built-in functions

2.11.1. String functions

2.11.1.1. Overview

CHAR_LENGTH

- Syntax: `CHAR_LENGTH(A)`

- Input parameter: The A parameter is of the VARCHAR data type.
- Description: Returns the number of characters in a string.

Example

Input example

var1 (VARCHAR)
ss
231ee

Sample code

```
SELECT  CHAR_LENGTH(var1) as aa
FROM T1;
```

Output example

aa (INT)
2
5

ASCII2STR

- Syntax: `string ascii2str(int ascii)`
- Description: Converts a number into an ASCII character. If any argument is NULL, the return value is NULL.
- Input parameter: The value of the ascii parameter ranges from 1 to 127. If the value fall out of the range, an exception occurs.
- Example:

```
ascii2str(9) = '\t'
ascii2str(65) = 'A'
```

CONCAT

- Syntax: `string concat(string a, string b, ...)`
- Description: Concatenates all input strings and returns a new string. If any argument is NULL, the return value is NULL.
- Example:

```
concat('abc', 'def', 'gh') = 'abcdefgh'
concat('abc', NULL) = NULL
```

CONCAT_WS

- Syntax: `string concat_ws(string separator, string a, string b, ...)`
- Description: Concatenates all input strings. Different from the CONCAT function, this function adds a separator between two concatenated strings. If any argument is NULL, the return value is NULL.
- Example:

```
concat_ws(':', 'abc', 'def', 'gh') = 'abc:def:gh'
concat(':', 'abc', NULL) = NULL
```

SUBSTR

- Syntax

```
string substr(string a, int start)
string substr(string a, int start, int len)
```

- Description: Extracts a substring of the specified length from a string, starting from the specified position. If the length is not specified, a substring is extracted from the specified position to the end of the string. The value of the start parameter begins from 1. If the value is negative, a substring is extracted from the end of the string. If any argument is NULL, the return value is NULL.
- Example:

```
substr('galaxy_sql', 5) = 'xy_sql'
substr('galaxy_sql', -5) = 'y_sql'
substr('galaxy_sql', 5, 2) = 'xy'
```

LOWER

- Syntax

```
string lower(string a)
string lcase(string a)
```

- Description: Converts uppercase letters in a string into lowercase letters. If any argument is NULL, the return value is NULL.
- Example: `lower('fOoBaR') = 'foobar'`

UPPER

- Syntax

```
string upper(string a)
string ucase(string a)
```

- Description: Converts lowercase letters in a string into uppercase letters. If any argument is NULL, the return value is NULL.

- Example: `upper('fOoBaR') = 'FOOBAR'`

TRIM

- Syntax: `string trim(string a)`
- Description: Removes the leading and trailing spaces of a string. If any argument is NULL, the return value is NULL.
- Example: `trim(' foobar\t ') = 'foobar\t'`

LTRIM

- Syntax: `string ltrim(string a)`
- Description: Removes the leading spaces of a string. If any argument is NULL, the return value is NULL.
- Example: `ltrim(' foobar ') = 'foobar '`

RTRIM

- Syntax: `string rtrim(string a)`
- Description: Removes the trailing spaces of a string. If any argument is NULL, the return value is NULL.
- Example: `rtrim(' foobar ') = ' foobar'`

LPAD

- Syntax: `string lpad(string str, int len, string pad)`
- Description: Left-pads a string (str) with another string (pad), to the specified length. If any argument is NULL, the return value is NULL.
- Example:

```
lpad('hi', 5, '??') = '??? hi'
lpad('hi', 1, '??') = 'h'
lpad('---', 10, 'abc') = 'abcabca---'
```

RPAD

- Syntax: `string rpad(string str, int len, string pad)`
- Description: Right-pads a string (str) with another string (pad), to the specified length. If any argument is NULL, the return value is NULL.
- Example:

```
rpad('hi', 5, '??') = 'hi???'
rpad('hi', 1, '??') = 'h'
rpad('---', 10, 'abc') = '---abcabca'
```

LENGTH

- Syntax: `int length(string str)`
- Description: Returns the length of a string. If any argument is NULL, the return value is NULL.

- Example:

```
length('hi') = 2
length('I') = 1
length(NULL) = NULL
```

REPEAT

- Syntax: `string repeat(string str, int n)`
- Description: Returns a new string that repeats the specified string for the specified number of times. If any argument is NULL, the return value is NULL.
- Example: `repeat('hi', 2) = 'hihi'`

REVERSE

- Syntax: `string reverse(string str)`
- Description: Reverses a string. If any argument is NULL, the return value is NULL.
- Example: `reverse('abc') = 'cba'`

SPLIT_EX

- Syntax: `string split_ex(string str, string sep, int index)`
- Description: Uses a separator to split a string into several segments and returns the segment identified by the index parameter. If the segment identified by the index parameter does not exist, NULL is returned. The value of the index parameter starts from 0. If any argument is NULL, the return value is NULL.
- Example:

```
split_ex('1.2.3.4', '.', 1) = '2'
split_ex('1.2.3.4', '.', -1) = NULL
split_ex('1.2.3.4', '.', 4) = NULL
```

REGEXP_REPLACE

- Syntax: `string regexp_replace(string str, string pattern, string replacement)`
- Description: Returns a string from the str string with its substrings that match the pattern regular expression replaced with the replacement string.
- Note: Backslashes (\) are used in a regular expression to escape special characters. If any argument is NULL, the return value is NULL.
- Example:

```
regexp_replace('100-200', '(\\d+)', 'num') = 'num-num'
regexp_replace('2014-03-13', '-', '') = '20140313'
regexp_replace('foobar', 'oo|ar', '') = 'fb'
```

REGEXP_EXTRACT

- Syntax: `string regexp_extract(string str, string pattern, int index)`
- Description: Returns a substring of the str string that matches the pattern regular expression. Multiple substrings may all match the pattern. The index parameter identifies the substring to be returned. The value of the index parameter starts from 1.
- Note: A backslash (\) is used in a regular expression to escape special characters. If any argument is NULL, the return value is NULL.
- Example:

```
regexp_extract('foothebar', 'foo(. *?)( bar)', 2) = 'bar'
regexp_extract('100-200', '(\\d+)-(\\d+)', 1) = '100'
```

INSTR

- Syntax: `int instr(string str, string substr)`
- Description: Returns the position where a substring first appears in a string. The return value starts from 1. If the specified substring does not exist in the string, 0 is returned. If any argument is NULL, the return value is NULL.
- Example:

```
instr('foobar', 'foo') = 1
instr('foobar', 'abc') = 0
```

KEYVALUE

- Syntax: `string keyvalue(string str, string split1, string split2, string key_name)`
- Description: Parses key-value pairs from the specified string and returns the value that is mapped to the specified key. If the key does not exist, the return value is NULL. If any argument is NULL, the return value is NULL.
- Example:

```
keyvalue('k1=v1;k2=v2', ';', '=', 'k2') = 'v2'
keyvalue('k1:v1,k2:v2', ',', ':', 'k3') = NULL
```

HASH

- Syntax: `int hash(string str)`
- Description: Returns the absolute value of the integer returned by the hashCode() function that takes the specified string as an input. If any argument is NULL, the return value is NULL.

MD5

- Syntax: `string md5(string str)`
- Description: Returns the MD5 value of a string. If any argument is NULL or an empty string, the return value is an empty string.

2.11.1.2. CHAR_LENGTH

Syntax

```
CHAR_LENGTH(A)
```

Input parameter

A: VARCHAR data type

Description

Returns the number of characters in a string.

Example

Input example

var1 (VARCHAR)
ss
231ee

Sample code

```
SELECT  CHAR_LENGTH(var1) as aa
FROM T1;
```

Output example

aa (INT)
2
5

2.11.1.3. CHR

Syntax

```
VARCHAR CHR(INT ascii)
```

Arguments

ascii: specifies an integer of the INT type, ranging from 0 to 255. If the argument falls out of this range, null is returned.

Description

Convert ASCII codes into characters.

Examples

Input example

int1(INT)	int2(INT)	int3 (INT)
255	97	65

Sample code

```
SELECT CHR(int1) as var1, CHR(int2) as var2, CHR(int3) as var3
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
ÿ	a	A

2.11.1.4. CONCAT

Syntax

```
VARCHAR concat(VARCHAR var1, VARCHAR var2, ...)
```

Arguments

- var1: specifies a common string of the VARCHAR type.
- var2: specifies a common string of the VARCHAR type.

Description

Concatenate two or more strings into a single string. If any argument is NULL, the argument is skipped.

Examples

Input example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
Hello	My	World
Hello	null	World
null	null	World
null	null	null

Sample code

```
SELECT CONCAT(var1, var2, var3) as var
FROM T1
```

Output example

var (VARCHAR)
HelloMyWorld
HelloWorld
World
""

2.11.1.5. CONCAT_WS

Syntax

```
VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
```

Arguments

- separator: specifies a separator of the VARCHAR type.
- var1: specifies a value of the VARCHAR type.
- var2: specifies a value of the VARCHAR type.

Description

Concatenate each argument's value with a separator to a new string and return the new string, whose length and type depend on the input values.

> Note

- If the separator value is null, it is treated as an empty string for concatenating the argument's values.
- If any argument is null, the argument is skipped during concatenation.

Examples

Input example

sep (VARCHAR)	str1 (VARCHAR)	str2 (VARCHAR)	str3 (VARCHAR)
|	Jack	Harry	John
null	Jack	Harry	John
|	null	Harry	John

sep (VARCHAR)	str1 (VARCHAR)	str2 (VARCHAR)	str3 (VARCHAR)
|	Jack	null	null

Sample code

```
SELECT CONCAT_WS(sep, str1, str2, str3) as var
FROM T1
```

Output example

var (VARCHAR)
Jack|Harry|John
JackHarryJohn
Harry|John
Jack

2.11.1.6. FROM_BASE64

Syntax

```
BINARY FROM_BASE64(str)
```

Arguments

str: specifies a Base64 string of the VARCHAR type.

Description

Parse a Base64 string into binary data.

Examples

Input example

a (INT)	b (BIGINT)	c (VARCHAR)
1	1L	null

Sample code

```
SELECT
from_base64(c) as var1,from_base64('SGVsbG8gd29ybGQ=') as var2
FROM T1
```

Output example

var1 (BINARY)	var2 (BINARY)
null	Byte Array: [72('H'), 101('e'), 108('l'), 108('l'), 111('o'), 32(' '), 119('w'), 111('o'), 114('r'), 108('l'), 100('d')]

2.11.1.7. HASH_CODE

Syntax

```
INT HASH_CODE(VARCHAR str)
```

Arguments

str: VARCHAR type

Description

Return the absolute value of a string on which hashCode() has been executed.

Examples

Input example

str1(VARCHAR)	str2(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	k1:v1,k2:v2	null

Sample code

```
SELECT  HASH_CODE(str1) as var1, HASH_CODE(str2) as var2, HASH_CODE(nullstr) as var3
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
1099348823	401392878	null

2.11.1.8. INITCAP

Syntax

```
 INITCAP(A)
```

Input parameter

A: VARCHAR data type

Description

Returns a string with the first letter of each word in uppercase and all other letters in lowercase.

Example

Input example

var1 (VARCHAR)
aADvbn

Sample code

```
SELECT INITCAP(var1) as aa
FROM T1;
```

Output example

aa (VARCHAR)
Aadvbn

2.11.1.9. JSON_VALUE

Syntax

```
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
```

Arguments

- content: specifies a JSON object to be parsed, which is represented as a JSON string of the VARCHAR type.
- path: specifies a path expression of the VARCHAR type to parse the JSON objects. The following table lists the path expressions that are currently supported.

Path expressions supported

Symbol	Feature
$	Root object
[]	Array subscript
*	Array wildcard
.	Specified objects in a parent object

Description

Extract the value of the specified path from a JSON string. If the JSON string is invalid or any argument is NULL, the return value is NULL.

Examples

Input example

id (INT)	json(VARCHAR)	path(VARCHAR)
1	[10, 20, [30, 40]]	$[2][*]
2	{"aaa": "bbb", "ccc": {"ddd": "eee", "fff": "ggg", " +""hhh": ["h0", "h1", "h2"]}, "iii": "jjj"}	$.ccc.hhh [*]
3	{"aaa": "bbb", "ccc": {"ddd": "eee", "fff": "ggg"," +""hhh": ["h0", "h1", "h2"]}, "iii": "jjj"}	$.ccc.hhh [1]
4	[10, 20, [30, 40]]	NULL
5	NULL	$[2] [*]
6	“{xx]”	“$[2][*]”

Sample code

```
SELECT
    id,
    JSON_VALUE(json, path) AS value
FROM
    T1
```

Output example

id (INT)	value (VARCHAR)
1	[30, 40]
2	["h0", "h1", "h2"]
3	H1
4	NULL
5	NULL
6	NULL

2.11.1.10. KEYVALUE

Syntax

```
VARCHAR KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name)
```

Arguments

- str: specifies a string of key-value pairs. This argument is of the VARCHAR type.
- split1: specifies a delimiter of the VARCHAR type to separate the key-value pairs.
- split2: specifies a delimiter of the VARCHAR type to separate the key and value in a key-value pair.
- key_name: specifies a key value of the VARCHAR type.

Description

Parse the key-value pairs in a string to locate a key-value pair that matches the key-value pair delimiter and key-value delimiter. Then return the key value corresponding to the key name in the key-value pair. If the key name does not exist or an exception occurs, the return value is NULL.

Examples

Input example

str (VARCHAR)	split1 (VARCHAR)	split2 (VARCHAR)	key1 (VARCHAR)
k1=v1; k2=v2	;	=	k2
null	;	|	:
k1:v1|k2:v2	null	=	:
k1:v1|k2:v2	|	=	null
k1:v1|k2:v2	|	=	:
k1:v1|k2:v2	|	=	:

Sample code

```
SELECT  KEYVALUE(str, split1, split2, key1) as result
FROM T1
```

Output example

result (VARCHAR)
v2
null
null
null
null
null

2.11.1.11. LOWER

Syntax

```
VARCHAR LOWER(A)
```

Argument

A: specifies a value of the VARCHAR type.

Description

Convert the specified string to a string of lowercase letters.

Examples

Input example

var1 (VARCHAR)
Ss
yyT

Sample code

```
SELECT LOWER(var1) as aa
FROM T1;
```

Output example

aa (VARCHAR)
ss
yyt

2.11.1.12. LPAD

Syntax

```
VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
```

Arguments

- str: specifies a source string of the VARCHAR type.
- len: specifies the length of a new string. This argument is of the INT type.
- pad: specifies a string to be repeatedly padded to the source string. This argument is of the VARCHAR type.

Description

Left pad a source string with another string several times until the new string reaches the specified length. If any argument is NULL, the return value is NULL. If the source string is an empty string or the target length is a negative number, the return value is NULL. If the string to be padded to the source string is an empty string, there are two scenarios. A trimmed source string is returned when the specified length is equal to or less than the length of the source string. NULL is returned when the specified length is greater than the length of the source string.

Examples

Input example

str (VARCHAR)	len (INT)	pad (VARCHAR)
""	-2	""
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	""
""	2	s
asd	4	""
""	0	""

Sample code

```
SELECT LPAD(str, len, pad) AS result
FROM T1
```

Output example

result (VARCHAR)
null
JohnJHelloWorld
Jo
HelC
null

result (VARCHAR)
null
as
ss
null
""

2.11.1.13. MD5

Syntax

```
VARCHAR MD5(VARCHAR str)
```

Argument

str: specifies a string of the VARCHAR type.

Description

Return the MD5 value of a string. If the argument is NULL, the return value is NULL. If the argument is an empty string, the return value is an empty string.

Examples

Input example

str1 (VARCHAR)	str2 (VARCHAR)
k1 = v1; k2 = v2	null

Sample code

```
SELECT
   MD5(str1) as var1,
   MD5(str2) as var2
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)
19c17f42b4d6a90f7f9ffc2ea9bdd775	NA

2.11.1.14. OVERLAY

Syntax

```
VARCHAR OVERLAY ( (VARCHAR x PLACING VARCHAR y FROM INT start_position [ FOR INT length ])
)
```

Arguments

- x: VARCHAR type
- y: VARCHAR type
- start_position: INT type
- length (optional): INT type

Description

Replace a substring of x with y. The replacement starts from start_position and a total of length+1 characters are to be replaced.

Examples

Sample code

```
OVERLAY('abcdefg' PLACING 'hij' FROM 2 FOR 2) as result
FROM   T1
```

Output example

result(VARCHAR)
ahijdefg

2.11.1.15. PARSE_URL

Syntax

```
VARCHAR PARSE_URL(VARCHAR urlStr, VARCHAR partToExtract [, VARCHAR key])
```

Arguments

- urlStr: specifies a URL string of the VARCHAR type.
- partToExtract: specifies a value of the VARCHAR type obtained after parsing.
- key: specifies an argument name of the VARCHAR type.

Description

Parse a URL to obtain the value of partToExtract, such as QUERY. The options for partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, FILE, AUTHORITY, and USERINFO.

> **Note** If the partToExtract value is NULL, NULL is returned.

Examples

Input example

url1(VARCHAR)	nullstr(VARCHAR)
http://facebook.com/path/p1.php?query=1	null

Sample code

```
SELECT PARSE_URL(url1, 'QUERY', 'query') as var1,
       PARSE_URL(url1, 'QUERY') as var2,
       PARSE_URL(url1, 'HOST') as var3,
       PARSE_URL(url1, 'PATH') as var4,
       PARSE_URL(url1, 'REF') as var5,
       PARSE_URL(url1, 'PROTOCOL') as var6,
       PARSE_URL(url1, 'FILE') as var7,
       PARSE_URL(url1, 'AUTHORITY') as var8,
       PARSE_URL(nullstr, 'QUERY') as var9,
       PARSE_URL(url1, 'USERINFO') as var10,
       PARSE_URL(nullstr, 'QUERY', 'query') as var11
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)	var11(VARCHAR)
1	query=1	facebook.com	/path/p1.php	null	http	/path/p1.php?query=1	facebook.com	null	null	null

2.11.1.16. POSITION

Syntax

```
INTEGER POSITION( x IN y)
```

Arguments

- x: specifies a value of the VARCHAR type.
- y: specifies a value of the VARCHAR type.

Description

Return the position of the first occurrence of string x within string y. The function is similar to LOCATE(substr,str).

Examples

Sample code

```
POSITION('in' IN 'china') as result
FROM   T1
```

Output example

result (INT)
3

2.11.1.17. REGEXP

Syntax

```
BOOLEAN REGEXP(VARCHAR str, VARCHAR pattern)
```

Arguments

- str: specifies a string of the VARCHAR type.
- pattern: specifies a pattern of the VARCHAR type.

Description

Perform regular expression matching on a string to check whether it matches the specified pattern. If the string or pattern is empty or null, FALSE is returned.

Examples

Input example

str1(VARCHAR)	pattern1 (VARCHAR)
k1=v1;k2=v2	k2*
k1:v1|k2:v2	k3
null	k3
k1:v1|k2:v2	null
k1:v1|k2:v2	(

Sample code

```
SELECT  REGEXP(str1, pattern1) as result
FROM T1
```

Output example

result(BOOLEAN)
true
false
null
null
false

2.11.1.18. REGEXP_EXTRACT

Syntax

```
VARCHAR REGEXP_EXTRACT(VARCHAR str, VARCHAR pattern, INT index)
```

Arguments

- str: specifies a source string of the VARCHAR type.
- pattern: specifies the regular expression pattern used to match the string to be extracted. This argument is of the VARCHAR type.
- index: specifies the index number of the substring to be extracted from the source string. This argument is of the INT type.

> **Note** Specify constants in the regular expression with Java format because SQL string constants are completely compiled to Java code. For example, you can specify a digit (0-9) in the regular expression as \d, which is the same as the digit in a regular expression in Java code.

Description

Use the regular expression pattern to match and extract the indexed substring from a source string. The index starts from 1. If any argument is NULL or the regular expression is invalid, the return value is NULL.

Examples

Input example

str1 (VARCHAR)	pattern1 (VARCHAR)	index1 (INT)
foothebar	foo(. *?)(bar)	2
100-200	(\d+)-(\d+)	1
null	foo(. *?)(bar)	2

str1 (VARCHAR)	pattern1 (VARCHAR)	index1 (INT)
foothebar	null	2
foothebar	""	2
foothebar	(	2

Sample code

```
SELECT REGEXP_EXTRACT(str1, pattern1, index1) as result
FROM T1
```

Output example

result (VARCHAR)
bar
100
null
null
null
null

2.11.1.19. REGEXP_REPLACE

Syntax

```
VARCHAR REGEXP_REPLACE(VARCHAR str, VARCHAR pattern, VARCHAR replacement)
```

Arguments

- str: specifies a string of the VARCHAR type.
- pattern: specifies the regular expression pattern used to match the string to be replaced. This argument is of the VARCHAR type.
- replacement: specifies the string used to replace others. This argument is of the VARCHAR type.

> **Note** Specify constants in the regular expression with Java format because SQL string constants are completely compiled to Java code. For example, you can specify a digit (0-9) in the regular expression as \d, which is the same as the digit in a regular expression in Java code.

Description

Replace a substring that matches the regular expression pattern in a source string with a specified string, and return a new string. If any argument is NULL or the regular expression is invalid, the return value is NULL.

Examples

Input example

str1 (VARCHAR)	pattern1 (VARCHAR)	replace1 (VARCHAR)
2014-03-13	-	""
null	-	""
2014-03-13	-	null
2014-03-13	""	s
2014-03-13	(	s
100-200	(\d+)	num

Sample code

```
SELECT REGEXP_REPLACE(str1, pattern1, replace1) as result
FROM T1
```

Output example

result (VARCHAR)
20140313
null
null
2014-03-13
null
num-num

2.11.1.20. REPEAT

Syntax

```
VARCHAR REPEAT(VARCHAR str, INT n)
```

Arguments

- str: specifies the string to be repeated. This argument is of the VARCHAR type.
- n: specifies the number of repetitions. This argument is of the INT type.

Description

Return a new string consisting of N repetitions of the specified string. If any argument is NULL, the return value is NULL. If the value of n is 0 or a negative number, the return value is an empty string.

Examples

Input example

str(VARCHAR)	n(INT)
J	9
Hello	2
Hello	-9
null	9

Sample code

```
SELECT  REPEAT(str,n) as var1
FROM T1
```

Output example

result(VARCHAR)
JJJJJJJJ
HelloHello
“”
null

2.11.1.21. REVERSE

Syntax

```
VARCHAR REVERSE(VARCHAR str)
```

Arguments

str: specifies a string of the VARCHAR type.

Description

Return a string in the reverse order of the specified string. If the argument is NULL, the return value is NULL.

Examples

Input example

str1(VARCHAR)	str2(VARCHAR)	str3 (VARCHAR)	str4 (VARCHAR)
iPhoneX	Alibaba	World	null

Sample code

```
SELECT  REVERSE(str1) as var1 ,REVERSE(str2) as var2,
        REVERSE(str3) as var3,REVERSE(str4) as var4
FROM T1
```

Output example

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)
XenohPi	ababilA	dlroW	null

2.11.1.22. RPAD

Syntax

```
VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
```

Arguments

- str: specifies a source string of the VARCHAR type.
- len: specifies the length of a new string. This argument is of the INT type.
- pad: specifies a string to be repeatedly padded to the source string. This argument is of the VARCHAR type.

Description

Right pad a string with another string several times until the new string reaches the specified length. If any argument is NULL, the return value is NULL. If the source string is an empty string or the specified length is a negative number, the return value is NULL. If the string to be padded to the source string is an empty string, there are two scenarios. A trimmed source string is returned when the specified length is equal to or less than the length of the source string. NULL is returned when the specified length is greater than the length of the source string.

Examples

Input example

str (VARCHAR)	len (INT)	pad (VARCHAR)
""	-2	""
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	""
""	2	s
asd	4	""
""	0	""

Sample code

```
SELECT RPAD(str, len, pad) as result
FROM T1
```

Output example

result (VARCHAR)
null
HelloWorldJohnJ
Jo
CHel
null
null
as
ss
null
""

2.11.1.23. SPLIT_INDEX

Syntax

```
VARCHAR SPLIT_INDEX(VARCHAR str, VARCHAR sep, INT index)
```

Arguments

- str: specifies the string to be split. This argument is of the VARCHAR type.
- sep: specifies the separator of the VARCHAR type.
- index: specifies the index number of the substring to be obtained from the split string. This argument is of the INT type.

Description

Use a separator to split a string into several substrings and return the indexed substring. If no substring is indexed, NULL is returned. The index starts from 0. If any argument is NULL, the return value is NULL.

Examples

Input example

str (VARCHAR)	sep (VARCHAR)	index (INT)
Jack,John,Mary	,	2
Jack,John,Mary	,	3
Jack,John,Mary	null	0
null	,	0

Sample code

```
SELECT SPLIT_INDEX(str, sep, index) as var1
FROM T1
```

Output example

var1 (VARCHAR)
Mary
null
null
null

2.11.1.24. STR_TO_MAP

Syntax

```
MAP STR_TO_MAP(VARCHAR text)
MAP STR_TO_MAP(VARCHAR text, VARCHAR listDelimiter, VARCHAR keyValueDelimiter)
```

Arguments

- text: specifies given text of the VARCHAR type.
- listDelimiter: specifies the delimiter used to splits the given text into key-value pairs. This argument is of the VARCHAR type. The default delimiter is a comma (,).
- keyValueDelimiter: specifies the delimiter used to separate the key and value in a key-value pair. This argument is of the VARCHAR type. The default delimiter is an equal sign (=).

> **Note** The delimiters are defined by Java regular expressions. If a special character is used, it needs to be converted to an escape character.

Description

Use listDelimiter to split the given text into key-value pairs, use keyValueDelimiter to separate the key and value in each key-value pair, and then generate and return a map.

Examples

Sample code

```
SELECT
  STR_TO_MAP('k1 = v1, k2 = v2' ) ['k1'] as a
FROM T1
```

Output example

a (VARCHAR)
v1

2.11.1.25. SUBSTRING

Syntax

```
VARCHAR SUBSTRING(VARCHAR a, INT start)
VARCHAR SUBSTRING(VARCHAR a, INT start, INT len)
```

Arguments

- a: specifies the string from which a substring is to be obtained. This argument is of the VARCHAR type.

- len: specifies the length of the substring to be obtained. This argument is of the INT type.
- start: specifies the start position where the substring is to be obtained from the specified string. This argument is of the INT type.

Description

Obtain a substring of the specified length from a string, starting from the specified position. If the length is not specified, the entire string is obtained. start begins with 1. If the value is 0, it is regarded as 1. If the value is negative, obtain a substring of the specified length backwards starting from the last character in the string.

Examples

Input example

str (VARCHAR)	nullstr (VARCHAR)
k1 = v1; k2 = v2	null

Sample code

```
SELECT SUBSTRING('', 222222222) as var1,
       SUBSTRING(str, 2) as var2,
       SUBSTRING(str, -2) as var3,
       SUBSTRING(str, -2, 1) as var4,
       SUBSTRING(str, 2, 1) as var5,
       SUBSTRING(str, 22) as var6,
       SUBSTRING(str, -22) as var7,
       SUBSTRING(str, 1) as var8,
       SUBSTRING(str, 0) as var9,
       SUBSTRING(nullstr, 0) as var10
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)	var4 (VARCHAR)	var5 (VARCHAR)	var6 (VARCHAR)	var7 (VARCHAR)	var8 (VARCHAR)	var9 (VARCHAR)	var10 (VARCHAR)
""	1 = v1; k2 = v2	v2	v	1	""	""	k1 = v1; k2 = v2	k1 = v1; k2 = v2	null

2.11.1.26. TO_BASE64

Syntax

```
VARCHAR TO_BASE64(bin)
```

Argument

bin: specifies a value of the BINARY type.

Description

Convert the input value of the BINARY type to a Base64 string.

Examples

Input example

c (VARCHAR)
SGVsbG8gd29ybGQ =
SGk =
SGVsbG8 =

Sample code

```
SELECT TO_BASE64(FROM_BASE64(c)) as var1
FROM T1
```

Output example

var1 (VARCHAR)
SGVsbG8gd29ybGQ =
SGk =
SGVsbG8 =

2.11.1.27. TRIM

Syntax

```
VARCHAR TRIM( VARCHAR x )
```

Arguments

x: VARCHAR type

Description

Trim leading or trailing characters from a string. The most common use is to trim leading or trailing blank spaces.

Examples

Test example

```
SELECT TRIM('   Sample   '); as result
FROM   T1
```

Output example

result(VARCHAR)
Sample

2.11.1.28. UPPER

Syntax

```
VARCHAR UPPER(A)
```

Arguments

A: VARCHAR type

Description

Convert the specified string to a string of uppercase letters.

Examples

Input example

var1(VARCHAR)
ss
ttee

Sample code

```
SELECT UPPER(var1) as aa
FROM T1;
```

Output example

aa(VARCHAR)
SS
TTEE

2.11.2. Mathematical unctions

2.11.2.1. Addition

Syntax

```
A + B
```

Arguments

A and B: INT type

Description

Return the result of adding B to A.

Examples

Input example

int1(INT)	int2(INT)	int3(INT)
10	20	30

Sample code

```
SELECT int1+int2+int3 as aa
FROM T1;
```

Output example

aa(INT)
60

2.11.2.2. Subtraction

Syntax

```
A - B
```

Arguments

A and B: specify values of the INT type.

Description

Return the result of subtracting B from A.

Examples

Input example

int1 (INT)	int2 (INT)	int3 (INT)
10	10	30

Sample code

```
SELECT int3 - int2 - int1 as aa
FROM T1;
```

Output example

aa (INT)
10

2.11.2.3. Multiplication

Syntax

```
A * B
```

Arguments

A and B: specify values of the INT type.

Description

Return the result of multiplying A by B.

Examples

Input example

int1 (INT)	int2 (INT)	int3 (INT)
10	20	3

Sample code

```
SELECT int1*int2*int3 as aa
FROM T1;
```

Output example

aa (INT)
600

2.11.2.4. Division

Syntax

```
A/B
```

Arguments

A and B: INT type

Description

Return the result of dividing A by B.

Examples

Input example

int1(INT)	int2 (INT)
8	4

Sample code

```
SELECT int1 / int2 as aa
FROM T1;
```

Output example

aa(INT)
2

2.11.2.5. ABS

Syntax

```
DOUBLE ABS(A)
```

Arguments

A: DOUBLE type

Description

Return the absolute value of the specified number.

Examples

Input example

int1(DOUBLE)
4.3

Sample code

```
SELECT ABS(in1) as aa
FROM T1;
```

Output example

aa(DOUBLE)
4.3

2.11.2.6. ACOS

Syntax

```
DOUBLE ACOS(A)
```

Argument

A: specifies a number of DOUBLE type.

Description

Return the arccos value of the specified number.

Examples

Input example

int1 (DOUBLE)
0.7173560908995228
0.4

Sample code

```
SELECT ACOS(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.7707963267948966

aa (DOUBLE)
1.1592794807274085

2.11.2.7. BIN

Syntax

```
VARCHAR BIN(BIGINT number)
```

Arguments

number: BIGINT type. INT type arguments support implicit conversion, whereas other types of arguments do not.

Description

Convert a value of the BIGINT type to a binary string.

Examples

Input example

id(INT)	x (BIGINT)
1	12L
2	10L
3	0L
4	10000000000L

Sample code

```
SELECT  id, bin(x) as var1
FROM T1
```

Output example

id(INT)	var1(VARCHAR)
1	1100
2	1010
3	0
4	1001010100000010111110010000000000

2.11.2.8. ASIN

Syntax

```
DOUBLE ASIN(A)
```

Arguments

A: DOUBLE type

Description

Return the arcsine value of the specified number.

Examples

Input example

int1 (DOUBLE)
0.7173560908995228
0.4

Sample code

```
SELECT ASIN(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.8
0.41151684606748806

2.11.2.9. ATAN

Syntax

```
DOUBLE ATAN(A)
```

Arguments

A: DOUBLE type

Description

Return the arctangent value of the specified number.

Examples

Input example

int1 (DOUBLE)
0.7173560908995228
0.4

Sample code

```
SSELECT ATAN(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.6222796222326533
0.3805063771123649

2.11.2.10. BITAND

Syntax

```
INT BITAND(INT number1,INT number2)
```

Arguments

- number1: specifies a common argument of the INT type.
- number2: specifies a common argument of the INT type.

Description

Perform a bitwise **AND** operation on the specified values. The input and output values are both of the INT type.

Examples

Input example

a(INT)	b(INT)
2	3

Sample code

```
SELECT  BITAND(a, b) as intt
FROM T1
```

Output example

intt(INT)
2

2.11.2.11. BITNOT

Syntax

```
INT BITNOT((INT number)
```

Arguments

number: specifies a common argument of the INT type.

Description

Perform a bitwise NOT operation on the specified value. The input and output values are both integers.

Examples

Input example

a(INT)
7

Sample code

```
SELECT   BITNOT(a) as var1,
FROM  T1
```

Output example

var1(INT)
0xfff8

2.11.2.12. BITOR

Syntax

```
INT BITOR(INT number1, INT number2)
```

Arguments

- number1: specifies a common argument of the INT type.
- number2: specifies a common argument of the INT type.

Description

Perform a bitwise OR operation on the specified values. The input and output values are both of the INT type.

Examples

Input example

a(INT)	b(INT)
2	3

Sample code

```
SELECT BITOR(a, b) as var1,
FROM  T1
```

Output example

var1(INT)
3

2.11.2.13. BITXOR

Syntax

```
INT BITXOR(INT number1, INT number2)
```

Arguments

- number1: specifies a common value of the INT type.
- number2: specifies a common value of the INT type.

Description

Return the bitwise XOR (exclusive OR) of two given numbers. The argument and result are both of the INT type.

Examples

Input example

a (INT)	b (INT)
2	3

Sample code

```
SELECT BITXOR(a, b) as var1,
FROM  T1
```

Output example

var1 (INT)
1

2.11.2.14. CARDINALITY

Syntax

```
CARDINALITY(str)
```

Arguments

str: specifies an array.

Description

Return the number of elements in an array.

Examples

Input example

str(INT)	b (Array[INT])
1	Array(1, 2, 3)

Sample code

```
SELECT cardinality(b) AS result
FROM T1
```

Output example

result(INT)
3

2.11.2.15. COS

Syntax

```
DOUBLE COS(A)
```

Arguments

A: DOUBLE type

Description

Return the cosine value of the specified number.

Examples

Input example

in1(DOUBLE)
0.8
0.4

Sample code

```
SELECT COS(in1) as aa
FROM T1;
```

Output example

aa(DOUBLE)
0.6967067093471654
0.9210609940028851

2.11.2.16. COT

Syntax

```
DOUBLE COT(A)
```

Arguments

A: specifies a value of the DOUBLE type.

Description

Return the cotangent value of the specified number.

Examples

Input example

in1 (DOUBLE)
0.8
0.4

Sample code

```
SELECT COT(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
1.0296385570503641
0.4227932187381618

2.11.2.17. E

Syntax

```
DOUBLE E()
```

Arguments

DOUBLE type

Description

Return the DOUBLE type value of natural constant E.

Examples

Input example

id(INT)	X(DOUBLE)
1	8.0

Sample code

```
SELECT id, e() as dou1, E() as dou2
FROM T1
```

Output example

id(INT)	dou1(DOUBLE)	dou2 (DOUBLE)
1	2.718281828459045	2.718281828459045

2.11.2.18. EXP

Syntax

```
DOUBLE EXP(A)
```

Argument

A: specifies a value of the DOUBLE type.

Description

Return the power of e, where e is the base of the natural logarithm.

Examples

Input example

in1 (DOUBLE)
8.0
10.0

Sample code

```
SELECT EXP(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
2980.9579870417283
22026.465794806718

2.11.2.19. FLOOR

Syntax

```
DOUBLE FLOOR(A)
```

Arguments

A: DOUBLE type

Description

Return the maximum integer that is less than or equal to the provided number.

Examples

Input example

in1(DOUBLE)
8.123
10.321

Sample code

```
SELECT FLOOR(in1) as aa
FROM T1;
```

Output example

aa(INT)
8
10

2.11.2.20. LN

Syntax

```
DOUBLE ln(DOUBLE number)
```

Arguments

number: specifies a number of the DOUBLE type. If the input value is of the VARCHAR or BIGINT type, it is implicitly converted to the DOUBLE type before an operation.

Description

Return the natural logarithm of the specified number. The return value is a logarithm of the DOUBLE type.

Examples

Input example

ID (INT)	X (DOUBLE)
1	100.0
2	8.0

Sample code

```
SELECT id, ln(x) as dou1, ln(e()) as dou2
FROM  T1
```

Output example

ID (INT)	dou1 (DOUBLE)	dou2 (DOUBLE)
1	4.605170185988092	1.0
2	2.0794415416798357	1.0

2.11.2.21. LOG

Syntax

```
DOUBLE LOG(DOUBLE base, DOUBLE x)
DOUBLE LOG(DOUBLE x)
```

Arguments

- base: specifies a value of the DOUBLE type.
- x: specifies a value of the DOUBLE type.

Description

Return the natural logarithm of x to the base specified by the base argument. The return value is a logarithm of the DOUBLE type. If there is only one argument value, the return value is a natural logarithm to the base specified by the argument.

Examples

Input example

ID (INT)	BASE (DOUBLE)	X (DOUBLE)
1	10.0	100.0
2	2.0	8.0

Sample code

```
SELECT id, LOG(base, x) as dou1, LOG(2) as dou2
FROM  T1
```

Output example

ID (INT)	dou1 (DOUBLE)	dou2 (DOUBLE)
1	2.0	0.6931471805599453
2	3.0	0.6931471805599453

2.11.2.22. LOG10

Syntax

```
DOUBLE LOG10(DOUBLE x)
```

Argument

x: specifies a value of the DOUBLE type.

Description

Return the base-10 logarithm of x. If x is NULL, the return value is NULL. If x is a negative number, an exception occurs.

Examples

Input example

id (INT)	X (INT)
1	100
2	10

Sample code

```
SELECT id, log10(x) as dou1
FROM T1
```

Output example

ID (INT)	dou1 (DOUBLE)
1	2.0
2	1.0

2.11.2.23. LOG2

Syntax

```
DOUBLE LOG2(DOUBLE x)
```

Arguments

x: specifies a multiple of 2. This argument is of the DOUBLE type.

Description

Return the base-2 logarithm of x. If x is NULL, the return value is NULL. If x is a negative number, an exception occurs.

Examples

Input example

id (INT)	X (INT)
1	8
2	2

Sample code

```
SELECT id, log2(x) as dou1
FROM T1
```

Output example

ID (INT)	dou1 (DOUBLE)
1	3.0
2	1.0

2.11.2.24. PI

Syntax

```
DOUBLE PI( )
```

Description

Obtain the ratio of circumference to diameter, namely, Pi.

Examples

Input example

id (INT)	X (INT)
1	8

Sample code

```
SELECT id, PI() as dou1
FROM T1
```

Output example

ID (INT)	dou1 (DOUBLE)
1	3.141592653589793

2.11.2.25. POWER

Syntax

```
DOUBLE POWER(A, B)
```

Input parameters

A and B: DOUBLE data type

Description

Returns the result of raising A to the power of B.

Example

Input example

int1 (DOUBLE)	in2 (DOUBLE)
2.0	4.0

Sample code

```
SELECT POWER(in1, in2) as aa
FROM T1;
```

Output example

aa (DOUBLE)
16.0

2.11.2.26. RAND

Syntax

```
DOUBLE RAND([BIGINT seed])
```

Arguments

seed: specifies a random number of the BIGINT type. This argument specifies the start number of a random number sequence.

Description

Return a random number of the DOUBLE type. The return value range is `[0, 1)`.

Examples

Input example

id(INT)	X(INT)
1	8

Sample code

```
SELECT id, rand(1) as dou1, rand(3) as dou2
FROM T1
```

Output example

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	0.7308781907032909	0.731057369148862

2.11.2.27. SIN

Syntax

```
DOUBLE SIN(A)
```

Arguments

A: specifies a number of the DOUBLE type.

Description

Return the sine value of the specified number.

Examples

Input example

in1 (DOUBLE)
8.0
0.4

Sample code

```
SELECT SIN(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
0.9893582466233818
0.3894183423086505

2.11.2.28. SQRT

Syntax

```
DOUBLE SQRT(A)
```

Arguments

A: specifies a number of the DOUBLE type.

Description

Return the square root of the specified number.

Examples

Input example

in1 (DOUBLE)
8.0

Sample code

```
SELECT SQRT(in1) as aa
FROM T1;
```

Output example

aa (DOUBLE)
2.8284271247461903

2.11.2.29. TAN

Syntax

```
DOUBLE TAN(A)
```

Arguments

A: DOUBLE type

Description

Return the tangent value of the specified number.

Examples

Input example

in1(DOUBLE)
8.0
0.4

Sample code

```
SELECT TAN(in1) as aa
FROM T1;
```

Output example

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

2.11.2.30. CEIL

Syntax

```
value_class CEIL(value)
```

Arguments

value: specifies a value of the INT, DOUBLE, or BIGINT type.

Description

Return the smallest integer greater than or equal to a specified value. The value_class data type returned by CEIL is the same as that of the argument.

Examples

Input example

ID (INT)	value1 (INT)	value2 (DOUBLE)
1	3	3.6

Sample code

```
SELECT CEIL (value2) as result1
SELECT CEIL (value2) as result2
FROM   T1
```

Output example

ID (INT)	result1 (INT)	result2 (DOUBLE)
1	3	4.0

2.11.2.31. CHARACTER_LENGTH

Syntax

```
INTEGER CHARACTER_LENGTH( VARCHAR x )
```

Arguments

x: specifies a value of the VARCHAR type.

Description

Return the number of characters contained in a string.

Examples

Input example

ID (INT)	X (VARCHAR)
1	StreamCompute

Sample code

```
SELECT CHARACTER_LENGTH( x ) as result
FROM   T1
```

Output example

ID (INT)	result (INT)
1	13

2.11.2.32. DEGREES

Syntax

```
DOUBLE DEGREES(double x)
```

Arguments

x: specifies a radian value of the DOUBLE type.

Description

Return the result of converting radians to degrees.

Examples

Sample code

```
SELECT DEGREES( PI() ) as result
FROM   T1
```

Output example

result(DOUBLE)
180.0

2.11.2.33. MOD

Syntax

```
Integer MOD(Integer x,Integer y)
```

Arguments

- x: specifies a value of the Integer type.
- y: specifies a value of the Integer type.

Description

Return the remainder when integer x is divided by integer y, without considering the quotient. When x is a negative integer or both x and y are negative integers, the result is negative.

Examples

Input example

X (INT)	Y (INT)
29	3
-29	3
-29	-3

Sample code

```
SELECT MOD(x,y) as result
FROM   T1
```

Output example

ID (INT)	result (INT)
1	2
2	-2
3	-2

2.11.2.34. ROUND

Syntax

```
DECIMAL ROUND(DECIMAL x, INT n)
```

Argument

x: DECIMAL type

Description

Round the x argument's value to n decimal places.

Examples

Input example

in1 (DECIMAL)
0.7173560908995228

in1 (DECIMAL)
0.4

Sample code

```
SELECT ROUND(in1,2) as `result`
FROM T1;
```

Output example

result (DECIMAL)
0.72
0.40

2.11.3. Date functions

2.11.3.1. CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP
```

Arguments

None

Description

Return the current UTC (GMT+0) timestamp, which is eight hours earlier than Beijing time.

Examples

Sample code

```
SELECT CURRENT_TIMESTAMP as var1
FROM T1
```

Output example

var1 (TIMESTAMP)
2007-04-30 13:10:02.047

2.11.3.2. DATEDIFF

Syntax

Syntax

```
INT DATEDIFF(VARCHAR enddate, VARCHAR startdate)
INT DATEDIFF(TIMESTAMP enddate, VARCHAR startdate)
INT DATEDIFF(VARCHAR enddate, TIMESTAMP startdate)
INT DATEDIFF(TIMESTAMP enddate, TIMESTAMP startdate)
```

Arguments

- startdate: specifies a start date of the TIMESTAMP or VARCHAR type. The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
- enddate: specifies an end date of the TIMESTAMP or VARCHAR type. The format of a VARCHAR type date is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.

Description

Compute the number of days between the specified start date and end date. The return value is an integer of the TIMESTAMP type or in the format of `yy-MM-dd HH:mm:ss` or `yy-MM-dd`. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

datetime1(VARCHAR)	datetime2 (VARCHAR)	nullstr(VARCHAR)
2017-10-15 00:00:00	2017-09-15 00:00:00	null

Sample code

```
SELECT  DATEDIFF(datetime1, datetime2) as int1,
        DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',datetime2) as int2,
        DATEDIFF(datetime2,TIMESTAMP '2017-10-15 23:00:00') as int3,
        DATEDIFF(datetime2,nullstr) as int4,
        DATEDIFF(nullstr,TIMESTAMP '2017-10-15 23:00:00') as int5,
        DATEDIFF(nullstr,datetime2) as int6,
        DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',TIMESTAMP '2017-9-15 00:00:00')as int7
FROM T1
```

Output example

int1(INT)	int2(INT)	int3(INT)	int4(INT)	int5(INT)	int6(INT)	int7 (INT)
30	31	-31	null	null	null	31

2.11.3.3. DATE_ADD

Syntax

```
VARCHAR DATE_ADD(VARCHAR startdate, INT days)
VARCHAR DATE_ADD(TIMESTAMP time, INT days)
```

Arguments

- startdate: specifies a date of the VARCHAR type. The format is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
- time: specifies a date of the TIMESTAMP type.
- days: specifies the number of days of the INT type.

Description

Return a date that is X (specified by days) days after the specified date of the VARCHAR type. The date format can be yyyy-MM-dd hh:mm:ss, yyyy-MM-dd, and timestamp string. The return value is a string type date in the format of yyyy-MM-dd. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

datetime1 (VARCHAR)	nullstr (VARCHAR)
2017-09-15 00:00:00	null

Sample code

```
SELECT DATE_ADD(datetime1, 30) as var1,
        DATE_ADD(TIMESTAMP '2017-09-15 23:00:00',30) as var2,
        DATE_ADD(nullstr,30) as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
2017-10-15	2017-10-15	null

2.11.3.4. DATE_FORMAT

Syntax

```
VARCHAR DATE_FORMAT(TIMESTAMP time, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR from_format, VARCHAR to_format)
```

Input parameters

- date: specifies a date of the VARCHAR data type. The default format is yyyy-MM-dd HH:mm:ss.
- time: specifies a date of the TIMESTAMP data type.
- from_format: specifies a source date format.
- to_format: specifies a target date format.

Description

Converts a specified date from a source format to a target format. The date or time parameter specifies a date. The from_format parameter is optional and specifies the source date format. The default format is yyyy-MM-dd hh:mm:ss. The to_format parameter specifies the target date format. The return value is a date in the specified target format. If any argument is NULL or a parsing error occurs, the return value is NULL.

Example

Input example

date1 (VARCHAR)	datetime1 (VARCHAR)	nullstr (VARCHAR)
0915-2017	2017-09-15 00:00:00	null

Sample code

```
SELECT  DATE_FORMAT(datetime1, 'yyMMdd') as var1,
        DATE_FORMAT(nullstr, 'yyMMdd') as var2,
        DATE_FORMAT(datetime1, nullstr) as var3,
        DATE_FORMAT(date1, 'MMdd-yyyy', nullstr) as var4,
        DATE_FORMAT(date1, 'MMdd-yyyy', 'yyyyMMdd') as var5,
        DATE_FORMAT(TIMESTAMP '2017-09-15 23:00:00', 'yyMMdd') as var6
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)	var4 (VARCHAR)	var5 (VARCHAR)	var6 (VARCHAR)
170915	null	null	null	20170915	170915

2.11.3.5. DATE_SUB

Syntax

```
VARCHAR DATE_SUB(VARCHAR startdate, INT days)
VARCHAR DATE_SUB(TIMESTAMP time, INT days)
```

Arguments

- startdate: specifies a date of the VARCHAR type. The format is yyyy-MM-dd or yyyy-MM-dd HH:mm:ss.
- time: specifies a date of the TIMESTAMP type.
- days: specifies the number of days. This argument is of the INT type.

Description

Return the date that is X (specified by days) days before the specified date. The return value is a VARCHAR type date in the format of yyyy-MM-dd. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

date1 (VARCHAR)	nullstr (VARCHAR)
2017-10-15	null

Sample code

```
SELECT  DATE_SUB(date1, 30) as var1,
        DATE_SUB(TIMESTAMP '2017-10-15 23:00:00',30) as var2,
        DATE_SUB(nullstr,30) as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
2017-09-15	2017-09-15	null

2.11.3.6. DAYOFMONTH

Syntax

```
INT DAYOFMONTH(TIMESTAMP time)
INT DAYOFMONTH(DATE date)
```

Arguments

- date: specifies a date of the DATE type.
- time: specifies a date of the TIMESTAMP type.

Description

Return the day number of the specified timestamp or date. Valid range: [1,31].

Examples

Input example

tsStr (VARCHAR)	dateStr (VARCHAR)	tdate (DATE)	ts (TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT DAYOFMONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
       DAYOFMONTH(DATE '2017-09-22') as int2,
       DAYOFMONTH(tdate) as int3,
       DAYOFMONTH(ts) as int4,
       DAYOFMONTH(CAST(dateStr AS DATE)) as int5,
       DAYOFMONTH(CAST(tsStr AS TIMESTAMP))  as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
15	22	10	15	15	15

2.11.3.7. FROM_UNIXTIME

Syntax

```
VARCHAR FROM_UNIXTIME(BIGINT unixtime[, VARCHAR format])
```

Arguments

- unixtime: specifies a Unix timestamp (in seconds) of the BIGINT type. An exception occurs if this argument is of another type.
- format: specifies the target date format of the VARCHAR type. The default format is yyyy-MM-dd hh:mm:ss. This argument is optional.

Description

Return a VARCHAR type date in the specified date format. The default format is yyyy-MM-dd HH:mm:ss. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

unixtime1(INT)	nullstr(VARCHAR)
1505404800	null

Sample code

```
SELECT FROM_UNIXTIME(unixtime1) as var1,
        FROM_UNIXTIME(unixtime1,'MMdd-yyyy') as var2,
        FROM_UNIXTIME(unixtime1,nullstr) as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
2017-09-15 00:00:00	0915-2017	null

2.11.3.8. HOUR

Syntax

```
 INT HOUR(TIMESTAMP timestamp)
INT HOUR(TIME time)
```

Arguments

- time: specifies a time value of the TIME type.
- time: specifies a time value of the TIMESTAMP type.

Description

Return the hours (in 24-hour format) from the specified time or timestamp string. The return value ranges from 0 to 23.

Examples

Input example

datetime1 (VARCHAR)	time1 (VARCHAR)	time2 (TIME)	timestamp1 (TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

Sample code

```
SELECT HOUR(TIMESTAMP '2016-09-20 23:33:33') as int1,
       HOUR(TIME '23:30:33') as int2,
       HOUR(time2) as int3,
       HOUR(timestamp1) as int4,
       HOUR(CAST(time1 AS TIME)) as int5,
       HOUR(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
23	23	22	11	22	11

2.11.3.9. LOCALTIME

Syntax

```
TIME LOCALTIME
```

Arguments

The function is a non-parameterized constructor and can be used directly as a variable.

Description

Return the current time of the TIME type in the session time zone.

Examples

Sample code

```
SELECT  LOCALTIME as `result`
FROM T1
```

Output example

result(TIME)
19:00:47

2.11.3.10. MINUTE

Syntax

```
INT MINUTE(TIMESTAMP timestamp)
 INT MINUTE(TIME time)
```

Arguments

- time: specifies a time value of the TIME type.
- time: specifies a time value of the TIMESTAMP type.

Description

Return the minutes from the specified time value. Valid range: [0, 59].

Examples

Input example

datetime1 (VARCHAR)	time1 (VARCHAR)	time2 (TIME)	timestamp1 (TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

Sample code

```
SELECT MINUTE(TIMESTAMP '2016-09-20 23:33:33') as int1,
       MINUTE(TIME '23:30:33') as int2,
       MINUTE(time2) as int3,
       MINUTE(timestamp1) as int4,
       MINUTE(CAST(time1 AS TIME)) as int5,
       MINUTE(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
33	30	23	12	23	12

2.11.3.11. MONTH

Syntax

```
INT MONTH(TIMESTAMP time)
 INT MONTH(DATE date)
```

Arguments

- time: specifies a time value of the TIMESTAMP type.
- date: specifies a time value of the TIME type.

Description

Return the month number of the specified time or date. Valid range: [1, 12].

Examples

Input example

tsStr (VARCHAR)	dateStr (VARCHAR)	tdate (DATE)	ts (TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT MONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
       MONTH(DATE '2017-09-22') as int2,
       MONTH(tdate) as int3,
       MONTH(ts) as int4,
       MONTH(CAST(dateStr AS DATE)) as int5,
       MONTH(CAST(tsStr AS TIMESTAMP))  as int6
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)	int5 (INT)	int6 (INT)
9	9	11	10	9	10

2.11.3.12. NOW

Syntax

```
BIGINT NOW()
```

Arguments

If no argument is specified, the Unix timestamp (in seconds) of the current system time is returned.

Description

Return the Unix timestamp (in seconds) of the current time zone. You can also specify an INT type argument as an offset (in seconds) and use UNIX_TIMESTAMP(CAST(NOW AS VARCHAR)) to convert the INT type argument to a timestamp value. Then add the offset to the current timestamp to return a value.

Examples

Input example

b(VARCHAR)
null

Sample code

```
SELECT  NOW() as big1, NOW(b) as big2
FROM T1
```

Output example

big1(BIGINT)	big2 (BIGINT)
1403006911	null

2.11.3.13. SECOND

Syntax

```
INT SECOND(TIMESTAMP timestamp)
 INT SECOND(TIME time)
```

Arguments

- time: specifies a time value of the TIME type.
- time: specifies a time value of the TIMESTAMP type.

Description

Extract the seconds from the specified time value. The return value ranges from 0 to 59.

Examples

Input example

datetime1(VARCHAR)	time1 (VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

Sample code

```
SELECT SECOND(TIMESTAMP '2016-09-20 23:33:33') as int1,
       SECOND(TIME '23:30:33') as int2,
       SECOND(time2) as int3,
       SECOND(timestamp1) as int4,
       SECOND(CAST(time1 AS TIME)) as int5,
       SECOND(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1
```

Output example

int1(INT)	int2(INT)	int3(INT)	int4 (INT)	int5 (INT)	int6 (INT)
33	33	24	13	24	13

2.11.3.14. TIMESTAMPADD

Syntax

```
TIMESTAMP TIMESTAMPADD(interval,INT int_expr,TIMESTAMP datetime_expr)
DATE TIMESTAMPADD(interval,INT int_expr,DATE datetime_expr)
```

Arguments

- interval

FRAC_SECOND	Specifies an interval in milliseconds.
SECOND	Specifies an interval in seconds.
MINUTE	Specifies an interval in minutes.
HOUR	Specifies an interval in hours.

DAY	Specifies an interval in days.
WEEK	Specifies an interval in weeks.
MONTH	Specifies an interval in months.
QUARTER	Specifies an interval in quarters.
YEAR	Specifies an interval in years.

- int_expr: specifies an interval that is added to the specified timestamp or date. The value must be an integer.
- datetime_expr: specifies a timestamp or date to which the interval is added.

Description

The data type of the result returned by the TIMESTAMPADD function is the same as that of the datetime_expr argument. The TIMESTAMPADD function adds an interval (int_expr) to a timestamp or date (datetime_expr) and returns the updated time.

Examples

Input example

a (TIMESTAMP)	b (DATE)
2018-07-09 10:23:56	1990-02-20

Sample code

```
SELECT
TIMESTAMPADD(HOUR,3,a) AS `result1`,
TIMESTAMPADD(DAY,3,b) AS `result2`
FROM T1
```

Output example

result1 (TIMESTAMP)	result2 (DATE)
2018-07-09 13:23:56	1990-02-23

2.11.3.15. TO_DATE

Syntax

```
Date TO_DATE(INT time)
Date TO_DATE(VARCHAR date)
Date TO_DATE(VARCHAR date, VARCHAR format)
```

Input parameters

- time: specifies the number of days from January 1, 1970 to a specified date. This parameter is of the INT data type.
- date: specifies a date of the VARCHAR data type. The default format is yyyy-MM-dd.
- format: specifies a date format of the VARCHAR data type.

Description

Converts a date of the INT or VARCHAR data type into a date of the DATE data type.

Example

Input example

date1 (bigint)	date2 (VARCHAR)	date3 (VARCHAR)
100	2017-09-15	20170915

Sample code

```
SELECT  TO_DATE(date1) as var1,
        TO_DATE(date2) as var2,
        TO_DATE(date3, "yyyyMMdd") as var3
FROM T1
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)	var3 (VARCHAR)
1970-04-11	2017-09-15	2017-09-15

2.11.3.16. TO_TIMESTAMP

Syntax

```
TIMESTAMP TO_TIMESTAMP(BIGINT time)
TIMESTAMP TO_TIMESTAMP(VARCHAR date)
TIMESTAMP TO_TIMESTAMP(VARCHAR date, VARCHAR format)
```

Arguments

- time: specifies a time value of the BIGINT type. The unit is milliseconds.
- date: specifies a date of the VARCHAR type. The default format is yyyy-MM-dd HH:mm:ss[. SSS].
- format: specifies the source date format of the VARCHAR type.

Description

Convert a date of the BIGINT or VARCHAR type to a date of the TIMESTAMP type.

Examples

Input example

timestamp1 (bigint)	timestamp2 (VARCHAR)	timestamp3 (VARCHAR)
1513135677000	2017-09-15 00:00:00	20170915000000

Sample code

```
SELECT TO_TIMESTAMP(timestamp1) as var1,
        TO_TIMESTAMP(timestamp2) as var2,
        TO_TIMESTAMP(timestamp3, "yyyyMMddHHmmss") as var3
FROM T1
```

Output example

var1 (TIMESTAMP)	var2 (TIMESTAMP)	var3 (TIMESTAMP)
2017-12-13 03:27:57.0	2017-09-15 00:00:00.0	2017-09-15 00:00:00.0

2.11.3.17. UNIX_TIMESTAMP

Syntax

```
BIGINT UNIX_TIMESTAMP()
BIGINT UNIX_TIMESTAMP(VARCHAR date)
BIGINT UNIX_TIMESTAMP(VARCHAR date, VARCHAR format)
```

Arguments

- date: specifies a date of the VARCHAR type. The default format is yyyy-MM-dd HH:mm:ss.
- format: specifies the source date format of the VARCHAR type. The default format is yyyy-MM-dd HH:mm:ss.

Description

Convert the specified date to a Unix timestamp (in seconds) of the BIGINT type. Both the date and format parameters are optional. If neither is specified, the Unix timestamp (in seconds) of the current time is returned. If any argument is NULL or a parsing error occurs, the return value is NULL.

Examples

Input example

nullstr (VARCHAR)
null

Sample code

```
SELECT  UNIX_TIMESTAMP() as big1,
        UNIX_TIMESTAMP(nullstr) as big2
FROM T1
```

Output example

big1 (BIGINT)	big2 (BIGINT)
1403006911	null

2.11.3.18. WEEK

Syntax

```
INT WEEK(DATE date)
INT WEEK(TIMESTAMP time)
```

Arguments

- date: specifies a date of the DATE type.
- time: specifies a date of the TIMESTAMP type.

Description

Calculate the week number of the specified date in the year. The valid week number range is from 1 to 53.

Examples

Input example

dateStr(VARCHAR)	date1 (DATE)	ts1 (TIMESTAMP)
2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT WEEK(TIMESTAMP '2017-09-15 00:00:00') as int1,
        WEEK(date1) as int2, WEEK(ts1) as int3, WEEK(CAST(dateStr AS DATE)) as int4
FROM T1
```

Output example

int1 (INT)	int2 (INT)	int3 (INT)	int4 (INT)
37	45	41	37

2.11.3.19. YEAR

Syntax

```
INT YEAR(TIMESTAMP time)
INT YEAR(DATE date)
```

Arguments

- date: specifies a date of the DATE type.
- time: specifies a date of the TIMESTAMP type.

Description

Extract the year from the specified date.

Examples

Input example

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate (DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

Sample code

```
SELECT YEAR(TIMESTAMP '2016-09-15 00:00:00') as int1,
       YEAR(DATE '2017-09-22') as int2,
       YEAR(tdate) as int3,
       YEAR(ts) as int4,
       YEAR(CAST(dateStr AS DATE)) as int5,
       YEAR(CAST(tsStr AS TIMESTAMP))  as int6
FROM T1
```

Output example

int1 (BIGINT)	int2 (BIGINT)	int3 (BIGINT)	int4 (BIGINT)	int5 (BIGINT)	int6 (BIGINT)
2016	2017	2017	2017	2015	2017

2.11.4. Conditional functions

2.11.4.1. CASE WHEN

Syntax

```
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
```

Input parameters

Parameter	Data type
a	BOOLEAN
b	BOOLEAN
c	BOOLEAN
d	BOOLEAN
e	BOOLEAN

Description

- If a is true, b is returned.
- If a is false and c is true, d is returned.
- If both a and c are false, e is returned.

Example

> **Note**
>
> If the CASE WHEN function returns a constant string, spaces are added after the string. An example is described as follows. When the return value is "ios", several spaces are added after this string.
>
> ```
> case when device_type = 'android'
> then 'android'
> else 'ios'
> end as os
> ```
>
> You can resolve the issue in the following two ways:
>
> - Use the trim() method to remove the spaces. As shown in the following example, you can change os to trim(os) in the statement.
> - Use the cast() method to convert the constant string into the VARCHAR data type.

- Input example

device_type (VARCHAR)
android
ios
win

- Sample code 1: using the trim() method

```
SELECT
   trim(os), // The trim() method is used.
   CHAR_LENGTH(trim(os)) // The trim() method is used.
from(
   SELECT
     case when device_type = 'android'
     then 'android'
     else 'ios'
end as os
FROM T1
);
```

Sample code 2: using the cast() method

```
SELECT
 os,
CHAR_LENGTH(os)
from
(SELECT
 case when device_type = 'android'
 then cast('android' as varchar) // The cast() method is used.
 else cast('ios' as varchar) // The cast() method is used.
end as os
FROM T1
);
```

- Output example

os (VARCHAR)	length (INT)
android	7
ios	3
ios	3

2.11.4.2. COALESCE

Syntax

```
COALESCE(A,B,...)
```

Arguments

A and B: specify values to be checked. NULL values are allowed in the list of values. Non-NULL values in the list must be of the same type. Otherwise, an exception occurs. The return type is the same as the argument type.

Description

Return the first non-NULL value in the specified list. If all values in the list are NULL, the return value is NULL.

> **Note** The list must contain at least one argument. Otherwise, an exception occurs.

Examples

Input example

var1(VARCHAR)	var2(VARCHAR)
null	30

Sample code

```
SELECT COALESCE(var1,var2) as aa
FROM T1;
```

Output example

aa(VARCHAR)
30

2.11.4.3. IF

Syntax

```
T IF(BOOLEAN testCondition, T valueTrue, T valueFalseOrNull)
```

> **Note** T indicates any return type.

Arguments

- testCondition: BOOLEAN type
- valueTrue and valueFalseOrNull: These two arguments must be of the same type. The specific type is not restricted.

Description

Use the BOOLEAN value of testCondition as the criterion. If the value is true, valueTrue is returned. If the value is false, valueFalseOrNull is returned. If testCondition is NULL, valueFalseOrNull is returned. If any other argument is NULL, the operation is performed based on normal semantics, and the return value is of the T type.

Examples

Input example

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	Jack	Harry
1	2	Jack	null
1	2	null	Harry

Sample code

```
SELECT IF(int1 < int2,str1, str2) as int1
FROM T1
```

Output example

int1(VARCHAR)
Jack
Jack
null

2.11.4.4. IS_ALPHA

Syntax

```
BOOLEAN IS_ALPHA(VARCHAR str)
```

Arguments

str: specifies a string of the VARCHAR type.

Description

Check whether the specified string contains only letters. If it contains only letters, TRUE is returned. Otherwise, FALSE is returned.

Examples

Input example

e (VARCHAR)	f (VARCHAR)	g (VARCHAR)
3	asd	null

Sample code

```
SELECT  IS_ALPHA(e) as boo1,IS_ALPHA(f) as boo2,IS_ALPHA(g) as boo3
FROM T1
```

Output example

boo1 (BOOLEAN)	boo2 (BOOLEAN)	boo3 (BOOLEAN)
false	Yes	false

2.11.4.5. IS_DECIMAL

Syntax

```
BOOLEAN IS_DECIMAL(VARCHAR str)
```

Input parameter

str: specifies a string of the VARCHAR data type.

Description

If a specified string contains only decimal characters, true is returned. Otherwise, false is returned.

Example

Input example

a (VARCHAR)	b (VARCHAR)	c (VARCHAR)	d (VARCHAR)	e (VARCHAR)	f (VARCHAR)	g (VARCHAR)
1	123	2	11.4445	3	asd	null

Sample code

```
SELECT IS_DECIMAL(a) as boo1,IS_DECIMAL(b) as boo2,IS_DECIMAL(c) as boo3,
       IS_DECIMAL(d) as boo4,IS_DECIMAL(e) as boo5,IS_DECIMAL(f) as boo6,IS_DECIMAL(g) as b
oo7
FROM T1
```

Output example

boo1 (BOOLEAN)	boo2 (BOOLEAN)	boo3 (BOOLEAN)	boo4 (BOOLEAN)	boo5 (BOOLEAN)	boo6 (BOOLEAN)	boo7 (BOOLEAN)
true	true	true	true	true	false	false

2.11.4.6. IS_DIGIT

Syntax

```
BOOLEAN IS_DIGIT(VARCHAR str)
```

Arguments

str: specifies a string of the VARCHAR type.

Description

Check whether the specified string contains only digits. If it only contains digits, TRUE is returned. Otherwise, FALSE is returned. The return value is of the BOOLEAN type.

Examples

Input example

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

Sample code

```
SELECT  IS_DIGIT(e) as boo1,IS_DIGIT(f) as boo2,IS_DIGIT(g) as boo3
FROM T1
```

Output example

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
true	false	false

2.11.4.7. NULLIF

Syntax

```
NULLIF(A,B)
```

Arguments

A and B: INT type

Description

If A and B have the same value, null is returned. If A and B have different values, the value of the first argument is returned.

Examples

Input example

var1(INT)	var2(INT)
30	30

Sample code

```
SELECT NULLIF(var1,var2) as aa
FROM T1;
```

Output example

aa(VARCHAR)
null

2.11.5. Table-valued functions

2.11.5.1. GENERATE_SERIES

Syntax

```
GENERATE_SERIES(INT from, INT to)
```

Arguments

- from: specifies a lower limit of the INT type.
- to: specifies an upper limit of the INT type.

Description

Generate a series of continuous values from the lower limit to the upper limit minus one.

Examples

Input example

s(INT)	e(INT)
1	3
-2	1

Sample code

```
SELECT s, e, v
FROM T1, lateral table(GENERATE_SERIES(s, e))
as T(v)
```

Output example

s(INT)	e(INT)	v(INT)
1	3	1

s(INT)	e(INT)	v(INT)
1	3	2
-2	1	-2
-2	1	-1
-2	1	-0

2.11.5.2. JSON_TUPLE

Syntax

```
JSON_TUPLE(str, path1, path2 ..., pathN)
```

Arguments

- str: specifies a JSON string.
- path1 to pathN: specify strings that represent paths, which do not need to begin with a dollar sign ($).

Description

Obtain the value represented by each path string from a JSON string.

Examples

Input example

d(VARCHAR)	s(VARCHAR)
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	qwe3
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	qwe2

Sample code

```
SELECT d, v
FROM T1, lateral table(JSON_TUPLE(d, 'qwe', s))
as T(v)
```

Output example

d(VARCHAR)	v(VARCHAR)
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	asd

d(VARCHAR)	v(VARCHAR)
{ “qwe” :” asd” ,” qwe2” :” asd2” ,” qwe3” :” asd3” }	asd3
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	asd4
{ “qwe” :” asd4” ,” qwe2” :” asd5” ,” qwe3” :” asd3” }	asd5

2.11.5.3. STRING_SPLIT

Syntax

```
STRING_SPLIT(VARCHAR , separator)
```

Arguments

separator: specifies a separator of the VARCHAR type.

Description

Use a separator to split a string into multiple substrings.

Examples

Input example

d (VARCHAR)	s (VARCHAR)
abc-bcd	-
hhh	-

Sample code

```
SELECT d, v
FROM T1, lateral table(STRING_SPLIT(d, s))
as T(v)
```

Output example

d (VARCHAR)	v (VARCHAR)
abc-bcd	abc
abc-bcd	bcd
hhh	hhh

2.11.6. Type conversion functions

2.11.6.1. CAST

Syntax

```
CAST(A AS type)
```

Argument

A: For more information about the argument type, see Data types.

Description

Convert a value to a specified type.

Examples

Input example

var1 (VARCHAR)	var2 (INT)
1000	30

Sample code

```
SELECT CAST(var1 AS INT)  as aa
FROM T1;
```

Output example

aa (INT)
1000

2.11.7. Aggregate functions

2.11.7.1. AVG

Syntax

```
AVG(A)
```

Argument

A: specifies a value of numeric types including TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE.

Description

Return the average number.

Examples

Input example

var1 (INT)	var2 (INT)
4	30
6	30

Sample code

```
SELECT AVG(var1)  as aa
FROM T1;
```

Output example

aa (INT)
5

2.11.7.2. CONCAT_AGG

Syntax

```
CONCAT_AGG([linedelimiter, ] value)
```

Argument

linedelimiter: specifies a connector, which currently can only be a string constant.

Description

Use a connector to concatenate multiple fields. The default connector is "\n". The return value is a new concatenated string of the VARCHAR type.

Examples

Input example

c (VARCHAR)
Hi
Hi
Hi

c (VARCHAR)
Hi
Hi
Hi
Hi
Hi
Hi
Hi

Sample code

```
SELECT concat_agg(c) as var1,
       concat_agg('-', c) as var2
FROM MyTable
GROUP BY c
```

Output example

var1 (VARCHAR)	var2 (VARCHAR)
Hi\nHi\nHi\nHi\nHi\nHi\nHi\nHi	Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi-Hi

2.11.7.3. COUNT

Syntax

```
COUNT(A)
```

Arguments

A

- Supported data types include numeric types (TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE), BOOLEAN, and VARCHAR.
- The DATE, TIME, TIMESTAMP, and VARBINARY data types are not supported.

Description

Return the number of input parameters.

Examples

Input example

var1 (VARCHAR)
1000
100
10
1

Sample code

```
SELECT COUNT(var1)  as aa
FROM T1;
```

Output example

aa (BIGINT)
4

2.11.7.4. FIRST_VALUE

Syntax

```
T FIRST_VALUE(value: T)
```

To obtain the first record based on parameters, define the function as follows:

```
T FIRST_VALUE(value: T, order: Long)
```

Arguments

- value: specifies a value of any type. Only one argument is supported.
- order: specifies a record with the smallest order value as the first record.

Description

Obtain the first non-null record of a stream.

Examples

Input example

a (BIGINT)	b (INT)	c (VARCHAR)
1L	1	"Hello"
2L	2	"Hello"

a (BIGINT)	b (INT)	c (VARCHAR)
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"
20L	20	"Hello World"

Sample code

```
SELECT
c,
first_value(b) OVER (PARTITION BY c ORDER BY proctime RANGE UNBOUNDED preceding) as var1
from T1
```

Output example

c (VARCHAR)	var1 (INT)
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
Hello	1
Hello World	7
Hello World	7
Hello World	7

2.11.7.5. LAST_VALUE

Syntax

```
T LAST_VALUE(value: T)
```

To return the last value in an ordered set of values, define the function as follows:

```
T LAST_VALUE(value: T, order: Long)
```

Arguments

- value: specifies a value of any type.
- order: specifies the order of the values. A value with the greatest order value is considered as the last value.

Description

Obtain the last data record of a stream.

Examples

Input example

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	"Hello"
2L	2	"Hello"
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"
20L	20	"Hello World"

Sample code

```
SELECT
c,
last_value(b) OVER (PARTITION BY c ORDER BY proctime RANGE UNBOUNDED preceding) as var1
from T1
```

Output example

c(VARCHAR)	var1(INT)
Hello	1
Hello	2

c(VARCHAR)	var1(INT)
Hello	3
Hello	4
Hello	5
Hello	6
Hello World	7
Hello World	8
Hello World	20

2.11.7.6. MAX

Syntax

```
MAX(A)
```

Arguments

A

- Supported data types include numeric values (TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE), BOOLEAN, and VARCHAR.
- The DATE, TIME, TIMESTAMP, and VARBINARY data types are not supported.

Description

Return the maximum value of all the input values.

Examples

Input example

var1(INT)
4
8

Sample code

```
SELECT MAX(a)  as aa
FROM T1;
```

Output example

aa(INT)
8

2.11.7.7. MIN

Syntax

```
MIN(A)
```

Arguments

A

- Supported data types include numeric values (TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE), BOOLEAN, and VARCHAR.
- The DATE, TIME, TIMESTAMP, and VARBINARY data types are not supported.

Description

Return the minimum value of all the input values.

Examples

Input example

var1(INT)
4
8

Sample code

```
SELECT MIN(a)  as aa
FROM T1;
```

Output example

aa(INT)
4

2.11.7.8. STDDEV_POP

Syntax

```
T STDDEV_POP(T value)
```

Argument

value: specifies a number of the BIGINT or DOUBLE type.

Description

Return the standard deviation for the population of all values in the specified expression.

Examples

Input example

a (DOUBLE)	c (VARCHAR)
0	Hi
1	Hi
2	Hi
3	Hi
4	Hi
5	Hi
6	Hi
7	Hi
8	Hi
9	Hi

Sample code

```
SELECT c, STDDEV_POP(a) as dou
FROM MyTable
GROUP BY c
```

Output example

c (VARCHAR)	dou1 (DOUBLE)
Hi	2.8722813232690143

2.11.7.9. SUM

Syntax

```
SUM(A)
```

Arguments

A: specifies a value of the numeric types including TINYINT, SMALLINT, INT, BIGINT, FLOAT, DECIMAL, and DOUBLE.

Description

Return the sum of all input values.

Examples

Input example

var1 (INT)
4
4

Sample code

```
SELECT sum(a)  as aa
FROM T1;
```

Output example

aa (INT)
8

2.11.7.10. VAR_POP

Syntax

```
T VAR_POP(T value)
```

Argument

value: specifies a number of the BIGINT or DOUBLE type.

Description

Return the statistical variance for the population of all values in the specified expression.

Examples

Input example

a (BIGINT)	c (VARCHAR)
2900	Hi

a (BIGINT)	c (VARCHAR)
2500	Hi
2600	Hi
3100	Hello
11000	Hello

Sample code

```
SELECT
   VAR_POP(a) as `result`,
   c
FROM MyTable
GROUP BY c
```

Output example

result (BIGINT)	c
28889	Hi
15602500	Hello

2.11.8. Other functions

2.11.8.1. UUID

Syntax

```
VARCHAR UUID()
```

Arguments

None

Description

Return a globally unique identifier.

Examples

Sample code

```
SELECT  uuid() as `result`
FROM T1
```

Output example

result(VARCHAR)
a364e414-e68b-4e5c-9166-65b3a153e257

2.12. UDXs

2.12.1. Overview

This topic describes how to build a development environment and use user-defined extensions (UDXs) in Realtime Compute for Apache Flink.

> **Notice**
>
> - Only Realtime Compute for Apache Flink in exclusive mode supports UDXs.
> - Blink is developed based on Apache Flink SQL by Alibaba Cloud Realtime Compute for Apache Flink to improve computing performance. UDXs can be used only in Blink.

UDX type

Realtime Compute for Apache Flink supports three types of UDXs, as described in the following table.

UDX type	Description
UDF	User-defined scalar function. The relationship between the input and output of UDFs is one-to-one mapping, which indicates that one value is returned each time a UDF reads one row of data.
UDAF	User-defined aggregation function. The relationship between the input and output of UDAFs is many-to-one mapping. A UDAF aggregates multiple input records into one output record. A UDAF can be used with the GROUP BY clause of SQL.
UDTF	User-defined table-valued function. When a UDTF is called, it generates multiple columns or rows of data.

Example

Realtime Compute for Apache Flink provides an example of a UDX to facilitate your business development. This example shows how to develop a UDF, UDAF, and UDTF.

> **Note**

- In this example, a development environment of the required version is configured. You do not need to build another development environment.
- The example provides Maven projects. You can use IntelliJ IDEA for development.
- Realtime Compute for Apache Flink V3.0

  Blink_UDX_3x
- Realtime Compute for Apache Flink V2.0

Blink_UDX_2x

- Realtime Compute for Apache Flink V1.0

  Blink_UDX_1x

Build a development environment

The development of UDXs depends on the following JAR packages. You can download the packages as required.

- Realtime Compute for Apache Flink versions earlier than V3.2.1
  - flink-streaming-java_2.11
  - flink-table_2.11
  - flink-core-blink-2.2.4
- Realtime Compute for Apache Flink V3.2.1 and later

  Add a POM dependency based on your open source software version. Download and view the example of a complete dependency.

  > **Note** If you want to use Snapshot, you can add a POM dependency based on your Snapshot version.

Register and use resources

1. Log on to the Realtime Compute for Apache Flink console.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure the resource parameters.

Parameter	Description
Location	You can upload JAR packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Select Resource Name	Enter a name for the resource.
Resource Description	Enter a description for the resource.
Authorization type	Select the type of the resource: JAR, DICTIONARY, or PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.
7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX at the beginning. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Types of parameters and return values

When you define Java UDXs in Realtime Compute for Apache Flink, you can use Java data types in parameters and return values. The following table lists the mappings between Realtime Compute for Apache Flink and Java data types.

Data type of Realtime Compute for Apache Flink	Java data type
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL	java.math.BigDecimal
BOOLEAN	java.lang.Boolean
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CHAR	java.lang.Character
STRING	java.lang.String
VARBINARY	java.lang.byte[]
ARRAY	Not supported
MAP	Not supported

Obtain parameters of UDXs

UDXs support an optional `open(FunctionContext context)` method. You can use `FunctionContext` to pass custom configuration items.

For example, you must add the following two parameters to your job:

```
testKey1=lincoln
test.key2=todd
```

The following example shows how to use `context.getJobParameter` in the open method to obtain parameters of a UDTF.

```
public void open(FunctionContext context) throws Exception {
      String key1 = context.getJobParameter("testKey1", "empty");
      String key2 = context.getJobParameter("test.key2", "empty");
      System.err.println(String.format("end open: key1:%s, key2:%s", key1, key2));
}
```

2.12.2. UDF

This topic describes how to build a development environment, write business logic code, and publish a user-defined scalar function (UDF) in Realtime Compute for Apache Flink.

> **Notice** Only Realtime Compute for Apache Flink in exclusive mode supports user-defined extensions (UDXs).

Definition

A UDF maps zero, one, or more scalar values to a new scalar value.

Build a development environment

For more information about how to build a development environment, see Build a development environment.

Write business logic code

To define a UDF, you must extend the ScalarFunction class by implementing the `eval` method. The `open` and `close` methods are optional.

> **Notice** UDFs return the same output for the same input by default. However, a UDF where an external service is called may return different output results even if the input values are the same. If a UDF cannot generate the same output for the same input, we recommend that you use the `override isDeterministic()` method to make it return `false`. Otherwise, the output may not meet your expectations in some cases. For example, a UDF operator moves forward.

The following sample code is written in Java:

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
    // The open method is optional.
    // To use the open method, you must add 'import org.apache.flink.table.functions.Functi
onContext;' to the code.
    @Override
    public void open(FunctionContext context) {
        }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    // The close method is optional.
    @Override
    public void close() {
        }
}
```

Write SQL statements

You can write SQL statements in a specified class. The following example shows the SQL statements in a UDX:

```
-- udf str.length()
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
create table sls_stream(
    a int,
    b int,
    c varchar
) with (
    type='sls',
    endPoint='<yourEndpoint>',
    accessKeyId='<yourAccessId>',
    accessKeySecret='<yourAccessSecret>',
    startTime = '2017-07-04 00:00:00',
    project='<yourProjectName>',
    logStore='<yourLogStoreName>',
    consumerGroup='consumerGroupTest1'
);
create table rds_output(
    id int,
    len bigint,
    content VARCHAR
) with (
    type='rds',
    url='yourDatabaseURL',
    tableName='<yourDatabaseTableName>',
    userName='<yourDatabaseUserName>',
    password='<yourDatabasePassword>'
);
insert into rds_output
select
    a,
    stringLengthUdf(c),
    c as content
from sls_stream;
```

Register and use resources

1. Log on to the Realtime Compute for Apache Flink console.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure the resource parameters.

Parameter	Description

Parameter	Description
Location	You can upload only JAR packages from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the size of the JAR package exceeds 300 MB, you must upload the package to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload the package.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a description for the resource.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.
7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX at the beginning. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Publish and use a UDF

Click **Publish** on the **Development** page for the job where you want to publish a UDF. Then, find the job on the **Administration** page and click **Start** in the Actions column to publish the UDF.

FAQ

Q: Why does the random number generator always generate the same value at runtime?

A: If no input parameters are passed to a UDX and you do not declare it as nondeterministic, the UDX may be optimized during compilation to return a constant value. To avoid this issue, you can use the `override isDeterministic()` method to make it return `false`.

2.12.3. UDAF

This topic describes how to build a development environment, write business logic code, and publish a user-defined aggregation function (UDAF) in Realtime Compute for Apache Flink.

Notice Only Realtime Compute for Apache Flink in exclusive mode supports user-defined extensions (UDXs).

Definition

A UDAF aggregates multiple values into a single value.

Methods of the UDAF abstract class

> **Note** A UDAF can be implemented in Java or Scala. However, we recommend that you use Java because Scala data types may cause unnecessary performance overhead.

The following code shows the core methods of the AggregateFunction class.

- createAccumulator and getValue methods

```
/*
* @param <T> The type of the output returned by a UDAF.
* @param <ACC> The accumulator type of a UDAF. An accumulator stores the intermediate agg
regation results of a UDAF. You can design an accumulator for each UDAF as required.
*/
public abstract class AggregateFunction<T, ACC> extends UserDefinedFunction {
/*
* Initialize the accumulator in AggregateFunction.
* The system calls the following method before it aggregates data for the first time.
*/
public ACC createAccumulator();
/*
* The system calls the following method after each aggregation is complete.
*/
public T getValue(ACC accumulator);
}
```

> **Note**
> - The createAccumulator and getValue methods can be defined in the AggregateFunction abstract class.
> - A UDAF must contain at least one accumulate method.

- accumulate method

```
public void accumulate(ACC accumulator, ...[user input]...);
```

> **Note**
> - You must implement an accumulate method to describe how to compute input data and update an accumulator to the aggregation result.
> - The first parameter of the accumulate method must be an accumulator of the ACC type defined in AggregateFunction. When the system is running, the runtime code sends the previous value in the accumulator and the specified upstream data to the accumulate method for aggregation. The upstream data can be of any type and can contain any number of data records.

- retract and merge methods

The createAccumulator, getValue, and accumulate methods can be used together to design a basic UDAF. However, Realtime Compute for Apache Flink also requires the retract and merge methods in some special scenarios.

In most scenarios, computing is early firing for an infinite stream. To refine early fired results, you can implement a retract method. The SQL optimizer automatically determines the conditions in which data needs to be retracted and the operations that are needed to process data marked with retract tags. You must implement a retract method to retract input data.

```
public void retract(ACC accumulator, ...[user input]...);
```

Note

- The retract method is the reverse operation of the accumulate method. For example, in a count UDAF, the number of data records in the computing result increases by one each time the accumulate method is called to process a data record, whereas the number of data records in the result decreases by one each time the retract method is called to process a data record.
- Similar to the accumulate method, the first parameter of the retract method must be an accumulator of the ACC type defined in AggregateFunction. When the system is running, the runtime code sends the previous value in the accumulator and the specified upstream data to the retract method for aggregation. The upstream data can be of any type and contain any number of data records.

Realtime Compute for Apache Flink requires the merge method in some scenarios. For example, if you use a session window to aggregate data, you must use the merge method. Realtime Compute for Apache Flink can process out-of-order data. Newly arrived data may fill the gap between two separate sessions, which results in the merge of the two sessions. In this case, you must use the merge method to integrate multiple accumulators into one accumulator.

```
public void merge(ACC accumulator, Iterable<ACC> its);
```

Note

- The first parameter of the merge method must be an accumulator of the ACC type defined in AggregateFunction. After the merge method is executed, the state data of AggregateFunction is stored in the first accumulator.
- The second parameter of the merge method is an iterator of one or more accumulators of the ACC type.

Build a development environment

For more information about how to build a development environment, see Build a development environment.

Write business logic code

The following Java code is an example:

```
import org.apache.flink.table.functions.AggregateFunction;
public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
    // Define the data schema of the accumulator that stores the state data of a count UDAF
.
    public static class CountAccum {
        public long total;
    }
    // Initialize the accumulator of the count UDAF.
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }
    // Call the getValue method to obtain the result of the count UDAF from the accumulator
that stores the state data of the count UDAF.
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }
    // Call the accumulate method to update the accumulator that stores the state data of t
he count UDAF based on the input data.
    public void accumulate(CountAccum accumulator, Object iValue) {
        accumulator.total++;
    }
    public void merge(CountAccum accumulator, Iterable<CountAccum> its) {
         for (CountAccum other : its) {
            accumulator.total += other.total;
         }
    }
}
```

Note The open and close methods are optional for a subclass of AggregateFunction. For more information, see the examples of UDF or UDTF.

Register and use resources

1. Log on to the Realtime Compute for Apache Flink console.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure resource parameters.

Parameter	Description

Parameter	Description
Location	You can upload Java Archive (JAR) packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a resource description.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.
7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Publish and use a UDAF

For more information about how to publish and use a UDAF, see Publish and use a UDAF.

Example

```
-- Use a UDAF to calculate the count.
CREATE FUNCTION countUdaf AS 'com.hjc.test.blink.sql.udx.CountUdaf';
create table sls_stream(
   a int,
   b bigint,
   c varchar
) with (
   type='sls',
   endPoint='yourEndpoint',
   accessKeyId='yourAccessId',
   accessKeySecret='yourAccessSecret',
   startTime='2017-07-04 00:00:00',
   project='<yourPorjectName>',
   logStore='stream-test2',
   consumerGroup='consumerGroupTest3'
);
create table rds_output(
   len1 bigint,
   len2 bigint
) with (
   type='rds',
   url='yourDatabaseURL',
   tableName='<yourDatabaseTableName>',
   userName='<yourDatabaseUserName>',
   password='<yourDatabasePassword>'
);
insert into rds_output
select
   count(a),
   countUdaf(a)
from sls_stream;
```

2.12.4. UDTF

This topic describes how to build a development environment, write code, and publish a user-defined table-valued function (UDTF) in Realtime Compute for Apache Flink.

> **Note** Only Realtime Compute for Apache Flink in exclusive mode supports user-defined extensions (UDXs).

Definition

Similar to a user-defined scalar function (UDF), a UDTF uses zero, one, or multiple scalar values as input parameters (including variable-length parameters). Different from a UDF, a UDTF returns any number of rows, rather than a single value. The returned rows can consist of one or more columns.

Build a development environment

For more information, see Build a development environment.

Write code that implements business logic

A UDTF needs to implement the eval method in the TableFunction class. The open and close methods are optional. The following Java code is an example:

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
public class SplitUdtf extends TableFunction<String> {
    // The open method is optional. To use the open method, you must add 'import org.apache
.flink.table.functions.FunctionContext;' to the code.
    @Override
    public void open(FunctionContext context) {
        // ... ...
        }
    public void eval(String str) {
        String[] split = str.split("\\|");
        for (String s : split) {
            collect(s);
        }
    }
    // The close method is optional.
    @Override
    public void close() {
        // ... ...
        }
}
```

Return multiple rows

A UDTF can convert the output result from a single row to multiple rows by calling the `collect` method multiple times.

Return multiple columns

A UDTF can also convert the output result from a single column to multiple columns. If you want a UDTF to return multiple columns, declare the return value as a tuple or row. The following examples show how to declare a return value as a tuple and how to declare a return value as a row.

- Declare the return value as a tuple.

  Realtime Compute for Apache Flink supports Tuple1 to Tuple25, which define 1 to 25 fields. The following example is a UDTF that uses Tuple3 to return three fields:

```
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.table.functions.TableFunction;
// If the return value is declared as a tuple, you must explicitly declare the generic ty
pes of the tuple, such as STRING, LONG, and INTEGER in this example.
public class ParseUdtf extends TableFunction<Tuple3<String, Long, Integer>> {
public void eval(String str) {
String[] split = str.split(",");
// The following code is used for reference only. In actual scenarios, you must add code
that implements verification logic.
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Tuple3<String, Long, Integer> tuple3 = Tuple3.of(first, second, third);
collect(tuple3);
}
}
```

Note If the return value is declared as a tuple, column values cannot be null and a maximum of 25 columns are allowed.

- Declare the return value as a row.

  For example, you can enable this feature to return three columns. The sample code is an example.

```
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
public class ParseUdtf extends TableFunction<Row> {
public void eval(String str) {
String[] split = str.split(",");
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Row row = new Row(3);
row.setField(0, first);
row.setField(1, second);
row.setField(2, third);
collect(row);
}
@Override
// If the return value is declared as a row, you must overload the getResultType method t
o explicitly declare the data type of the return value.
public DataType getResultType(Object[] arguments, Class[] argTypes) {
return DataTypes.createRowType(DataTypes.STRING, DataTypes.LONG, DataTypes.INT);
}
}
```

Note If the return value is declared as a row, the field value can be null. However, you must overload the `getResultType` method.

SQL syntax

A UDTF supports CROSS JOIN and LEFT JOIN. When you use a UDTF, you must add the keywords `LATERAL` and `TABLE`. You must also specify an alias for the UDTF, such as `ParseUdtf` in the preceding code.

```
CREATE FUNCTION parseUdtf AS 'com.alibaba.blink.sql.udtf.ParseUdtf';
```

- cross join

  Each row in the left table is joined with a row of data that is generated by the UDTF. If the UDTF does not generate any data for a row, the row is not returned.

  ```
  select S.id, S.content, T.a, T.b, T.c
  from input_stream as S,
  lateral table(parseUdtf(content)) as T(a, b, c);
  ```

- left join

  Each row in the left table is joined with a row of data that is generated by the UDTF. If the UDTF does not generate any data for a row, the UDTF fields in the row are filled with null.

  > **Note** A LEFT JOIN statement that uses a UDTF must end with `on true`.

  ```
  select S.id, S.content, T.a, T.b, T.c
  from input_stream as S
  left join lateral table(parseUdtf(content)) as T(a, b, c) on true;
  ```

Register and use resources

1. Log on to the Realtime Compute for Apache Flink console.
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane, click the **Resources** tab.
4. In the upper-right corner of the **Resources** pane, click **Create Resource**.
5. In the **Upload Resource** dialog box, configure resource parameters.

Parameter	Description
Location	You can upload JAR packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.

Parameter	Description
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a resource description.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

6. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.
7. In the drop-down list, select **Reference**.
8. In the code editor, declare the UDX at the beginning. The following statement is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

Publish and use a UDTF

For more information about how to publish and use a UDTF, see Publish and use a UDTF.

Example

```
-- UDTF str.split("\\|");
create function splitUdtf as 'com.hjc.test.blink.sql.udx.SplitUdtf';
create table sls_stream(
a INT,
b BIGINT,
c VARCHAR
) with (
type='sls',
endPoint='yourEndpoint',
accessKeyId='yourAccessKeyId',
accessKeySecret='yourAccessSecret',
startTime = '2017-07-04 00:00:00',
project='yourProjectName',
logStore='yourLogStoreName',
consumerGroup='consumerGroupTest2'
);
-- Use the splitUdtf function to extract data from the c field. Table T(s) with multiple ro
ws and one column is returned. s is the name of the column.
create view v1 as
select a,b,c,s
from sls_stream,
lateral table(splitUdtf(c)) as T(s);
create table rds_output(
id INT,
len BIGINT,
content VARCHAR
) with (
type='rds',
url='yourDatabaseURL',
tableName='yourDatabaseTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);
insert into rds_output
select
a,b,s
from v1;
```

2.12.5. Develop a UDX by using IntelliJ IDEA

This topic describes how to develop a user-defined extension (UDX) in Realtime Compute for Apache Flink by using IntelliJ IDEA. The development process includes building a development environment and referencing a UDX in a Realtime Compute for Apache Flink job.

Context

Notice

- Only Realtime Compute for Apache Flink in exclusive mode supports UDXs.
- We recommend that you use IntelliJ IDEA to develop a UDX.

Configure Maven

1. Download a Maven installation package.
   i. Go to the download page at the Apache Maven official website.
   ii. Download apache-maven-3.5.3-bin.tar.gz.
   iii. Decompress the downloaded package to a specified directory, such as */Users/<userName>/Documents/maven*.
2. Configure environment variables.
   i. In the command terminal, run the `vim ~/.bash_profile` command.
   ii. Add the following commands to the .bash_profile file:

   ```
   export M2_HOME=/Users/<userName>/Documents/maven/apache-maven-3.5.3
   export PATH=$PATH:$M2_HOME/bin
   ```

   iii. Save and close the .bash_profile file.
   iv. Run the `source ~/.bash_profile` command for the configuration to take effect.
3. Run the `mvn -v` command to check whether the configuration takes effect.

   If information similar to the following information is displayed, the configuration takes effect:

   ```
   Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-04T03:39:06+08:00
   )
   Maven home: /Users/<userName>/Documents/maven/apache-maven-3.5.0
   Java version: 1.8.0_121, vendor: Oracle Corporation
   Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/jre
   Default locale: zh_CN, platform encoding: UTF-8
   OS name: "mac os x", version: "10.12.6", arch: "x86_64", family: "mac"
   ```

Build a development environment

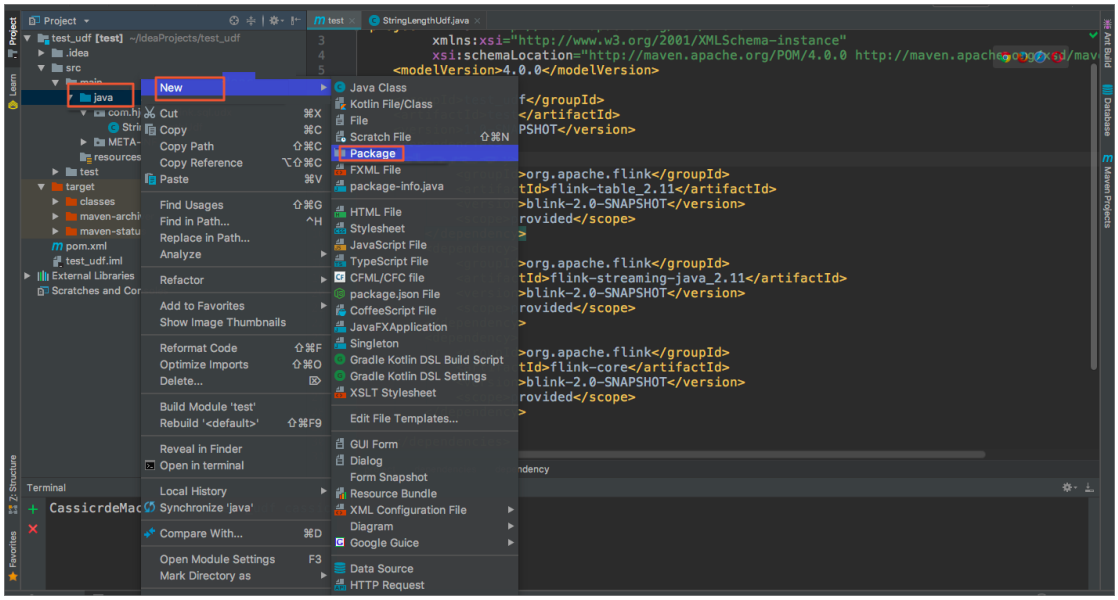
1. Download the UDX demo.
2. Decompress the downloaded package in a Linux operating system.

   ```
   tar xzvf RealtimeCompute-udxDemo.gz
   ```

3. Open **IntelliJ IDEA** and click **Open** to open the demo.

Reference a JAR package in a job

1. Create a package.
   i. Right-click **java** and choose **New > Package**.



   ii. In the **New Package** dialog box, enter a package name. In this example, a package named `com.hjc.test.blink.sql.udx` is created.
   iii. Click **OK**.
2. Create a class.
   i. Right-click **com.hjc.test.blink.sql.udx** and choose **New > Java Class**.

ii. In the **Create New Class** dialog box, enter a class name.

> **Note** Retain the default value **Class** of **Kind**.

iii. Click **OK**.

3. Enter the following code in the class:

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
    // The open method is optional.
    // To write the open method, you must add 'import org.apache.flink.table.functions.
FunctionContext;' to the code.
    @Override
    public void open(FunctionContext context) {
        }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    // The close method is optional.
    @Override
    public void close() {
        }
}
```

4. In the command terminal, run the `mvn package` or `mvn assembly:assembly` command to add the project to the JAR package.

> **Note**
>
> - If you need to add a third-party dependency to the JAR package, run the `mvn assembly:assembly` command.
> - The compiled JAR package is RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT.jar or RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT-jar-with-dependencies.jar. It contains a third-party dependency.

5. Reference the JAR package in a Realtime Compute for Apache Flnik job.
   i. Log on to the Realtime Compute for Apache Flink console.
   ii. In the top navigation bar, click **Development**.
   iii. In the left-side navigation pane, click the **Resources** tab.

iv. In the upper-right corner of the **Resources** pane, click **Create Resource**.

Parameter	Description
Location	You can upload JAR packages only from your on-premises machine in the Realtime Compute for Apache Flink console. **Note** The maximum size of a JAR package that can be uploaded from your on-premises machine is 300 MB. If the JAR package exceeds 300 MB, you must upload it to the Object Storage Service (OSS) bucket that is bound to your cluster or use an API to upload it.
Resource	Click **Upload Resource** to select the resource that you want to reference.
Resource Name	Enter a name for the resource.
Resource Description	Enter a resource description.
Resource Type	Select the type of the resource. Valid values: JAR, DICTIONARY, and PYTHON.

v. In the **Resources** pane, find the new resource, and move the pointer over **More** in the Actions column.

vi. In the drop-down list, select **Reference**.

vii. In the code editor, declare the UDX at the beginning. The following code is an example:

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```